Reconocimiento Automático de Matrículas
Software *RecAM*

Rodrigo Abal – Nicolás Pebet – Raúl Medeglia

Octubre, 2004

Universidad de la República
Facultad de Ingeniería
Instituto de Ingeniería Eléctrica

Proyecto de Fin de Carrera

Reconocimiento Automático de Matrículas

Software *RecAM*

Estudiantes

Rodrigo Abal
Nicolás Pebet
Raúl Medeglia

Tutor

Alvaro Pardo

Universidad de la República
Facultad de Ingeniería
Instituto de Ingeniería Eléctrica

Octubre, 2004

Agradecimientos

Queremos agradecer al Dr. Alvaro Pardo por su directa supervisión del proyecto y su invaluable conocimiento sobre procesamiento de señales e imágenes. También queremos agradecer a todas las personas que de una forma u otra han colaborado, y especialmente al Dr. Charles Galambos por su rápida respuesta a nuestras consultas sobre RAVL.

A nuestras familias por su paciencia y su aliento, ya que sin su apoyo este trabajo no hubiera sido posible. Quisiéramos también agradecer a todos nuestros amigos, por su invalorable apoyo.

Resumen

El software RecAM, es un proyecto de grado de la carrera de Ingeniería Eléctrica opción Telecomunicaciones de la Facultad de Ingeniería, Universidad de la República.

Consiste en reconocer la matrícula de un automóvil en forma automática, a partir de una imagen del frente del vehículo, devolviendo en un string de caracteres ASCII, el número de la matrícula.

En la presente documentación se exponen aspectos teóricos y descripción de los algoritmos implementados en el desarrollo de RecAM, así como ejemplos visuales de funcionamiento a nivel de usuario. También se hace un repaso de los paquetes de software requeridos para poder ejecutar RecAM correctamente.

Por último, se presentan algunos ensayos realizados con varias imágenes de automóviles de distintas características, analizando sus resultados, y se discuten las dificultades encontradas en el proceso de desarrollo.

Índice General

Índice de Figuras

Capítulo 1

1. Introducción

Actualmente es común que una persona al entrar en un estacionamiento tarifado o privado, retire un ticket o inserte una tarjeta de acceso especial para ingresar. Esto le permitirá retirarse más tarde o abonar el importe correspondiente según el tiempo que haya permanecido dentro.

Otra forma de permitir o controlar el acceso es mediante la lectura y el posterior reconocimiento automático de la matrícula del vehículo. En este caso, el número reconocido puede ser almacenado en una base de datos junto con la hora de entrada y cuando el usuario se retira, el número es leído nuevamente, pudiéndose calcular la tarifa, utilizando la hora de entrada y salida.

Este tipo de sistema podría ser utilizado también para cobro de peajes para eliminar congestionamientos, corroboración de pago de patentes, identificación de vehículos que hayan cometido infracciones de tránsito, etc.

Este procedimiento tiene las ventajas de facilitar y agilizar la tarea de un operador, además de evitar errores causados por fatiga. Adicionalmente, el equipamiento necesario para esta tecnología es muy simple y de bajo costo. El sistema completo estaría compuesto por un sistema de adquisición de imágenes (cámara interactuando con un sensor), y un PC medianamente potente con una base de datos.

El presente proyecto tiene como objetivo el desarrollo de un software para el reconocimiento automático de las matrículas. El punto de partida es una imagen del frente del automóvil, obtenida con una cámara digital utilizando mínima compresión, simulando así el procedimiento de adquisición de las imágenes. De ésta manera, las diferentes fases necesarias para llegar al reconocimiento final son: localización de la matrícula, procesamiento (binarización, corrección

de la inclinación, eliminación de bordes), segmentación de caracteres, y el reconocimiento de cada caracter (OCR). Todo esto forma parte del software RecAM.

Capítulo 2

2. Objetivos y Requerimientos

El objetivo general del Proyecto, como ya se mencionó, es mediante la automatización de procesos manuales, optimizar algunas funciones que hasta ahora son cumplidas por personas. Se implementará un sistema de reconocimiento automático del número de una matrícula utilizando técnicas de tratamiento de imágenes. Dicho sistema consiste en un paquete de software implementado en C++ que corre sobre Linux. El mismo deberá cumplir con ciertos requisitos:

- Tiempo de reconocimiento menor a 1 segundo.
- Extracción correcta del número de matrícula en un porcentaje alto de casos (más de 70%).
- Robustez, ya sea frente a las condiciones de luminosidad, suciedad, nitidez, o ruido.
- Tolerancia a variaciones en la distancia de la cámara al vehículo.
- Tolerancia al ángulo de inclinación de la matrícula.

Los posibles interesados en este Proyecto son aquellas personas que deseen automatizar el proceso de admisión a ciertos establecimientos, y/o supervisar a los operadores evitando posibles fraudes.

Para poder cumplir con los objetivos planteados, y ya que el proceso de adquisición no está comprendido dentro de las especificaciones de este Proyecto, se simula el sistema de adquisición de la imagen tomando varias fotografías de vehículos con una cámara digital.

Sobre la imagen de entrada se ejecutan algoritmos de tratamiento de imágenes, los cuales segmentarán la matrícula del fondo. Luego, a la imagen segmentada se le aplican algoritmos de reconocimiento de caracteres (OCR), los cuales devuelven un string de caracteres.

Por tanto el sistema está formado por tres grandes módulos. El primero, encargado de adquirir la imagen (en el presente Proyecto, se carga desde un archivo), el segundo debe ser lo más robusto posible y se encarga de segmentar la matrícula del resto de la imagen, y el tercero se encarga del reconocimiento, OCR (Optical Character Recognition).

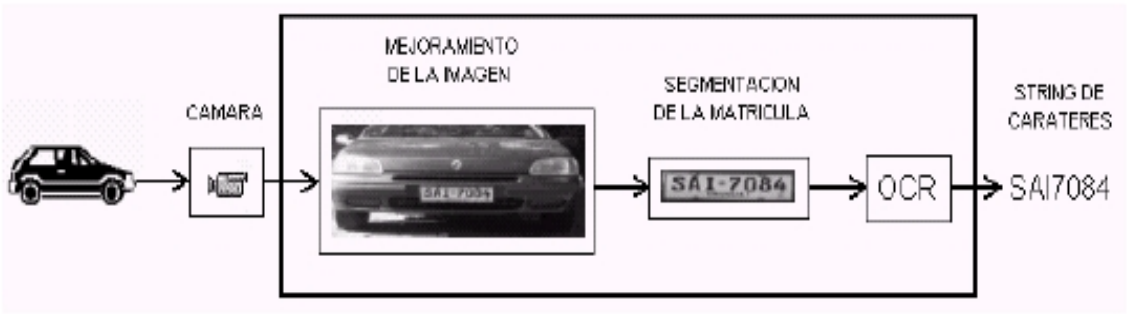


Fig. 2.1 Diagrama del sistema

El motor de reconocimiento desarrollado en este Proyecto puede ser utilizado en distintos tipos de sistemas, a modo de ejemplo se muestra un diagrama de lo que podría ser un sistema de control de acceso.

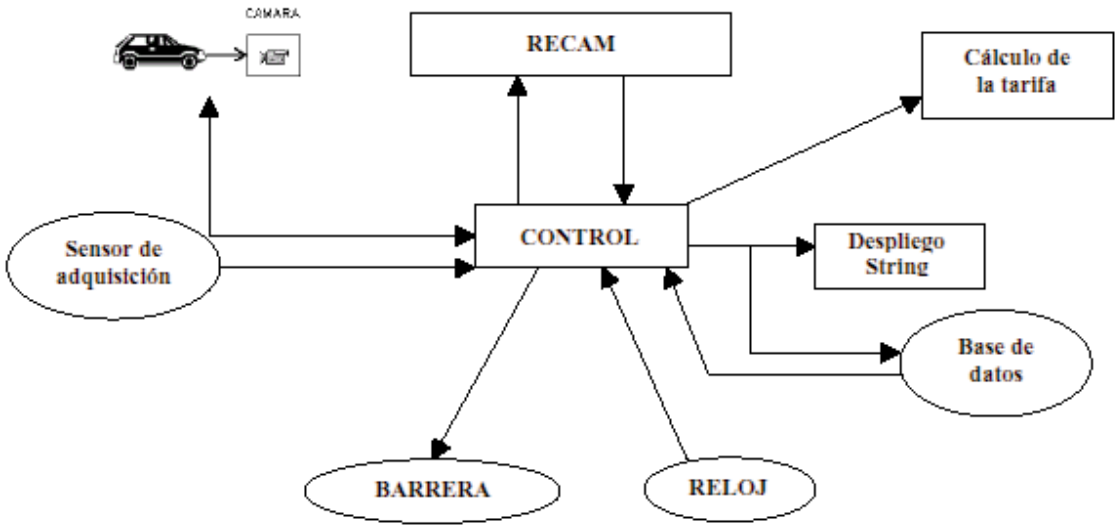


Fig. 2.2 Sistema de control de acceso

El proceso natural del sistema ante la llegada de un vehículo es el siguiente:

1. Se activa el sensor de adquisición notificando al bloque de control la presencia de un vehículo.
2. Control indica a la cámara que obtenga la imagen del frente del vehículo.
3. El bloque de control obtiene la foto y la pasa a RecAM.
4. La imagen es procesada por el bloque RecAM que devuelve un string con el número de la matrícula al bloque de control.
5. Control despliega el string, pudiendo éste tener algunos símbolos como ? donde debería estar el supuesto número de la matrícula. Ésto sucede cuando no se pudo reconocer al símbolo.
   a) Si el string contiene ? (error), se regresa al paso 2. Ésto puede suceder un número previamente establecido de veces.
   b) Si el string contiene el número de matrícula, éste se almacena en una base de datos, junto con la hora (leída por el bloque controlador de un reloj en tiempo real).
6. En el caso que el string volviese a contener un mensaje de error, se despliega una alarma y se aborta el proceso.
7. El bloque de control levanta y baja la barrera de paso.

El proceso ante la salida de un vehículo es el siguiente:

1. Se activa el sensor de adquisición notificando al bloque de control la presencia de un vehículo.
2. Control indica a la cámara que obtenga la imagen del frente del vehículo.
3. El bloque de control obtiene la foto y la pasa a RecAM.
4. La imagen es procesada por el bloque RecAM que devuelve un string con el número de la matrícula al bloque de control.
5. Control despliega un string, pudiendo éste tener algunos símbolos como ? donde debería estar el supuesto número de la matricula.
   a) Si el string contiene ? (error), se regresa al paso 2. Ésto puede suceder un número previamente establecido de veces.
   b) Si el string contiene el número de matrícula, éste se almacena en una base de datos, junto con la hora (leída por el bloque controlador de un reloj en tiempo real)
6. En el caso que el string volviese a contener un mensaje de error, se despliega una alarma y se aborta el proceso.
7. El bloque controlador, obtiene los valores almacenados para éste número de matrícula y calcula la tarifa a pagar.
8. Se realiza el cobro.
9. Control levanta y baja la barrera.

Para la implementación se consideraron los siguientes requerimientos:

- Distancia fija entre la cámara y el vehículo, con cierto margen de tolerancia.
- Vehículo quieto.
- Chapa con dígitos oscuros sobre fondo claro (si se supiera de antemano que la matrícula tiene dígitos claros sobre fondo oscuro, se debería invertir los tonos de gris de la imagen antes de procesarla).
- Imágenes obtenidas con luz ambiental (sin flash) y en niveles de gris.
- Matrículas relativamente limpias y en buen estado (las nuevas, en Montevideo, son de aluminio, lo cual hace que se preserven mejor).
- Tolerancia frente a la inclinación de la matrícula: ±10 grados.
- Imágenes obtenidas del frente del vehículo.

Capítulo 3

3. Aspectos teóricos y Descripción de los algoritmos

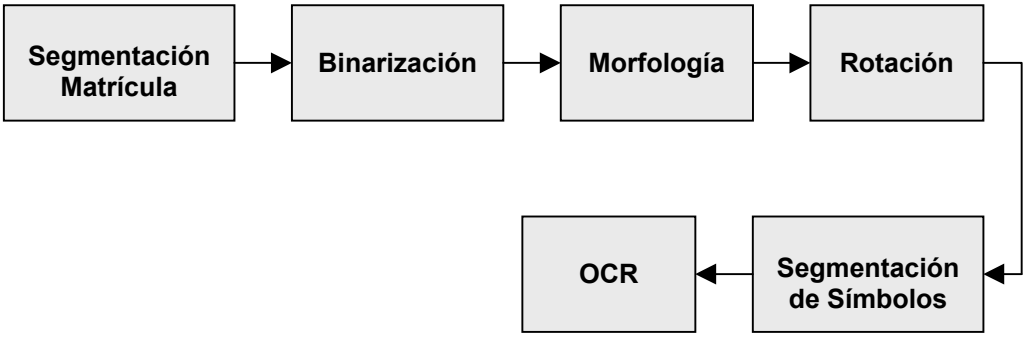


Fig. 3.1 Diagrama en bloques del sistema

3.1 Segmentación de la matrícula y binarización

Dentro del procedimiento del reconocimiento, es especialmente importante aislar la matrícula del resto de la imagen. Para esto, se dispone de cierta información a priori, caracteres oscuros sobre fondo blanco y distancia de la cámara a la matrícula constante dentro de un posible rango de variación. Aprovechando esta información, se puede identificar la ubicación de la matrícula identificando un patrón que claramente identifique los dígitos.

El patrón mencionado se obtiene de aplicar el operador Sobel (sobre la imagen original) en la dirección horizontal. El resultado es una imagen con los gradientes horizontales, en la cual en las transiciones de claro a oscuro, aparece un pico positivo y en las de oscuro a claro, uno negativo. Se clasifica este resultado de acuerdo a tres posibles estados, para ello se fijan dos umbrales iguales y opuestos. Se impuso que dicho umbral se calculara fijando un porcentaje de 5%

sobre el histograma de gradiente en valor absoluto, esto es: superarán el umbral el 5% de los valores. Entonces, si se encuentra un pico positivo seguido de uno negativo y separados no más de una cierta distancia, se está ante un trazo negro sobre fondo blanco.

Un corte horizontal de una matrícula define un patrón de trazos. Si se es capaz de encontrar este patrón repetido en varias filas consecutivas, se podrá deducir que allí se encuentra la matrícula.

A continuación se ilustra un ejemplo de esto:

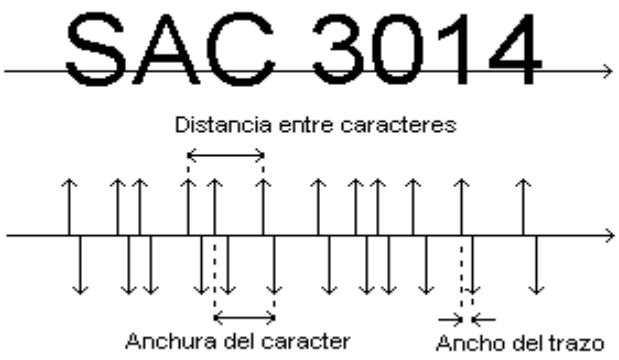


Fig. 3.2 Patrón buscado

El algoritmo de segmentación, utiliza algunos parámetros:

- Ancho de la matrícula.
- Margen de variación del ancho de la matrícula.
- Ancho del trazo de los caracteres.
- Separación máxima de los caracteres.
- Se tiene en cuenta además que para clasificar un patrón de trazos como una posible matrícula, debe contener un mínimo y un máximo de trazos.

Con lo anterior en cuenta se recorre la imagen del gradiente en busca de patrones repetidos en varias filas consecutivas.

Cabe aclarar que se eligió el operador Sobel, "un clásico en la literatura", por su eficiencia. Como solamente son necesarios los bordes verticales, esto implica un ahorro de operaciones a la hora de compararlo con otros detectores de borde que detectan todos los bordes, por ejemplo el filtro de J. F. Canny [10], el cual claramente da un mejor resultado, pero requiere un mayor tiempo de ejecución (esto presenta un problema a la hora de hacer una implementación en tiempo real). También están siendo muy utilizados los detectores de borde que utilizan métodos variacionales, por ejemplo la detección con contornos activos utilizando EDP's, pero es impensable utilizarlos

ya que para converger son necesarias muchas iteraciones de pasos pequeños, si no, podría diverger.

Luego de tener segmentada la matrícula, se procede a binarizar la misma. Para ello se utiliza nuevamente el operador Sobel pero ahora el umbral se fija en un porcentaje de 15% sobre el histograma de gradiente en valor absoluto (superarán el umbral el 15% de los valores). Una vez que se obtiene la respuesta del filtro Sobel, se define un nuevo umbral para decidir finalmente cuales píxeles serán blancos y cuales negros (binarización). Debido a la diversidad de matrículas, a la iluminación, etc., ese umbral debería ser *adaptivo*.

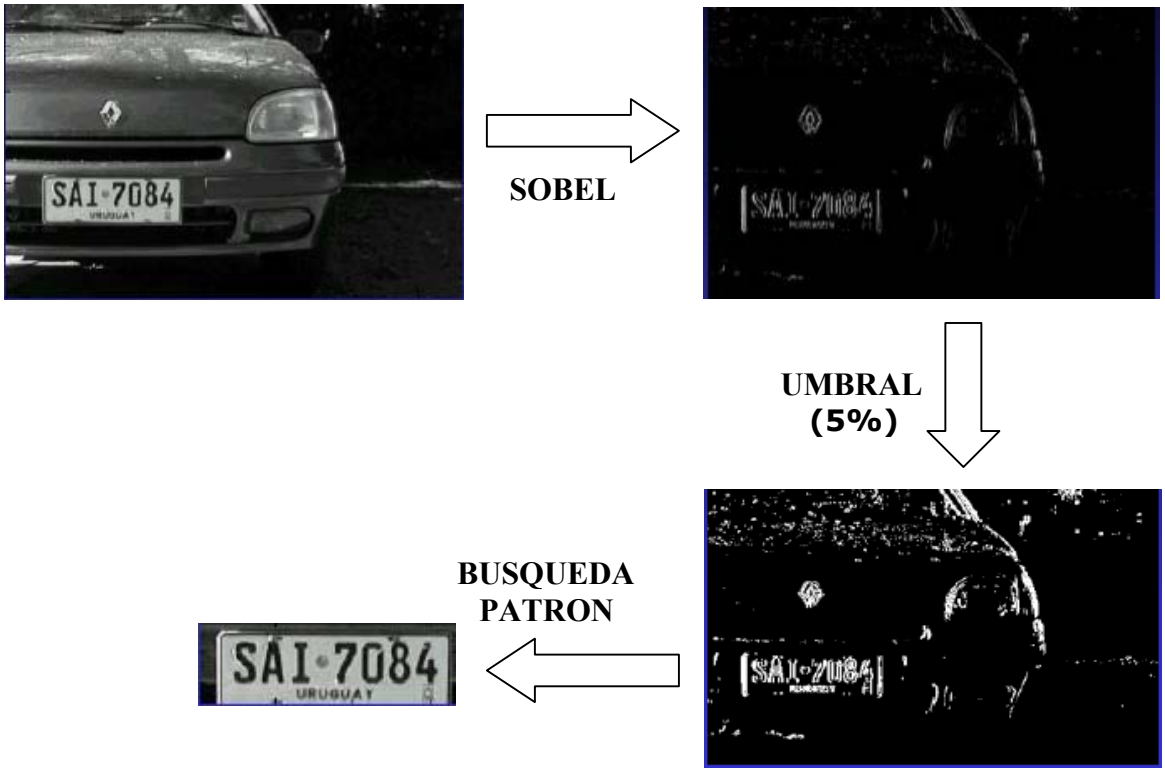
El problema radica en decidir cual es el umbral óptimo para cada caso. En la publicación realizada por F. Martín y X. Fernández [3], se propone el umbral:

$$Umbral = \frac{\mu_N \cdot N_B + \mu_B \cdot N_N}{N_N + N_B}$$

Siendo μ_N la media de las intensidades de los píxeles negros, μ_B la de los píxeles blancos, N_N es el número de píxeles negros y N_B el de los blancos. Se realizaron pruebas utilizando este umbral obteniéndose muy buenos resultados, por lo que finalmente se decidió utilizarlo.

Tanto el valor de la media de las intensidades de píxeles blancos o negros como la cantidad de píxeles negros o blancos surgen del conocimiento a priori que se tiene de la matrícula, es decir, dígitos negros sobre fondo blanco.
Se empieza contando los píxeles en la primer fila de la respuesta del filtro de Sobel hasta que se encuentra con un pico, si por ejemplo este fuera positivo, allí se está en una transición de claro a oscuro, y por lo tanto, todos los píxeles antes contados son blancos, siguiendo con este procedimiento se empieza a contar hasta que se encuentra el siguiente pico (que debe ser negativo), entonces todos los píxeles contados habrán sido negros. Esto se repite hasta recorrer toda la matrícula. De esta manera se obtiene la cantidad de píxeles que son blancos y los que son negros, también se fueron acumulando tanto las intensidades de los blancos como las de los negros pudiéndose ahora calcular las medias, las probabilidades y con ellas el umbral óptimo.

A modo de ejemplo, el procedimiento de segmentación de la matrícula es el siguiente:



Y el posterior proceso de binarizado, se ejemplifica a continuación:

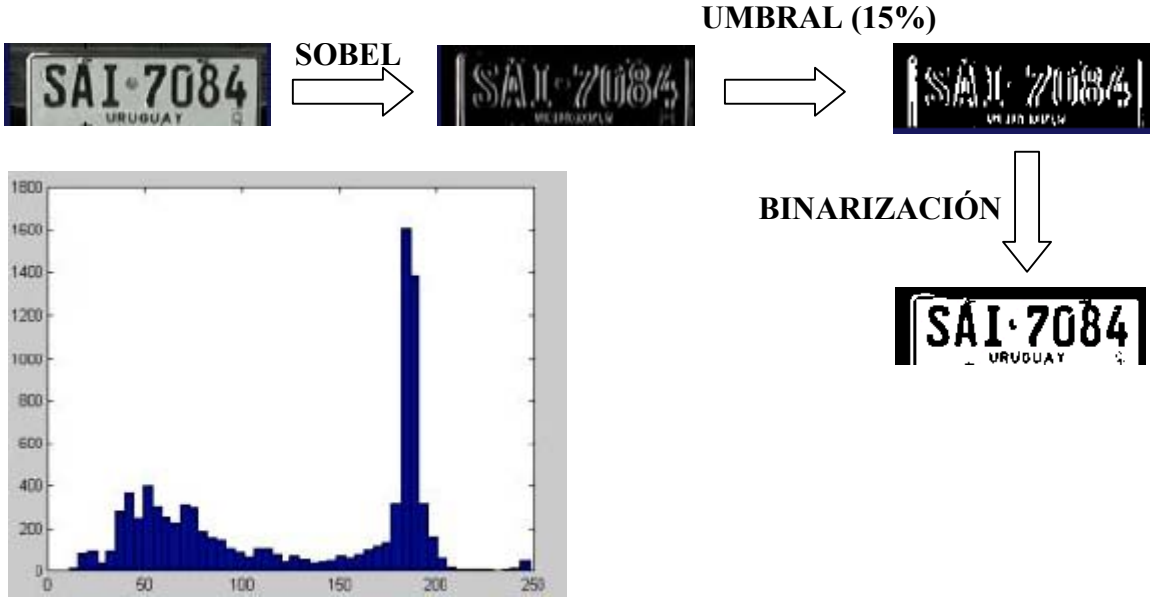


Fig. 3.3 Histograma de la matricula.

El umbral para la binarización calculado por RecAM es 120,23.

3.2 Morfología y rotación de la matrícula

Para que el reconocimiento de la matrícula sea efectivo, es necesario que la misma se encuentre horizontal.

Al sacar una foto del automóvil pueden suceder dos cosas:

1) Que la foto esté inclinada un cierto ángulo con respecto al eje horizontal, con lo cual la matrícula también aparecerá inclinada.

2) Que la foto esté horizontal pero la matrícula presente un cierto ángulo de inclinación, por ejemplo, si se encontrara torcida en el automóvil.

En cualquiera de los dos casos anteriores, se debe corregir la inclinación de la matrícula. Para esto es necesario calcular el ángulo de inclinación que presenta, para posteriormente proceder a la rotación de la misma.

Este método comienza a partir de que se tiene la matrícula segmentada de la foto del vehículo.

Para elegir el método adecuado, se realizó un estudio previo sobre las características que presentan algunas matrículas segmentadas en distintas circunstancias como por ejemplo, fotos horizontales, inclinadas, más alejadas o más cercanas, etc.

Fig. 3.4 Fotos de matrículas

Como se puede observar en las imágenes anteriores, los caracteres de la matrícula siempre están visibles pero los bordes de la matrícula estarán visibles dependiendo de la posición.

Por dicho motivo, se eligió un método para obtener el ángulo de rotación a partir de las letras y números de la matrícula (sin tener en cuenta los caracteres del país, URUGUAY en este caso).

A continuación se explica en detalle el método con el cual se obtuvieron los mejores resultados y por ende el utilizado. Luego se muestran otros métodos evaluados para obtener el ángulo que fueron descartados debido a su pobre performance.

3.2.1 Centroide o Baricentro y Mínimos Cuadrados

Este método es el método utilizado en el presente Proyecto.

Consiste en calcular el centroide o baricentro de cada caracter de la matrícula. Mediante mínimos cuadrados se calcula la ecuación de la recta que pasa por todos los centroides, y el ángulo que forma ésta con el eje horizontal es el ángulo de rotación buscado.

Los pasos efectuados a partir de la matrícula binarizada para calcular el ángulo de rotación se describen en Fig. 3.5.

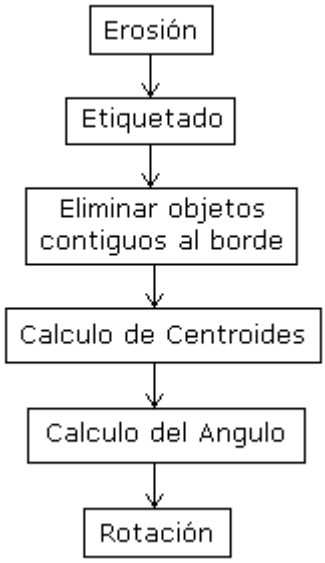


Fig. 3.5 Diagrama de Bloques para la rotación

Erosión

Se realiza erosión para eliminar posibles objetos aislados considerados como ruido sobre la matrícula binarizada. También sirve para separar objetos, como por ejemplo, símbolos unidos por un tornillo. De esta forma, mediante un algoritmo de etiquetado se logra etiquetar símbolos distintos con etiquetas distintas.

Para la erosión se utilizó un elemento estructural circular, en éste caso, de conectividad de 4 vecinos (los dos verticales y los dos horizontales respecto al pixel examinado).

Se realizó erosión de regiones negras consistiendo en lo siguiente:

Para cada píxel de la matrícula binarizada (donde el objeto es negro y el fondo blanco), si el píxel es negro y alguno de los vecinos es blanco, entonces el píxel se cambia a blanco. De lo contrario, queda negro.

Etiquetado

Se etiquetan todos los objetos presentes en la matrícula con el fin de individualizarlos y poder acceder a cada uno por separado.

El número máximo de etiquetas es variable y depende de la cantidad de objetos presentes en la matrícula erosionada.

La implementación del etiquetado es la siguiente:

Se considera una variable *maxEtiqueta* que indica el total de etiquetas que tiene la imagen.

Se recorre la imagen comparando cada píxel actual (i,j) con los 4 vecinos (i,j-1), (i-1,j-1), (i-1,j), (i-1,j+1).

Cuando se encuentra un píxel vecino igual al píxel actual, se etiqueta el píxel actual con la ***misma etiqueta*** que tiene su vecino. Si no se encuentra un píxel igual, se incrementa la variable *maxEtiqueta* y se asigna una nueva etiqueta al píxel actual.

En el caso en que los píxeles actual y vecino sean iguales pero tengan ***diferente etiqueta***, se implementa el siguiente algoritmo:

Se consideran dos variables:

etMax = máx(etiqueta actual, etiqueta vecino)
etMin = mín(etiqueta actual, etiqueta vecino)

1) Se recorren todos los píxeles por encima de la fila i del píxel actual
    a) A todos los píxeles con etiqueta igual a *etMax* se les cambia la etiqueta a *etMin.*
    b) A todos los píxeles con etiqueta mayor a *etMax* se les decrementa en 1 el valor de la etiqueta.
2) Se recorren todos los píxeles de toda la fila a la izquierda del píxel actual incluyendo al píxel actual
    a) A todos los píxeles con etiqueta igual a *etMax* se les cambia la etiqueta a *etMin.*
    b) A todos los píxeles con etiqueta mayor a *etMax* se les decrementa en 1 el valor de la etiqueta.
3) Se decrementa *maxEtiqueta* en 1.
4) El fondo queda con etiqueta 1 y cada objeto con etiquetas mayores a 1.
    a) Se elimina etiqueta 1 del fondo dejándolo con etiqueta igual a 0.
    b) Se decrementa en 1 las etiquetas de cada objeto.

Eliminar objetos contiguos al borde

Aquí son eliminados todos los objetos que pertenecen a cualquiera de los 4 bordes de la imagen.

La implementación es la siguiente:

Si hay alguna etiqueta perteneciente al borde de la imagen etiquetada:

- se elimina el objeto correspondiente a dicha etiqueta.
- se decrementa en 1 las etiquetas de los demás objetos.

La etiqueta de los objetos es utilizada como referencia para saber cuantos objetos hay.

Cálculo de los centroides

El objetivo aquí es calcular los centroides de cada uno de los símbolos etiquetados que tienen un área mayor que un área mínima dada. Para ello cada símbolo es dilatado tres veces con una máscara de vecindad 8 antes de calcular el centroide.

Se utilizó un área mínima para eliminar todos aquellos objetos que no son símbolos y no fueron eliminados mediante erosión. De esa manera sólo quedan los objetos que son símbolos.

El área mínima se eligió tomando como referencia la letra I erosionada porque es el caracter que tiene menos área (0.9% de la imagen total). Por consiguiente, se puede elegir cualquier porcentaje menor. Para obtener los mejores resultados se escogió 0.8% de la imagen total.

El procedimiento para el cálculo de los centroides es el siguiente:

A partir de la imagen etiquetada sin bordes, se conoce la cantidad de objetos.

Pseudo código:

1) *Etiqueta* =1

2) De la imagen etiquetada se obtiene el objeto correspondiente. De esta forma se llega a una imagen binarizada del mismo tamaño de la matrícula con un objeto solo, donde el objeto es blanco y el fondo es negro.

3) Se calcula el área de dicho objeto contando la cantidad de píxeles blancos. Se verifica si esta área es mayor o menor que el área mínima. En caso de que sea menor, se descarta el objeto, se incrementa la *Etiqueta* en 1 y se vuelve al paso 2. En caso contrario se pasa al siguiente paso.

4) Se dilata 3 veces el símbolo obteniendo una imagen binarizada del mismo tamaño que la matrícula con un símbolo solo y dilatado.

5) Se calcula el centroide de dicho símbolo.

6) Se incrementa la *Etiqueta* en 1 y se vuelve al paso 2.

En el caso que los símbolos sean distintos, sus formas pueden ser muy diferentes, lo cual puede conllevar a que las posiciones de los centroides estén mas dispersas respecto de la recta que pasa por todos los centroides.
Para que dicha dispersión sea menor, se dilatan los símbolos logrando que la diferencia entre los símbolos sea menor y por consiguiente los centroides queden mejor alineados y el cálculo de la recta que pasa por ellos resulte más exacto.

El proceso de dilatación se realiza previo al cálculo del centroide porque como los símbolos están uno al lado del otro, si se dilata mucho la imagen con los símbolos juntos, estos pueden unirse entre sí.

En cambio, si se dilata cada símbolo por separado, éste puede dilatarse mucho logrando una mejoría notable en el cálculo de los centroides.

La dilatación consiste en tomar un elemento estructural cuadrado, de conectividad de 8 vecinos (los 8 vecinos que rodean al píxel examinado).

Se realiza dilatación de regiones blancas que consiste en lo siguiente:

Para cada píxel de la imagen del objeto binarizado (donde el objeto es blanco y el fondo negro), si el píxel es blanco, los 8 píxeles vecinos se cambian a blanco de lo contrario permanecen negros.

Los centroides se calculan a partir de la imagen con un símbolo solo dilatado:

$$x_o = \frac{\sum_{ij} \rho_{ij} x_j}{A}$$ columna del centroide del símbolo.

$$y_o = \frac{\sum_{ij} \rho_{ij} y_i}{A}$$ fila del centroide del símbolo.

$$A = \sum_{ij} \rho_{ij}$$ área del símbolo.

$\rho_{ij} = 1$ dentro del símbolo.

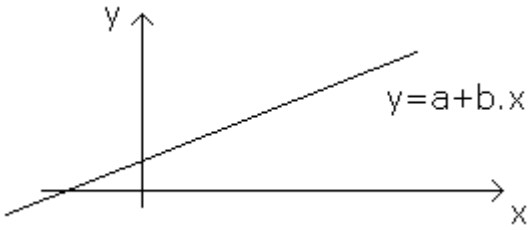
$\rho_{ij} = 0$ fuera del símbolo.

Mientras se calculan los centroides de cada símbolo, van almacenándose sus coordenadas en una tabla. Seguidamente se genera una imagen de fondo negro y un punto blanco en cada centroide calculado. Se calcula la recta que mejor ajusta dichos puntos.

Cálculo del ángulo

A partir de los centroides, se debe encontrar un método eficiente que determine con la mayor exactitud posible la ecuación de la recta que pasa por dichos centroides.

El método utilizado es el de *Mínimos Cuadrados* [12]. El mismo consiste en calcular la ecuación de la recta que mejor aproxima a los centroides (Regresión Lineal), ver apéndice C. El coeficiente angular (ángulo buscado) y el término independiente de la recta se obtiene de la siguiente manera:



$x_i = x_1, x_2, ..., x_n$ coordenada x de los centroides.

$y_i = y_1, y_2, ..., y_n$ coordenada y de los centroides.

La ecuación de la recta que mejor ajusta a los centroides es:

$$\hat{y}_i = a + b \cdot x_i \qquad i = 1,2,....,n$$

donde:

n es la cantidad máxima de centroides.

$$b = \frac{\sum_{i=1}^{n} x_i \cdot y_i - \frac{\sum_{i=1}^{n} y_i \cdot \sum_{i=1}^{n} x_i}{n}}{\sum_{i=1}^{n} x_i^{\,2} - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n}}$$ Coeficiente angular de la recta.

$$a = \frac{\sum_{i=1}^{n} y_i}{n} - b \cdot \frac{\sum_{i=1}^{n} x_i}{n}$$ Término independiente de la recta.

El ángulo θ que forma la recta con el eje horizontal expresado en grados es:

$$\theta = Arctg(b) \cdot \frac{180}{\pi}$$

La matrícula de Montevideo tiene siete símbolos, tres letras y cuatro números, por consiguiente contiene siete centroides. El método de Mínimos Cuadrados para calcular el ángulo que forma la recta con el eje horizontal utiliza sólo siete puntos, obteniéndose muy buenos resultados.

En algunos casos, el 2do y 6to símbolo de la matrícula aparecen unidos como un sólo objeto debido a que entre dichos símbolos y el borde se encuentra el tornillo que ajusta la matricula al automóvil. Es por esto que al eliminar el borde también son eliminados estos dos símbolos quedando solo cinco. Entonces, para calcular el ángulo, en vez de utilizar siete centroides, se utilizan cinco.

El peso que aportan esos dos dígitos de menos al cálculo del ángulo es poco en comparación con el peso de los cinco restantes, lo que hace que se sigan obteniendo buenos resultados.

El costo computacional es muy bajo en comparación con otros métodos (por ejemplo, transformada de Hough) debido a que sólo utiliza las coordenadas de los centroides para calcular los coeficientes de la recta.

Rotación

Una vez obtenido el *ángulo* (en grados) que forma la recta con el eje horizontal, se procede a rotar la imagen el *ángulo* calculado tomando como centro de rotación el centro de la imagen.

Para que los ejes estén en el centro de la imagen, es necesario hacer una transformación que traslade las coordenadas de la imagen original (i, j) a las coordenadas (x, y) cuyos ejes se encuentran en el

centro de la imagen original. Luego se aplica el movimiento, por último es necesaria una transformación para convertir las coordenadas (x, y) con los ejes centrados en la imagen destino en las coordenadas (u, v) de la imagen destino.

Para implementar la rotación, se debe conocer de qué tamaño quedará la nueva imagen rotada para así poder trasladar los ejes al centro de la misma.

Al aplicar el movimiento, la coordenada obtenida no es un entero, y por lo tanto se debe utilizar interpolación. En nuestro caso utilizamos interpolación bilineal, la cual consiste en calcular la intensidad del píxel destino ponderado por las intensidades de los cuatro píxeles vecinos.

No es posible aplicar esta interpolación con la transformación directa ya que no todos los vecinos del transformado tienen una intensidad asignada. Por consiguiente, se aplica la transformación inversa para asegurar que no queden huecos.

La implementación de la rotación es la siguiente:

1) Como el centro de rotación es el centro de la imagen, se calcula el centro de la imagen (x_o, y_o).

2) Se realiza un cambio de coordenadas de los ejes hacia el centro de la imagen (x_o, y_o) mediante la siguiente matriz de traslación *matrizTraslacionOrigen*:

$$\begin{bmatrix} 1 & 0 & -x_o \\ 0 & 1 & -y_o \\ 0 & 0 & 1 \end{bmatrix}$$

3) Se utiliza la siguiente matriz de rotación *matrizRotacion* de eje perpendicular al plano de la imagen.

$$\begin{bmatrix} Cos\,\theta & Sen\,\theta & 0 \\ -Sen\,\theta & Cos\,\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

donde $\theta = angulo \cdot \frac{\pi}{180}$ (en radianes).

4) Se multiplica ambas matrices *matrizRotacion* * *matrizTraslacionOrigen* y se obtiene la transformación geométrica necesaria para rotar la imagen con eje en el centro de la imagen.

5) Se calcula las dimensiones de la imagen rotada aplicándole la transformación geométrica anterior a los vértices de la imagen a rotar para saber qué tamaño va a tener y así poder sacar los ejes del centro de la imagen rotada (x_1, y_1).

6) Se aplica nuevamente una traslación, un cambio de coordenadas de los ejes desde el centro de la imagen (x_1, y_1) mediante la siguiente matriz de traslación *matrizTraslacionDestino*:

$$\begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix}$$

7) Se multiplica ambas matrices *matrizTraslacionDestino* * *(matrizRotacion* * *matrizTraslacionOrigen)* y se obtiene la transformación geométrica completa (*movimiento)* del movimiento de rotación.

8) Se calcula la inversa (*movimientoInverso)* de *movimiento* para asegurar que no queden huecos en la imagen rotada. Para cada píxel de la imagen rotada con coordenadas (u, v) se aplica la transformación *movimientoInverso* obteniendo las coordenadas (i, j) que son utilizadas para verificar si pertenecen o no a la imagen original. En caso de que (i, j) sí pertenezca, se utiliza interpolación bilineal (de orden 1) para calcular el valor de la intensidad del píxel (u, v) ponderado por los cuatro píxeles que rodean a la coordenada (i, j). En caso que no pertenezca, al píxel de coordenadas (u, v) se le asigna un píxel blanco.

La ***Interpolación Bilineal*** utilizada es la siguiente:

$x = Floor(i)$
$y = Floor(j)$ Se trunca el número flotante en un entero.

$$I(i,j) = (1-a)\cdot(1-b)\cdot I(x,y) + (1-b)\cdot a\cdot I(x+1,y) + (1-a)\cdot b\cdot I(x,y+1) + a\cdot b\cdot I(x+1,y+1)$$

donde

$$a = i - x$$

$$b = j - y$$

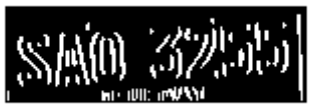
Ejemplo práctico

A continuación se detalla un ejemplo práctico mostrando las imágenes de los resultados obtenidos en cada bloque explicado previamente para calcular el ángulo y realizar la rotación de la matricula.

La matrícula segmentada original es:

Los procesos a través de los distintos bloques son los siguientes:

Detección de bordes

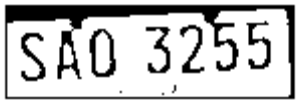


Cálculo del umbral óptimo

Umbral Óptimo = 75.7902

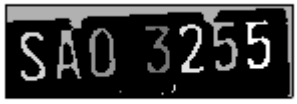
Binarización

Erosión



Como se puede observar, los símbolos ‘A’ y ‘5’ están unidos al borde luego de la erosión.

Etiquetado

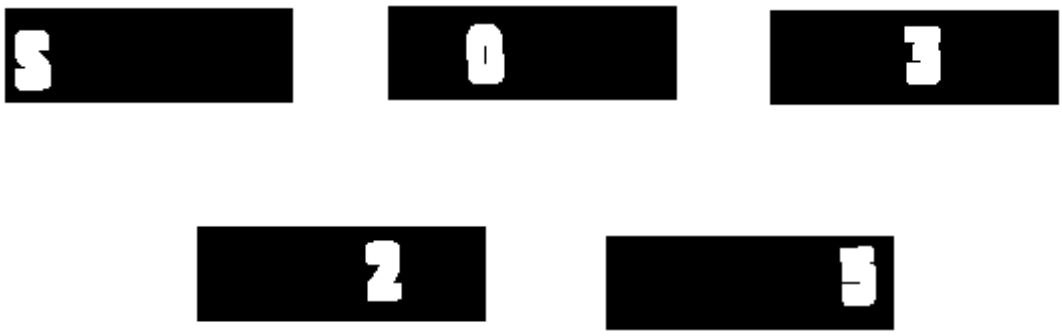


Eliminar objetos contiguos al borde

Como se observa, los símbolos unidos al borde fueron eliminados.

Cálculo de centroides

Los caracteres individuales dilatados son:



Los centroides calculados son:

Cálculo del ángulo

Los coeficientes de la recta calculados son:

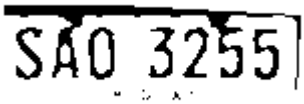
Termino Independiente (*a*) = 19.4203
Coeficiente Angular (*b*)= 0.0458

Angulo = $Arc\,\mathrm{tg}(b)\cdot\frac{180}{\pi}$ = 2.62º

Rotación

Al final se obtiene la matricula segmentada original rotada:

y la matricula binarizada, erosionada y rotada:



Como puede apreciarse, el método elegido de Mínimos Cuadrados es robusto y cumple con su objetivo utilizando en este ejemplo cinco centroides. En el caso que se trabaje con los siete centroides, se obtienen resultados más exactos, aunque es imperceptible el error cometido al usar solo cinco.

A continuación se presentan otros métodos evaluados para el cálculo del ángulo de rotación, descartados en función de su no tan buena performance.

3.2.2 Centroide o Baricentro y Transformada de Hough

Este método [16] fue probado para obtener la ecuación de la recta luego de tener la imagen de puntos indicando todos los centroides calculados.

Se quiere hallar la recta que pasa por los puntos 1, 2 y 3.
Para ello se utiliza la ecuación de la recta en coordenadas polares:

$$\rho = x \cdot Cos\theta + y \cdot Sen\theta$$

donde ρ es la distancia de la recta al origen
θ es el ángulo de ρ con el eje x.

Utilizando esta parametrización, a cada punto del plano (x, y) le corresponde una sinusoide en el plano (ρ,θ). Por consiguiente la recta que pasa por los tres puntos se visualizará en el plano (ρ,θ) como un punto que estará determinado por la intersección de las sinusoides correspondientes a cada punto.

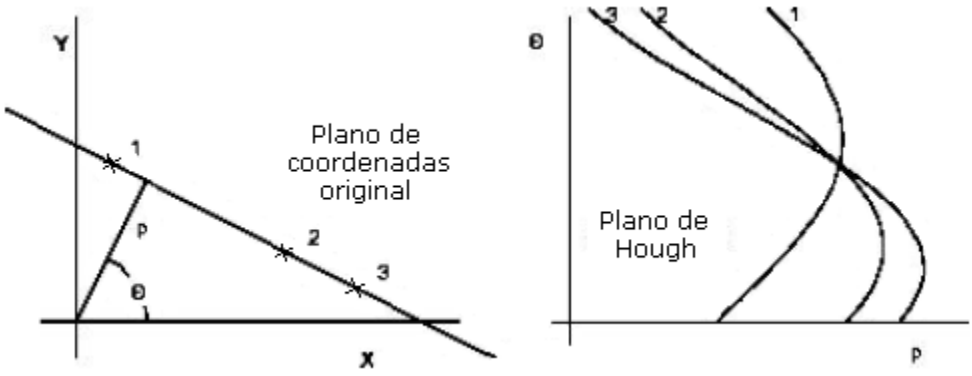


Fig. 3.6 Transformada de Hough

La implementación es la siguiente:

1) Se determina $\theta_{min}, \theta_{max}, \rho_{min}$ y ρ_{max}.

2) Se divide en celdas y se ponen en 0.

3) Se itera sobre los puntos de la imagen:

- Por cada (x_i, y_i) $\Rightarrow$ $\rho = x_i \cdot Cos\theta + y_i \cdot Sen\theta$
- Se recorre los θ_p (desde $\theta_{\min}$ hasta $\theta_{\max}$) y se determinan los ρ_p

$$\rho_p = x_i \cdot Cos\theta_p + y_i \cdot Sen\theta_p \quad \text{para cada} \quad \theta_p$$

- Se asigna plano (ρ_p, θ_p)= plano (ρ_p, θ_p)+1

4) Se halla el máximo del plano (ρ_p, θ_p) y ese punto corresponde al ángulo θ_p que forman la mayoría de las rectas, éste es el ángulo buscado.

A continuación se muestra una figura del plano (ρ_p, θ_p) resultante de calcular la recta que pasa por los centroides.

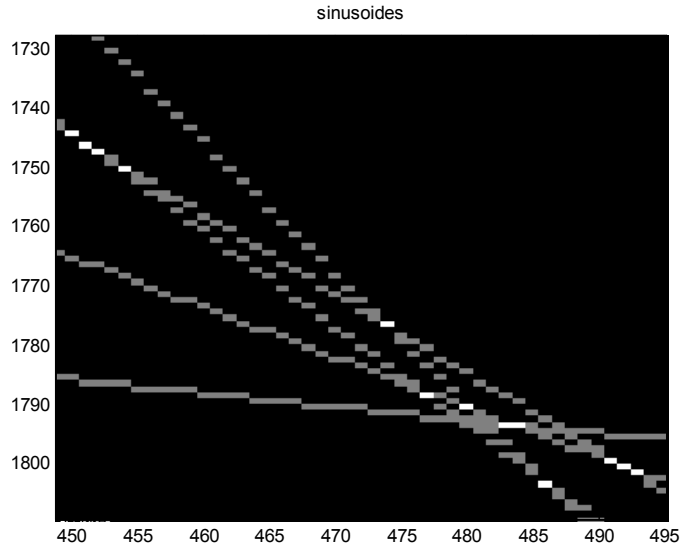


Fig. 3.7 Transformada de Hough de los centroides

Esta figura muestra un zoom en un entorno del punto donde se cortan las sinusoides. En éste caso son cinco sinusoides debido a que se eliminaron dos dígitos que estaban unidos al borde.

En éste caso, el máximo valor del plano es 2, lo cual está representado por un punto blanco. Como se puede apreciar, aparecen 13 puntos blancos que son todos de la misma intensidad, indicando que todos valen lo mismo. Es decir, hay 13 puntos que valen el máximo valor del plano.

Esto, claramente indica la no viabilidad de utilizar este método para encontrar la recta que pasa por los cinco centroides. La razón es la poca cantidad de puntos, ya que para obtener buenos resultados utilizando la transformada de Hough, es necesario tener una mayor cantidad de puntos de la recta para tener más cruces por un mismo punto en el plano (ρ_p, θ_p).

3.2.3 Eje de Inercia

Este método es adecuado para objetos con cierta simetría y consiste en lo siguiente:

1) Se calcula el tensor de inercia del objeto

$$M = \begin{pmatrix} Mxx & Myy \\ Mxy & Mxy \end{pmatrix}$$

siendo:
$$M_{xx} = \sum_{ij} \rho_{ij} x_{ij}^{\ 2} - \frac{\left(\sum_{ij} \rho_{ij} x_{ij} \right)^2}{A}$$

$$M_{yy} = \sum_{ij} \rho_{ij} y_{ij}^{\ 2} - \frac{\left(\sum_{ij} \rho_{ij} y_{ij} \right)^2}{A}$$

$$M_{xy} = \sum_{ij} \rho_{ij} x_{ij} y_{ij} - \frac{\sum_{ij} \rho_{ij} x_{ij} \sum_{ij} \rho_{ij} y_{ij}}{A}$$

$A = \sum_{ij} \rho_{ij}$ área del objeto.

$\rho_{ij} = 1$ dentro del objeto.

$\rho_{ij} = 0$ fuera del objeto.

2) Se resuelve la ecuación característica de la matriz para hallar una base de vectores propios:

$$\left|M - \lambda I\right| = 0 \Rightarrow \lambda_1, \lambda_2$$

3) Los vectores propios son:

$$vp_i = \ker\left(M - \lambda_i I\right)$$

4) Haciendo cuentas el ángulo que forma el eje de inercia es

$$\theta = Arctg\left(\frac{M_{yy} - M_{xx} \pm \sqrt{\left(M_{xx} - M_{yy}\right)^2 + 4M_{xy}{}^2}}{2M_{xy}}\right)$$

Se calcula el ángulo que forma el eje de inercia de cada símbolo, previamente dilatado, con respecto al eje horizontal. Luego se calcula el ángulo de rotación promediando los ángulos que forman todos los caracteres.

Al ser los símbolos diferentes, los ángulos que forman los ejes de inercia son todos muy diferentes también. Por lo tanto, este método no es útil para obtener el ángulo de rotación.

3.3 Segmentación de Símbolos

El bloque de segmentación de símbolos es el encargado de extraer como imágenes binarias individuales cada letra o número de la imagen binaria de la matrícula. Para esto, el método más natural sería realizar un etiquetado de los símbolos y luego obtener el mínimo rectángulo que contiene a cada símbolo. El problema con este método radica en la posibilidad de que algunos símbolos se encuentren unidos, incluso después de la erosión. Esta situación se da en matrículas en las cuales el tornillo de fijación une dos símbolos.

Por todo lo antes dicho se decidió utilizar un método más robusto, el cual se propone en [9] y consiste en acumular todas las columnas de una determinada fila y así obtener una fila acumulada (se crea el vector de las filas proyectadas). Si en una fila hay muchos píxeles blancos (mucho fondo) se obtendrán picos chicos y si por el contrario hay muchos píxeles negros (letras o números) se obtendrán picos grandes. Este procedimiento se aplica a todas las filas de la imagen de la matrícula.

De ésta forma, se puede hallar la región que contiene a los símbolos, dado que en esa región la cantidad de píxeles negros (letras o números) siempre se encuentra en una banda definida por dos umbrales obtenidos empíricamente. Para poder hacer esto, previamente se normaliza el ancho (nº de columnas) de las matrículas segmentadas manteniendo la relación de aspecto. Ésto se logra por medio de una homotecia utilizando interpolación bilineal, la cual se aplica a las matrículas segmentadas en niveles de gris.

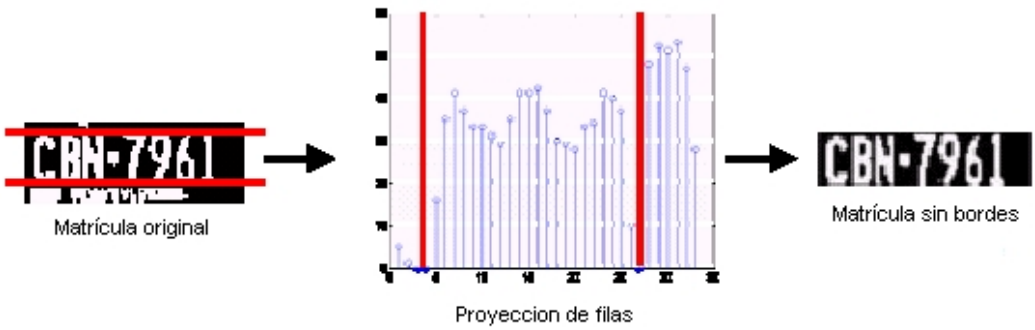


Fig. 3.8 Proyección de filas (Segmentación de símbolos)

El algoritmo desarrollado trabaja de la siguiente manera:

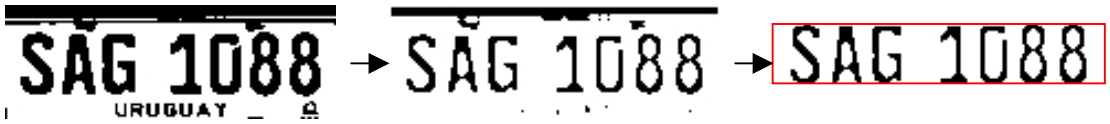
1) En el vector de las filas proyectadas se buscan los posibles primer pico de la región buscada, los cuales deben estar entre dos umbrales, *picoSup* y *picoInf*. Estos se guardan en un vector llamado *posiblesPicos*.

2) Se comienza con el primer elemento del vector *posiblesPicos*, y se verifica que la siguiente fila proyectada esté entre dos umbrales, *cotaSup* y *cotaInf,* así sucesivamente con las siguientes filas, siempre y cuando la anterior se encontrara entre dichos umbrales. Adicionalmente se van contando la cantidad de filas consecutivas que están entre los umbrales antes mencionados.

3) Si las filas contadas son mayor que *anchoMat,* entonces ya se encontró la región buscada y por tanto se termina el algoritmo, pero si no, se retrocede a 2) con el siguiente elemento del vector *posiblespicos*. Así se continúa hasta que se encuentre un candidato o se acaben los posibles primer pico.

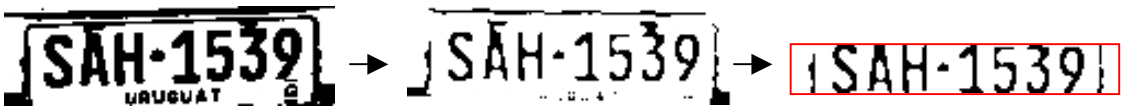
4) Si se acaban los posibles primer pico entonces se vuelve a 2) pero antes se disminuye en uno la *cotaInf* ampliando de esta forma el rango. Esto se repite hasta un valor mínimo de *cotaInf*, si se llega a ese valor y no se encontró un candidato, entonces se termina el algoritmo y se despliega un mensaje de que no se pudo eliminar los bordes.

Ejemplo 1:



Matrícula binarizada Erosionada y rotada Matrícula sin bordes

Ejemplo 2:



Matrícula binarizada Erosionada y rotada Matrícula sin bordes

Con el siguiente diagrama se trata de clarificar la idea del algoritmo desarrollado para eliminar los bordes:

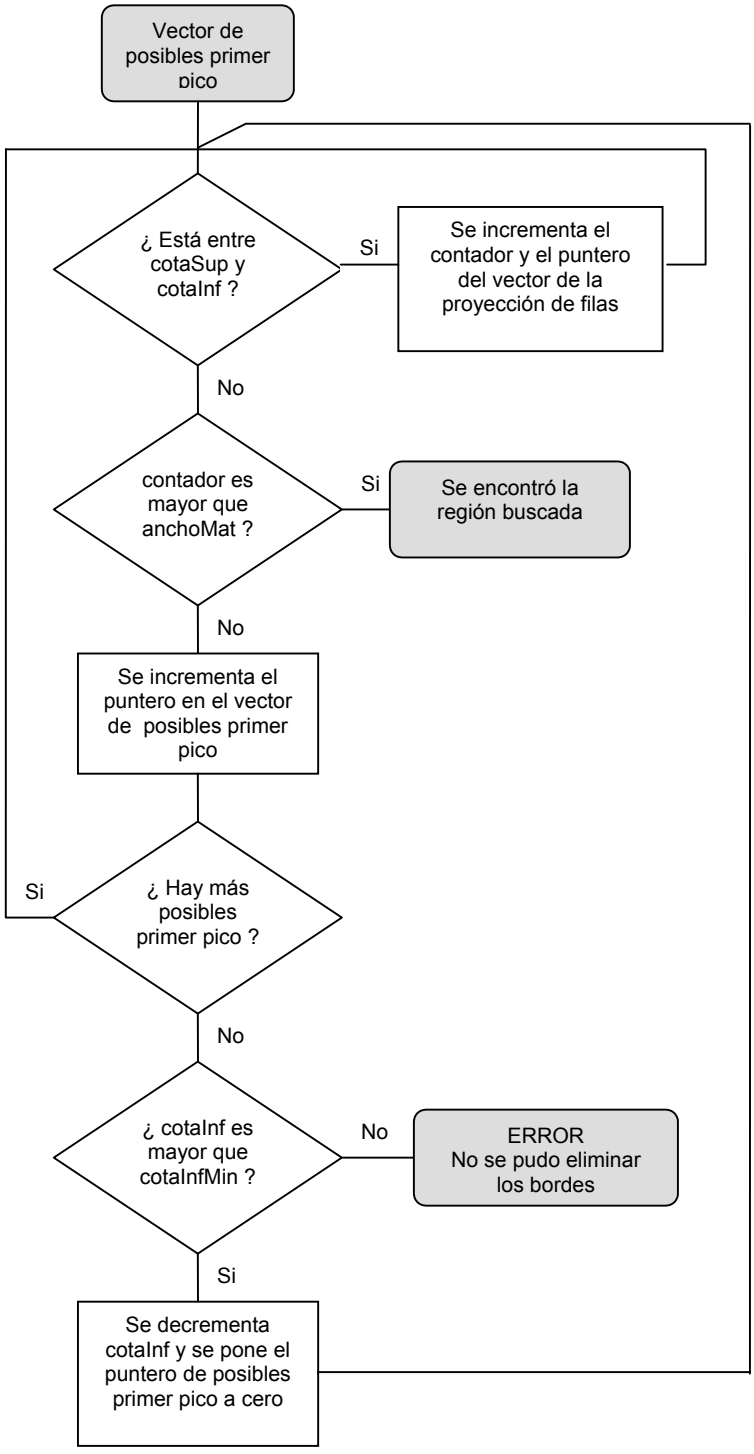


Fig. 3.9 Diagrama de flujo del algoritmo para eliminar los bordes

Luego de obtener la matrícula sin los bordes superior e inferior se deben segmentar los símbolos. Para ello se procede a proyectar las filas de una determinada columna y así obtener una columna acumulada (se crea el vector de las columnas proyectadas). Si hay muchos píxeles blancos (mucho fondo) se obtendrán picos chicos y si por el contrario hay muchos píxeles negros (letras o números) se obtendrán picos grandes. Este procedimiento debe ser hecho con todas las columnas de la imagen de la matrícula. De esta forma, se puede encontrar la separación de símbolos en la dirección horizontal buscando los picos en el vector de las columnas proyectadas.

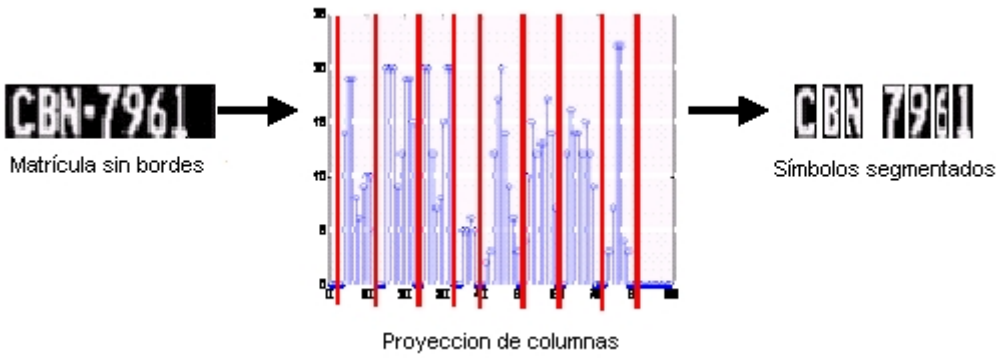


Fig. 3.10 Proyección de columnas (Segmentación de símbolos)

El algoritmo desarrollado trabaja de la siguiente manera:

1) Se proyectan las filas de una columna, esto para todas las columnas de la matrícula (se crea el vector de las columnas proyectadas).

2) Se recorre todo el vector de las columnas proyectadas, si proyeccionCol[j-1]<=*umbral* y proyeccionCol[j]>*umbral* entonces se encontró un borde izquierdo y por tanto se guarda en la tabla de bordes. En la cual los símbolos se guardan por fila y el borde izquierdo en la primer columna mientras que el borde derecho se guarda en la segunda columna.

3) Si proyeccionCol[j-1]>*umbral* y proyeccionCol[j]<=*umbral* y para este supuesto símbolo ya se guardo un borde izquierdo entonces se encontró un borde derecho y por tanto se guarda en la tabla de bordes. Se vuelve a 2) hasta recorrer todo el vector de las columnas proyectadas.

4) A los símbolos de los extremos se les verifica que la proyección sea más ancha que un espesor mínimo y además que sea en su punto medio menos alta que un 90% del alto de la matrícula sin bordes (esto para eliminar los bordes verticales macizos). Si no cumplen con alguna de estas condiciones se los descarta.

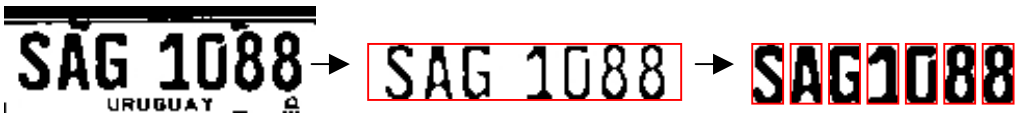
5) Para los símbolos que no son extremos se verifica que la proyección sea más ancha que un espesor mínimo y además que su área sea mayor que un cierto valor (esto para eliminar al punto que separa las letras de los números). Si no cumplen con alguna de estas condiciones se los descarta.

6) Si el número de símbolos segmentados es menor que *numeroDeSimbolos* y todavía no se iteró entonces se dilata la imagen de la matrícula sin bordes y se vuelve a 1), solo se puede iterar una vez, (por lo tanto a la imagen de la matrícula solo se la va a dilatar una única vez), si nuevamente el número de símbolos es menor que *numeroDeSimbolos* entonces se da un mensaje de error.

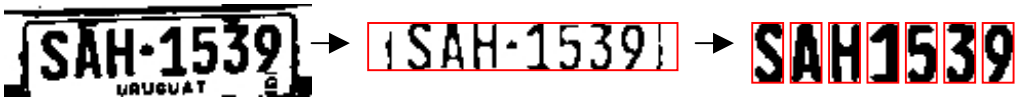
7) Si se obtuvieron bien las posiciones de los símbolos, o sea que la tabla de bordes tiene una cantidad de filas igual a *numeroDeSimbolos* se procede a segmentarlos de la imagen sin bordes y sin dilatar.

8) A cada símbolo por separado se lo dilata, esto se hace porque si recordamos a la matrícula binarizada se la erosionó para poder corregir la inclinación y facilitar la segmentación de símbolos. Además se le aplica una homotecia para llevarlo a 30x15, siendo este el tamaño utilizado para normalizar todos los símbolos, vale aclarar que los templates de los símbolos también tienen este tamaño. Quedando todo pronto para hacer la correlación en el algoritmo de OCR.

Ejemplo 1:



Ejemplo 2:



Con el siguiente diagrama se trata de clarificar la idea del algoritmo desarrollado para segmentar los símbolos desde el punto 1) al 3):

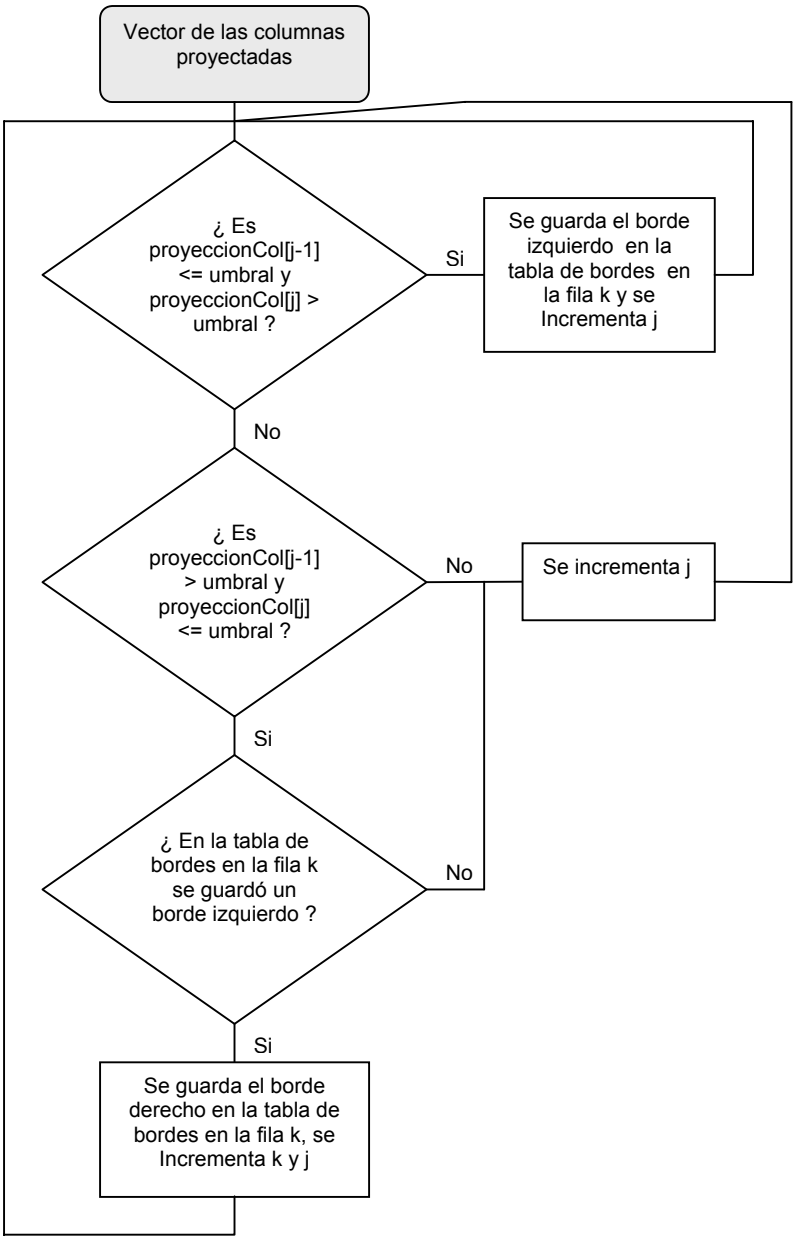


Fig. 3.11 Diagrama de flujo del algoritmo para segmentar los símbolos

3.4 Reconocimiento de Caracteres (OCR)

¿Qué es un OCR?

La palabra OCR significa Optical Character Recognition y es un método para convertir caracteres visualmente legibles en caracteres que pueda entender un computador. Si bien los seres humanos son capaces de leer caracteres, las computadoras no lo pueden hacer. Ellas utilizan códigos binarios que en su conjunto forman lo que se conoce como código ASCII.

Existen diferentes métodos con mayor o menor eficacia para realizar OCRs que se analizan en este capítulo.

En esta parte del Proyecto fueron estudiados distintos software gratuitos disponibles en Internet así como también algoritmos o métodos desarrollados en trabajos similares.

Algunos de los software disponibles en la web que realizan OCR son:

i. Clara OCR (http://www.claraocr.org)
ii. gOCR (http://jocr.sourceforge.net)
iii. LOCR (http://www.math.northwestern.edu/~mlerma/locr)
iv. OCRChie (http://http.cs.berkeley.edu/~fateman/kathey/ocrchie.html)

Los cuatro tienen código abierto y disponible para descargar desde su página web.

De todos ellos, ClaraOCR es sin dudas el más completo. Cuenta con una interfaz gráfica de usuario muy "amigable" (trabaja en el entorno XWindow de Linux), además de una interfaz por línea de comandos. Está pensado para ser usado en reconocimiento de libros. Utiliza un sistema de "entrenamiento" de manera de posibilitar un mejor reconocimiento para cada libro. Su desarrollo comenzó en 1999 y existen manuales y FAQs completos en su sitio web. Para su sistema de reconocimiento utiliza el matcheo con el esqueleto de las letras. Dos símbolos son considerados iguales cuando cada uno contiene el esqueleto del otro.

En las pruebas realizadas, arrojó resultados correctos con las letras de la matricula que no estaban debajo del tornillo. En estos casos, falló la segmentación uniendo la letra con el fondo y por lo tanto fallando en el reconocimiento.

gOCR se encuentra aún en desarrollo y presenta menos facilidades y no tan buenos resultados.

LOCR es un sistema de reconocimiento programado por Miguel A. Lerma, Director de Computación en el Depto. De Matemática de la Northwestern University de Illinois, EEUU. Este software está aún en etapa de desarrollo, y por el momento su performance no es buena. No cuenta con interfaz gráfica.

El OCRChie es un proyecto de fin de carrera en la Universidad de California en Berkeley (1996). Está pensado también para reconocimiento de texto en libros. Posee un sistema de entrenamiento más complejo de utilizar que el de ClaraOCR. Su funcionamiento se basa en dividir en grupos los caracteres aprendidos según sus características (tamaño, letras que tengan partes hacia abajo como por ejemplo p, j, y, caracteres especiales, etc.) y comparando las entradas según el grupo al cual correspondan.

En cuanto a los métodos o algoritmos encontrados en trabajos relacionados con reconocimientos de matrículas, se destacan: método de la proyección, clasificación mediante la utilización de redes neuronales, método de la grilla, número de Euler, y correlación.

3.4.1 Método de la proyección

En este método se obtienen las proyecciones verticales y horizontales del caracter a reconocer y se comparan con un alfabeto de caracteres posibles. La distancia mínima determina el caracter "ganador". Por ejemplo, se podría observar cuantos picos tienen en cada proyección determinado patrón o template y comparar con el dígito a reconocer. Se supone que previamente le fue aplicada una transformación de homotecia al caracter de manera de comparar siempre elementos del mismo tamaño. La homotecia podría ser hecha utilizando interpolación bilineal sobre las imágenes en niveles de gris.

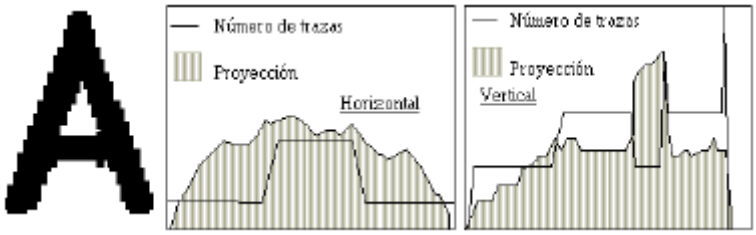


Fig. 3.12 Método de la proyección

3.4.2 Clasificación utilizando redes neuronales

Las redes neuronales están siendo cada vez mas usadas para reconocimiento automático de caracteres. Estas tratan de simular el comportamiento de aprendizaje de las neuronas humanas. Permiten por ejemplo interpolar resultados a partir de un conjunto de entradas no conocido para la red.

La red consiste de tres capas de neuronas: de entrada, oculta y de salida. Teniendo cada una de estas capas un número determinado de neuronas con un determinado peso cada una.

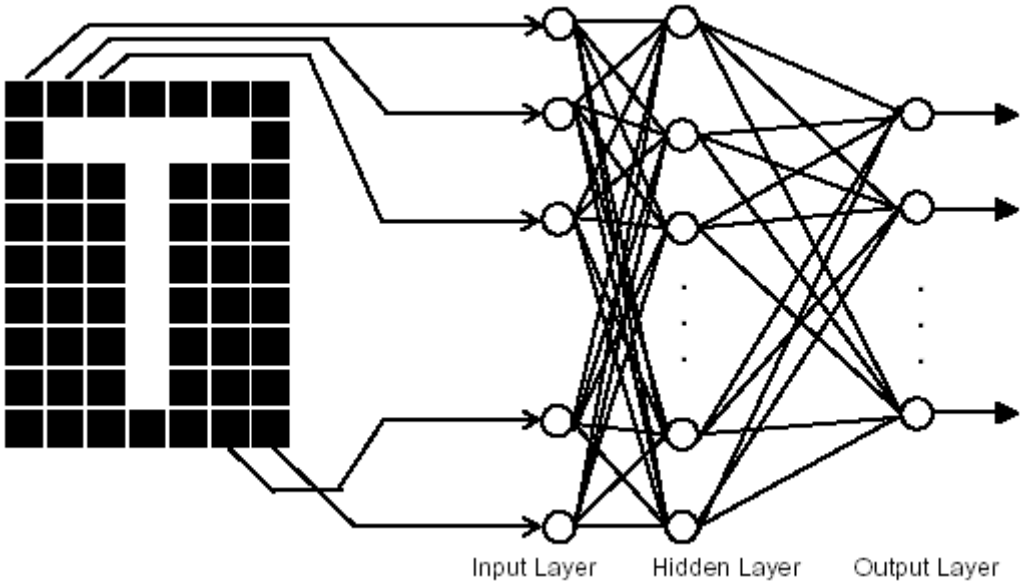


Fig. 3.13 Clasificación utilizando redes neuronales

Capa de entrada (*Input Layer*): Contiene en el ejemplo 450 neuronas que representan la entrada binaria de la imagen del caracter (0 o 1) de tamaño 15x30 píxeles.

Capa Oculta (*Hidden Layer*): Contiene 900 neuronas calculadas arbitrariamente como el doble de las neuronas de la capa de entrada. El que esta capa contenga muchas neuronas, hace que sea mas exacta la salida pero por el contrario perjudica la capacidad de generalización de la red para nuevas entradas. La generalización de las nuevas entradas es la capacidad de la red para interpolar y producir resultados correctos en caso que la entrada no haya pertenecido a ningún grupo de entrenamiento.

Capa de Salida (*Output Layer*): Contiene 36 neuronas donde cada caracter (10 números y 26 letras) está representado unívocamente por una neurona. Idealmente, la salida debería ser un '1' en la correspondiente neurona de salida y '0' para todas las demás con

respecto al caracter de entrada. Esto no es totalmente cierto y las neuronas no darán salida '0' sino que hay que establecer un umbral de decisión.

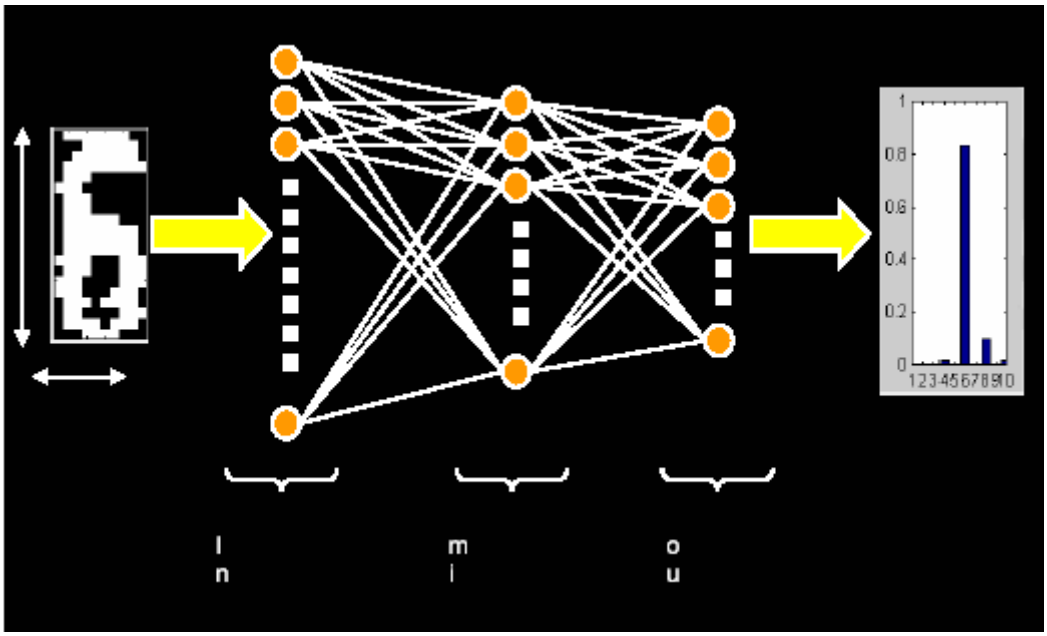


Fig. 3.14 Clasificación

Queda claro entonces que las redes neuronales son una función de vector a vector.

El algoritmo de *back-propagation* trabaja con lo que se llama entrenamiento supervisado. Primeramente se le ingresan de a una a la red entradas para que la recorran hacia adelante. La salida es comparada con la salida deseada idealmente suministrada por un supervisor. A continuación se calculan los errores de todas las neuronas en las etapas de salida. La idea básica detrás de la *back-propagation* es que ese error es propagado hacia atrás hasta las primeras capas, de manera que un algoritmo modifique adecuadamente los pesos asignados a cada neurona de cada capa para minimizar el error.

El entrenamiento se puede detener cuando el error de salida cae por debajo de determinado umbral.

Debajo se ilustra lo que podría ser un grupo de imágenes de entrenamiento de una red para el número 4:

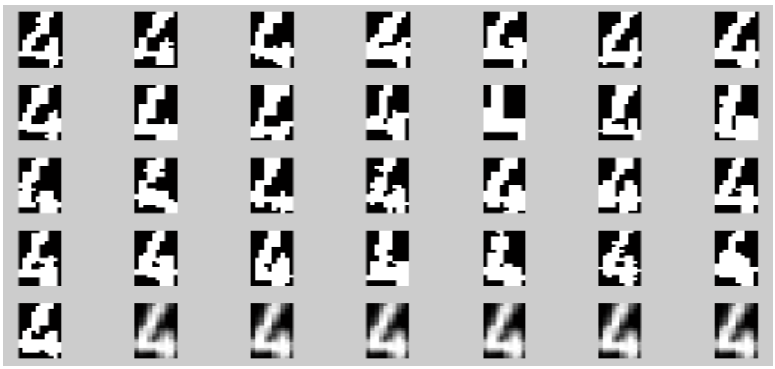


Fig. 3.15 Grupo de entrenamiento

3.4.3 Método de la grilla

Este método consiste en determinar en matrices patrones correspondientes a los dígitos la cantidad de veces que intersectan una determinada grilla.

De esta manera, si se tiene un símbolo segmentado de la matrícula en el proceso del reconocimiento, comparando las veces que es intersectado por la misma grilla contra los cortes "patrones" se podría deducir o tener una idea de los candidatos. Este método tiene la desventaja que puede darse el caso que dos dígitos distintos sean cortados igual cantidad de veces por la grilla.

De todos modos, podría ser tenido en cuenta como un método de respaldo o apoyo para algún otro método más robusto.

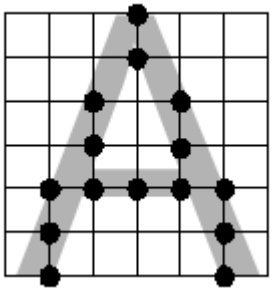


Fig. 3.16 Método de la grilla

3.4.4 Número de Euler

El número de Euler es una propiedad estructural de una imagen. Está definido como el número total de objetos en una imagen menos el número de agujeros en ellos. Por ejemplo si se tuviera la imagen de un 8, se tendría un objeto y dos huecos. El número de Euler sería 1-2=-1. Mediante el cálculo de este número es posible por ejemplo distinguir entre tres grupos de dígitos:

Grupo 1 (sin huecos): 1, 2, 3, 5, 7
Grupo 2 (un hueco): 6, 9, 0, 4
Grupo 3 (dos huecos): 8

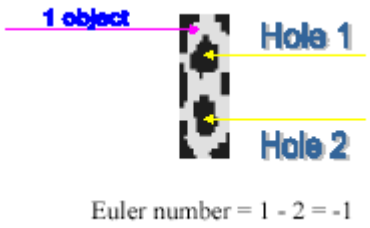


Fig. 3.17 Número de Euler

Este método combinado con el método anterior o con la correlación podría mejorar los resultados.

3.4.5 Correlación

Se decidió utilizar este método porque es una buena medida para considerar similitud entre matrices. No es sensible a pequeñas variaciones y presenta una buena relación entre performance y facilidad de implementación.

Se calcula la correlación de un símbolo patrón (template) y la matriz de los símbolos segmentados de la matrícula. La entrada al método de correlación son los símbolos binarios segmentados.
Se obtuvieron previamente patrones de matrículas los cuales fueron optimizados utilizando editores de imágenes.

Vale la pena aclarar que la comparación es hecha entre matrices de igual tamaño, estando éste estandarizado a 30x15 (tamaño que se eligió para los templates), a los símbolos segmentados de la matrícula se les aplica una homotecia para llevarlos a 30x15.

$$r = \frac{\sum_m \sum_n (A_{mn} - \overline{A}) \cdot (B_{mn} - \overline{B})}{\sqrt{(\sum_m \sum_n (A_{mn} - \overline{A})^2) \cdot (\sum_m \sum_n (B_{mn} - \overline{B})^2)}}$$

El estimador ***r***, da una aproximación a cuán parecido es el patrón o template al símbolo segmentado. El parámetro ***r***, varía entre 0 y 1. Cuanto más cercano a 1 sea el resultado, más parecidas son las imágenes.

En el algoritmo, las imágenes de entrada (símbolo segmentado e imagen patrón) son normalizadas de manera que el nivel de blanco sea 0 y el de negro 1. Luego, se calcula la media del nivel negro en estas matrices sumando la cantidad de píxeles negros y dividiendo entre la cantidad total de píxeles (se obtiene $\overline{A}$ y $\overline{B}$). Con este dato, es posible ahora calcular los valores de la correlación ***r*** para cada símbolo de entrada. Para esto, existe el método OCR que está dividido en dos, una parte para los números y otra para las letras en la matrícula. Aquí otra vez se hace uso de la información a priori con la que contamos, los tres primeros dígitos son letras y los cuatro últimos números. De esta forma, dividiendo en dos el OCR, se optimiza el rendimiento del programa, ya que por ejemplo, se sabe que no es necesario correlacionar un número segmentado con las 26 letras patrones del alfabeto.

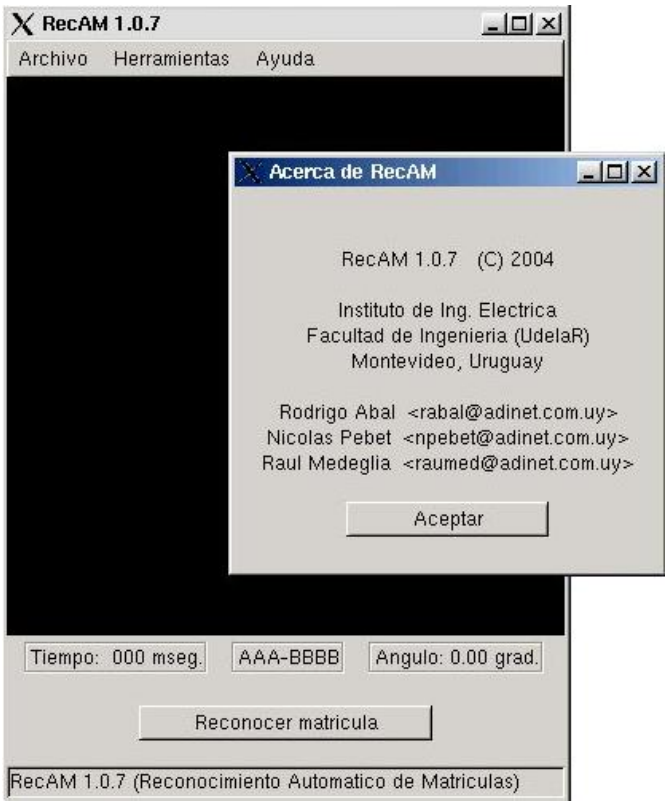
Lo que hace el OCR es aplicar la función correlación a cada símbolo y llenar un vector con los resultados.

A continuación, se busca a qué índice corresponde el mayor valor (más cercano a 1) y si supera cierto umbral (en este proyecto se eligió 0.4), ese será el candidato. El índice del vector mencionado se eligió de tal manera que el 0 corresponda a la A, el 1 a la B, el 2 a la C, y así sucesivamente en el caso de las letras. En el caso de los números hay una correspondencia directa.

Se usó un sólo template representativo de cada símbolo. Se podrían haber utilizado más de uno, y por ejemplo promediar los resultados. Aunque esto hubiera enlentecido la ejecución del código.

Capítulo 4

4. Ejemplo del funcionamiento de RecAM

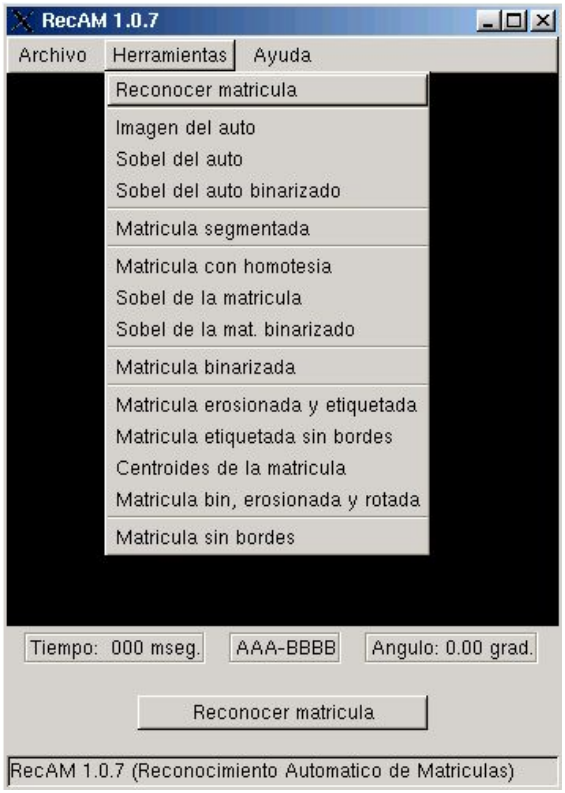


Los integrantes del proyecto RecAM se pueden visualizar en:

Ayuda ⇒ *Acerca de RecAM…*

Un ejemplo de como se visualiza el menú correspondiente a *Herramientas* donde aparecen los distintos pasos del reconocimiento de la matrícula.

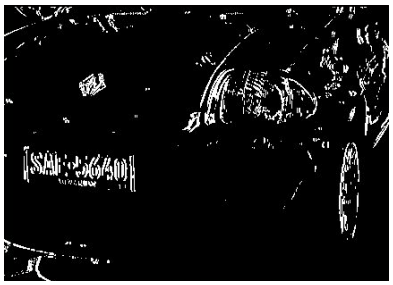
Haciendo click con botón izquierdo en cada ítem, se puede ver el resultado correspondiente.



Esta es la imagen del auto a tener como referencia para este ejemplo, ilustrando a continuación como se ven los distintos pasos intermedios hasta que se reconoce la matrícula.

⇓ Sobel del auto

⇓ Sobel del auto binarizado (umbral 5%)



⇓ Matrícula segmentada

⇓ Matrícula con homotecia

⇓ Sobel de la matrícula

⇓ Sobel de la mat. binarizado (umbral 15%)

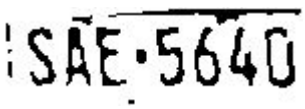
⇓ Matrícula binarizada

⇓ Matrícula erosionada y etiquetada

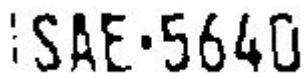
⇓ Matrícula etiquetada sin bordes

⇓ Centroides de la matrícula

⇓ Matrícula bin., erosionada y rotada



⇓ Matrícula sin bordes



Se observa que la imagen presenta una transformación de perspectiva, no solamente se encuentra torcida. Si se estimara la deformación a priori (de perspectiva), seguramente se obtendrán mejores resultados.

A partir de esta imagen de la matrícula sin bordes, se segmenta cada símbolo y luego dicho símbolo es dilatado.

Como se puede observar en los símbolos segmentados, debido a la transformación de perspectiva estos presentan una leve inclinación la cual podría ser corregida utilizando los ejes de inercia.

Al comparar con los templates, la correlación se maximiza para los siguientes símbolos.

Por tanto se obtiene como resultado:

Tiempo: 514 mseg.

String: SAE-5640

Angulo: -4.4 grados

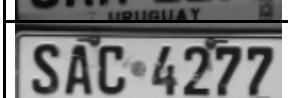
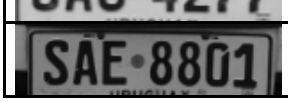
Capítulo 5

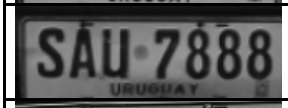
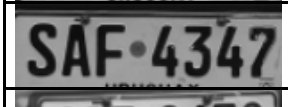
5. Resultados e Interpretación

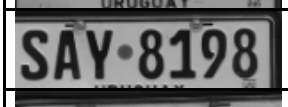
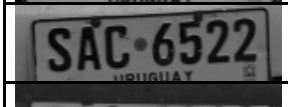
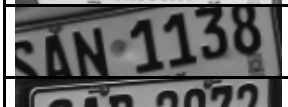
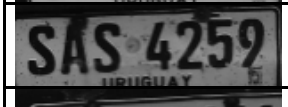
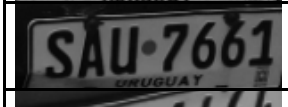
En este capítulo se muestran los resultados obtenidos de los ensayos realizados al software RecAM. En la siguiente tabla se muestran algunos de los parámetros que se consideran relevantes a la hora de evaluar su desempeño.
Cabe aclarar que el parámetro de tiempo es relativo al sistema en el cual se corre el software, en este caso los ensayos se realizaron sobre una máquina con las siguientes características:

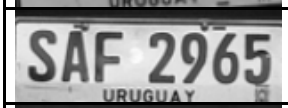
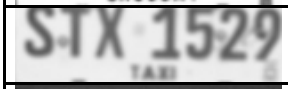
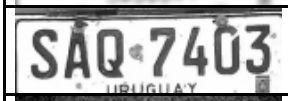
- Intel Celeron 500 Mhz
- 64 Mbytes de RAM
- Disco rígido de 6 Gbytes
- Linux RedHat 7.3
- RAVL 0.7

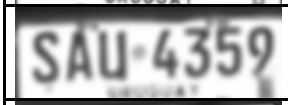
Las matrículas reconocidas incorrectamente se resaltan en color gris.

Matrícula Segmentada	Matrícula Reconocida	Tiempo de Reconocimiento (milisegundos)	Angulo Corregido (grados)
	SAU-8928	633	0.88
	SAE-8595	663	0
	SAA-3332	566	1.56
	SAA-2299	667	3.13
	BAC-4277	518	0.69
	JSA-?000	625	-0.8

	SAC-7667	660	-1.1
	SAU-7888	563	-0.6
	SAE-8703	635	2.43
	SAV-5598	648	0.72
	SAF-4347	631	-0.2
	SAE-8678	523	1.64
	SAG-1088	661	0.17
	SAA-5604	612	-0.9
	SAW-1?56	660	2.27
	SDA-6190	649	-1.2
	SAF-7064	654	-0.1
	SAH-1539	659	2.25
	SAJ-6557	667	2.75
	SAF-1832	687	-0.9
	SAK-7991	639	1
	?AS-4077	621	3.51
	SAC-1404	691	-0.1
	SME-1810	651	-0.9
	SAR-7685	673	0.17
	?SB-4975	677	1.29
	SAD-0302	639	0

	SAU-2876	659	1.4
	SAI-5895	654	1.53
	SAY-8198	630	0.37
	SAY-5507	648	2.4
	SAC-8709	665	2.33
	SAR-7613	651	-0.3
	SAJ-9574	659	0.77
	SAA-4302	640	-0.6
	SAV-5605	662	9.94
	SAI-3724	691	-7.3
	SAH-4729	681	3.74
	SAC-6522	628	2.86
	SAO-1948	618	0
	?AN-1138	611	8.97
	SAQ-2972	657	5.96
	SAJ-5074	625	4.72
	SAS-4259	673	2.31
	SAU-7661	648	2.57
	??N-1464	794	11.1
	SAD-7757	667	5.53
	SAJ-5789	656	3.1

	SAT-2271	674	4.79
	SA?-0293	678	5.44
	SAI-1652	638	-3.6
	SAA-5972	661	3.86
	SAG-1328	615	4.2
	SAF-2965	618	-0.6
	SAH-6533	581	0.37
	STX-1529	533	0
	SAO-4433	731	-0.1
	SAJ-6233	664	0.26
	SAQ-7403	666	1.82
	SDI-1891	607	0
	SAJ-6121	700	0
	SAK-1601	653	0.89
	SAN-2797	563	-0.3
	SAP-2090	576	0.46
	SAR-9601	672	0.58
	SAC-9931	673	-0.3
	SAY-5378	600	1.28
	SAN-5352	604	1.85

	SAV-2837	586	0.41
	SAH-8629	725	-0.3
	SAU-7871	589	0.65
	SAP-5199	843	0.34
	SAG-6662	643	-0.1
	SAI-3160	537	-1.8
	SAU-4359	648	1.77
	SAO-3255	647	2.62
	SAI-7084	578	-0.3
	SAW-3512	610	1.18
	SAE-5640	517	-4.4
	SAB-7552	705	0.43

5.1 Análisis de los resultados

En un total de 80 imágenes procesadas con RecAM, 58 fueron reconocidas correctamente, por tanto se podría decir que RecAM tiene una tasa de efectividad de un (58/80)*100=**72.5%**. De las 22 imágenes que fueron mal reconocidas, 14 de ellas fueron reconocidas con un sólo símbolo mal.

Vale la pena mencionar que se utilizaron imágenes en situaciones extremas, por ej. matrículas muy inclinadas, fotos tomadas desde un lateral del vehículo y no de frente, introduciendo esto último una distorsión por la perspectiva, etc.

Si se analizan los resultados, la mayoría de los errores son debidos al OCR. En la implementación actual de RecAM, el OCR utiliza la correlación como único método de reconocimiento, siendo esto un punto a mejorar en futuras versiones. En un futuro, se podrían

combinar dos métodos para alcanzar una mayor robustez en el reconocimiento. También existe la posibilidad de desarrollar un OCR basado en redes neuronales, lo cual incrementaría notoriamente la robustez del sistema.

En las imágenes mal reconocidas, se observa que el OCR muchas veces confunde al 8 con el 0 y al 2 con el 7, siendo éstos los puntos más débiles del OCR implementado en lo que refiere a números. Ésta debilidad puede ser corregida en una futura versión de RecAM, incorporando al OCR el método de Euler. De esta manera por ejemplo sería posible discriminar entre un 8 o un 0.

A continuación se muestra en color gris el número reconocido por RecAM y a su derecha el número de matrícula verdadero.

BAC-4277 SAC-4277

SAA-5604 SAA-5684

SAW-1?56 SAW-1156

SDA-6190 SBA-6190

SAF-1832 SAP-1832

?AS-4077 SAS-4027

SAC-1404 SAC-1484

SAC-8709 SAC-8209

SAA-4302 SAA-4382

SAQ-2972 SAD-2972

En estos resultados se observa claramente la confusión del 8 con el 0 y del 7 con el 2.

Otro tipo de error se presenta cuando se segmentan los símbolos. En particular el problema aparece cuando los bordes de las matrículas son considerados como símbolos. RecAM tiene implementado un algoritmo que tiene en cuenta esta situación, pero cuando por ejemplo hay una calcomanía pegada en el borde izquierdo, falla. Toma a la calcomanía y al borde como un símbolo y desplaza los símbolos verdaderos un lugar hacia la derecha. Por tanto la letra de más a la derecha será reconocida como un número y se pierde el número de más a la derecha. A continuación se muestran algunos ejemplos que ejemplifican este error:

JSA-?000 SAE-8801
En este ejemplo se dan los dos errores, el 8 es confundido con el 0, la E no es reconocida (ya que se la intenta reconocer como número) y la J es el borde que fue considerado un símbolo, además se perdió el 1.

?SB-4975 SBA-9759
En este sólo se presenta el error del borde, a la A, el OCR la reconoce como el numero 4 y el número 9 se pierde.

Otro tipo de error que encontramos se da cuando a un símbolo se lo parte en 2 símbolos y por tanto los símbolos a la derecha de este se desplazan un lugar y el de más a la derecha se pierde.

SAI-1652 SAU-6524
En este ejemplo se observa que la U se partió en dos símbolos y por tanto una mitad se reconoció como una I la otra como el número 1. El número 4 se perdió.

Un punto muy importante es que la aplicación debe ser en tiempo real. El tiempo de reconocimiento nunca superó el segundo. El promedio fue de 650 milisegundos, en un sistema estándar (500Mhz) y con relativamente poca memoria RAM (64 Mbytes).

Debe considerarse también, que este tiempo podría reducirse más aún. Al ser esta versión de RecAM académica, muchas de las imágenes obtenidas en los pasos intermedios del proceso son salvadas al disco rígido una importante cantidad de pasos intermedios para luego facilitar el análisis de los resultados. Si esto es eliminado, los tiempos de procesamiento se reducirían considerablemente. Las pruebas realizadas en un procesador P4 (2.8GHz) arrojaron resultados de 150 a 200 milisegundos.

Luego de este análisis podemos concluir que una efectividad en el reconocimiento de un **72.5%** no está nada mal, más considerando que las imágenes no son todas de frente y que además se está utilizando un solo método de OCR.

5.2 Análisis de desempeño

Para testear el desempeño de RecAM frente a imágenes distorsionadas se utilizaron dos métodos de ensayo: uno con imágenes a las cuales se les agregó ruido gaussiano, y otro con imágenes a las cuales se les aplicó una dispersión gaussiana y de esta forma se simularon imágenes fuera de foco.

A continuación se muestran algunos ejemplos:

Fig. 6.1 Imagen 1 original
Nº Reconocido:
SAF-2965

Tiempo:
620 mseg.

Angulo corregido:
-0.6º

Fig. 6.2 Imagen 1 fuera de foco
Nº Reconocido:
SAF-2965

Tiempo:
628 mseg.

Angulo corregido:
-0.9º

Fig. 6.3 Imagen 1 con ruido
Nº Reconocido:
SAF-2965

Tiempo:
621 mseg.

Angulo corregido:
-0.7º

Fig. 6.4 Imagen 2 original
Nº Reconocido:
SAO-4433

Tiempo:
730 mseg.

Angulo corregido:
-0.1º

Fig. 6.5 Imagen 2 fuera de foco
Nº Reconocido:
SAO-4433

Tiempo:
724 mseg.

Angulo corregido:
-0.7º

Fig. 6.6 Imagen 2 con ruido
Nº Reconocido:
SAO-4433

Tiempo:
728 mseg.

Angulo corregido:
-1.7º

Se muestra a continuación una pequeña tabla con los resultados obtenidos de varios ensayos:

Matrícula	Imagen	Matrícula Reconocida	Tiempo (mseg.)	Angulo (grados)
SAF-2965	Original	SAF-2965	620	-0.6
	Ruido	SAF-2965	628	-0.9
	Fuera de foco	SAF-2965	621	-0.7
SAH-6533	Original	SAH-6533	630	0.37
	Ruido	SAH-6533	568	0.24
	Fuera de foco	SAH-6533	595	-0.3
SAO-4433	Original	SAO-4433	730	-0.1
	Ruido	SAO-4433	724	-0.7
	Fuera de foco	SAO-4433	728	-1.7
SDI-1891	Original	SDI-1891	561	0
	Ruido	SDI-1891	572	0.52
	Fuera de foco	SDI-1891	558	0
SAJ-6121	Original	SAJ-6121	700	0
	Ruido	SAJ-6121	686	-0.1
	Fuera de foco	SAJ-6121	693	0.02
SAN-5352	Original	SAN-5352	610	1.85
	Ruido	¿?N-5352	537	3.21
	Fuera de foco	SAN-5352	573	1.59
SAD-7871	Original	SAU-7871	644	0.65
	Ruido	SAU-7871	577	1.17
	Fuera de foco	SAU-7871	589	0.75
SAI-7084	Original	SAI-7084	717	-0.8
	Ruido	SAI-7004	719	-0.7
	Fuera de foco	SAI-7084	726	-0.8

Si se analizan los resultados obtenidos, si bien no es una muestra muy representativa, ya que son sólo ocho matrículas, podemos ver que RecAM es bastante robusto a estos tipos de distorsión.

Capítulo 6

6. Dificultades

El presente Proyecto se basa en la biblioteca RAVL (Recognition And Vision Library) que contiene una base de clases en C++ junto con herramientas de apoyo de tratamiento de imágenes.

RAVL soporta varios sistemas operativos y compiladores, entre ellos se encuentran Windows en el cual trabajamos con el compilador Microsoft Visual C++ (6.0 o mayor) y Linux con el GNU gcc.

Al comienzo, se decidió trabajar con Microsoft Visual C++ en un ambiente Windows.

Como se presenta en el Apéndice E, RAVL presenta una serie de incompatibilidades con MVC++. Estos problemas son solucionados en alguna medida mediante la instalación de patches y Service Packs, siendo imposible solucionar la incompatibilidad con el uso de la GUI que implementa la RAVL.

Finalmente, se comenzó a utilizar Linux (Red Hat 7.3), comenzando la etapa de aprendizaje tanto en la instalación como uso del S.O.

La RAVL es completamente compatible con esta versión de Linux. Si bien también fue testeada en Red Hat 8, se decidió utilizar la versión 7.3 dado que es más flexible en cuanto a espacio requerido en el disco rígido (2GB aprox.).

Otra dificultad encontrada en el desarrollo del software fue la familiarización con la estructura de clases de la RAVL, especialmente dada la falta de documentación sobre su funcionamiento.

Capítulo 7

7. Conclusiones

El Software (RecAM) implementado cumple satisfactoriamente con los requerimientos de performance impuestos:

- Tiempo de reconocimiento debe ser menor que 1 segundo.
- Extracción del número de matrícula correctamente en un porcentaje alto de casos (más de un 70%).
- Robustez, ya sea frente a las condiciones de luminosidad, suciedad, nitidez, o ruido.
- Tolerancia a variaciones en la distancia de la cámara al vehículo,
- Tolerancia al ángulo de inclinación de la matricula.

Se logró desarrollar un software en tiempo real lo suficientemente robusto como para funcionar correctamente bien en situaciones normales. El tiempo requerido para los procesamientos es muy bajo. El balance entre el tiempo requerido para obtener el resultado y efectividad en el reconocimiento es muy bueno.

Es un software que se puede adaptar a diferentes tipos de matrícula. Por ejemplo, para reconocer matrículas que no tengan 3 letras y 4 números o que no sean de 7 símbolos, basta con cambiar la configuración del software en el archivo "recam.conf". En el caso de que la fuente de los caracteres de la matrícula cambie, solamente es necesario cambiar el template.

Este proyecto constituye un punto de partida para trabajos futuros. Por ejemplo, se pueden mejorar o cambiar los algoritmos de reconocimiento de caracteres utilizando otras técnicas como redes neuronales o mezclando varias, para brindar mayor robustez al software. También es posible completar el sistema incorporando el módulo de adquisición de imágenes, interacción con una base de datos, etc.

Apéndices

Apéndice A

Detector de bordes Sobel

El detector de bordes encuentra los limites de los objetos presentes en una imagen en escala de grises.

El algoritmo utilizado para detectar los bordes consiste en buscar los lugares en la imagen en los cuales la intensidad del píxel cambia rápidamente, mediante la técnica del gradiente (derivada primera).

El criterio a utilizar es encontrar los lugares donde el gradiente de la intensidad tiene una magnitud mayor que la de un umbral predefinido.

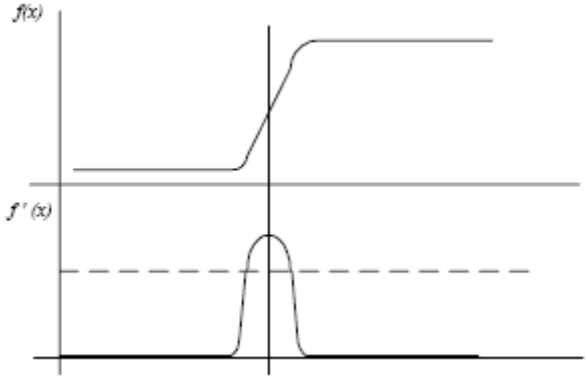


Fig. A.1 Intensidad vs. posición

El gradiente de una imagen siempre apunta en la dirección de máximo crecimiento y además es perpendicular al borde de la imagen.

El operador gradiente G aplicado a una imagen $f(x,y)$ esta definido como:

$$\nabla f(x,y) = [Gx \quad Gy] = \left[\frac{\partial f(x,y)}{\partial x} \quad \frac{\partial f(x,y)}{\partial y} \right]$$

El detector de bordes se basa en :

$|\nabla f(x,y)|$ detector de bordes no direccional

$\left| \frac{\partial f(x,y)}{\partial x} \right|$ detector de bordes en dirección vertical

$\left| \frac{\partial f(x,y)}{\partial y} \right|$ detector de bordes en dirección horizontal

El vector gradiente representa el cambio máximo de intensidad para el punto (x,y) donde:

su magnitud esta dada por $|\nabla f| = \sqrt{Gx^2 + Gy^2}$

y su dirección por $\theta = \arctan\left(\frac{Gy}{Gx}\right)$ con la dirección del gradiente perpendicular al borde.

Para reducir el costo computacional (o sea, por eficiencia), la magnitud se calcula mediante

$$|\nabla f| = |Gx| + |Gy|$$

Las mascaras referentes al operador Sobel para la detección de bordes son las siguientes:

-1	0	1
-2	0	2
-1	0	1

Gx

Este operador deriva en la dirección horizontal para detectar bordes en la dirección horizontal y realiza un suavizado (promedio) en la dirección vertical para suprimir el ruido de la imagen.

-1	-2	-1
0	0	0
1	2	1

Gy

Este operador deriva en la dirección vertical para detectar bordes en la dirección vertical y realiza un promedio en la dirección horizontal.

Debido a que el operador Sobel realiza un promedio de la imagen, para poder lograr este efecto utilizando matrices, es necesario multiplicar a la matriz correspondiente por $\frac{1}{8}$

$$\frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \frac{1}{8}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

luego se hace la convolución de la matriz correspondiente a la imagen original para obtener como resultado el gradiente de la imagen.

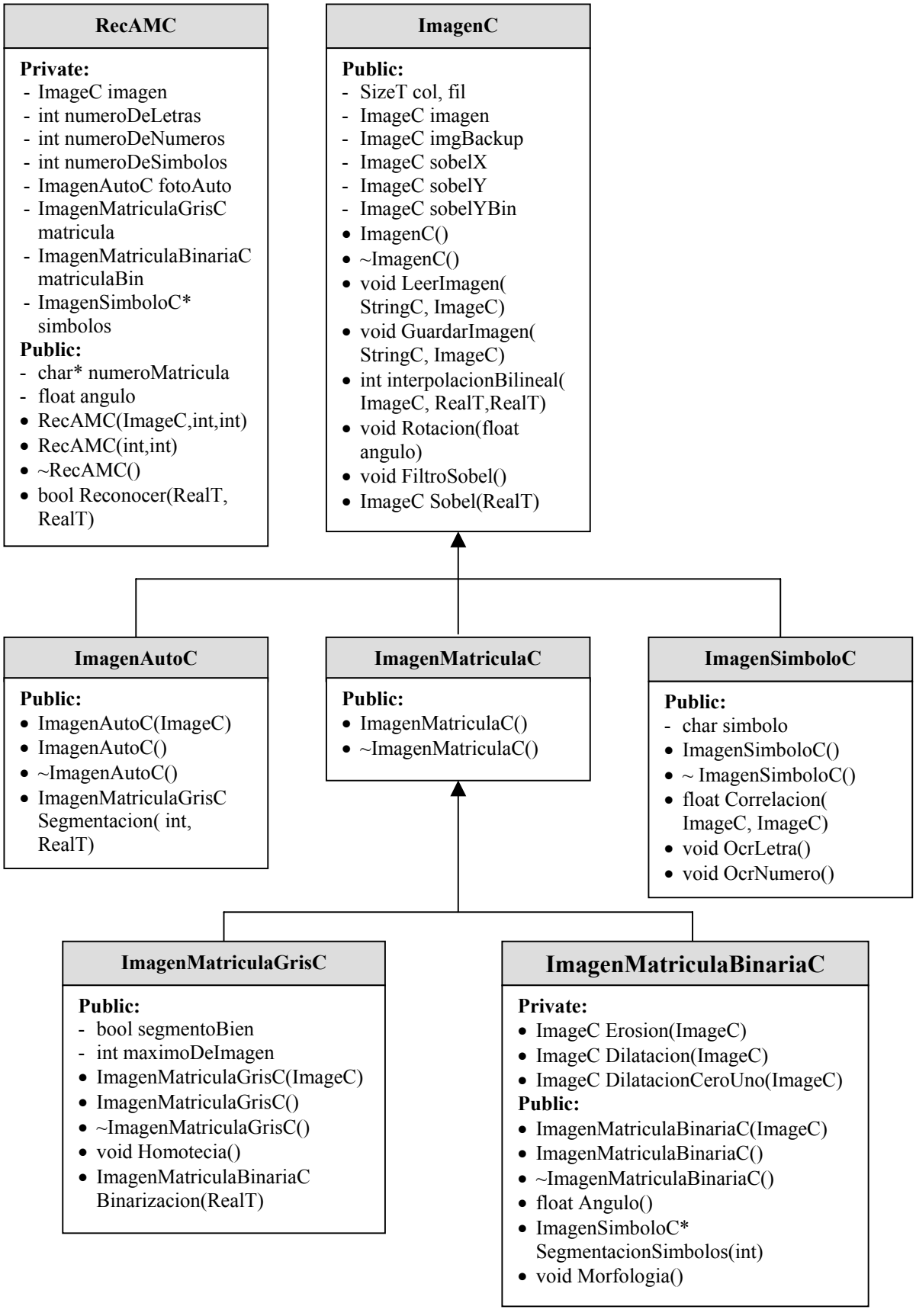
Calculando la magnitud absoluta del gradiente y representándola en una imagen, el usuario puede visualizar los resultados obtenidos.

Apéndice B

Diagrama de Clases

En este apéndice se muestra el diagrama de las clases creadas en el proyecto RecAM, cada clase tiene detallados sus atributos y métodos.

Fig. B.1 Diagrama de Clases



Apéndice C

Deducción teórica (Mínimos cuadrados)

Se quiere ajustar con una recta a una cierta cantidad de puntos (Centroides), para ello se utilizo una regresión lineal (mínimos cuadrados).

Ec. de la recta: $\hat{y}_i = a + b \cdot x_i$

Para calcular los coeficientes a y b utilizamos la condición de que el $Error(a,b) = \sum_{i=1}^{n} (y_i - a - b \cdot x_i)^2$, donde y_i e x_i son las coordenadas de los n centroides, sea mínimo.
Si existe el mínimo, este es único.

El mínimo del error se da donde se anula la derivada, por tanto:

$$\frac{\partial Error}{\partial a} = -2 \cdot \left(\sum_{i=1}^{n} (y_i - a - b \cdot x_i) \right) = 0$$

$$\Rightarrow \boxed{a = \frac{\sum_{i=1}^{n} y_i - b \cdot \sum_{i=1}^{n} x_i}{n}}$$

$$\frac{\partial Error}{\partial b} = -2 \cdot \left(\sum_{i=1}^{n} (y_i - a - b \cdot x_i) \cdot x_i \right) = 0$$

$$\Rightarrow b = \frac{\sum_{i=1}^{n} x_i \cdot y_i - a \cdot \sum_{i=1}^{n} x_i}{\sum_{i=1}^{n} {x_i}^2}$$

$$\Rightarrow \boxed{b = \frac{\sum_{i=1}^{n} x_i \cdot y_i - \frac{\sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} y_i}{n}}{\sum_{i=1}^{n} x_i^{\,2} - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n}}}$$ y por tanto como $b = tan(angulo)$

$$\Rightarrow \boxed{angulo = Atan(b) \cdot \frac{180}{\pi}}$$ (en grados)

Apéndice D

Otros Softwares (Comerciales) Disponibles

SeeCar LRP System - Hi Tech Solutions, Israel.
(http://www.htsol.com/Products/SeeCar.html)

Alphatec License Plate Reader - Alphatec, USA.
(http://www.alphatech.com/secondary/techpro/ISD/product.html)

GeoVIsion LPR - USA.
(http://www.rfconcepts.co.uk/licence_plate_recognition.htm)

AVISTA - CRS, USA.
(http://www.crs-its.com/main.htm)

PrimeVision License Plate Recognition Software - Holanda.
(http://www.roadtraffic-technology.com/contractors/detection/prime_vision)

LPRWare - ZAMIR, Israel.
(http://www.zamir.com/lpr.html)

Apéndice E

Paquetes de Software Utilizados

El proyecto RecAM fue desarrollado en C++, utilizando una biblioteca para tratamiento de imágenes llamada RAVL.

La RAVL (Recognition and Vision Library) es una biblioteca desarrollada en C++ que provee una serie de herramientas de reconocimiento de patrones, visión de computador, además de otras herramientas de tratamiento de imágenes.

El objetivo de RAVL es hacer publico software desarrollado en el Centro de Visión, Habla y Procesamiento de Señales (Centre for Vision, Speech and Signal Processing, CVSSP) de la Universidad de Surrey en Inglaterra que había sido desarrollado exclusivamente para uso interno de investigación y pruebas. De esta manera se busca fomentar su uso y aplicación en un entorno más amplio.

RAVL es una versión pública de una biblioteca utilizada internamente en el CVSSP llamada AMMA, la cual contiene más de 700.000 líneas de código en C++.

Sus características más notables son:

- su calidad de software libre.
- su facilidad de uso e interfaz de usuario (no es necesario por ejemplo el uso de punteros).
- su simple sistema de compilado que lo hace adecuado tanto para construir proyectos pequeños como grandes, su potente mecanismo de manejo de datos, que permite utilizar distintos formatos de archivos y conversión de tipos de manera relativamente sencilla e intuitiva.
- sus buenas herramientas para desarrollo de interfaces gráficas actuando como 'wrappers' para la biblioteca GTK.
- y por ultimo, su característica modular que permite separar o utilizar partes del software dependiendo de la aplicación.

Actualmente la RAVL esta siendo usada por un pequeño número de organizaciones alrededor del mundo. Ellas son: Centre for Vision, Speech and Signal Proccessing (Universidad de Surrey, Inglaterra), Omniperception Ltd, y Advanced Technology Laboratories, Lockheed Martin.

RAVL está pensada para trabajar sobre una gran variedad de plataformas y compiladores:

Sistema Operativo	Procesador	Compilador
Linux	i386	GNU gcc (3.3 recomendado)
Solaris	Sparc	GNU gcc (3.3 recomendado)
IRIX	Mips	MIPS Pro
Windows	i386	Microsoft Visual C++ (6.0 o mayor)

Para compilar la RAVL, si se piensa utilizar MVC++, es necesaria la instalación del ultimo Service Pack y modificación del registro de Windows de forma tal que sea posible el uso de archivos con extensión '.cc' y '.hh'. Existe una utilidad llamada CCFix que puede ser descargada de la pagina de la RAVL y permite realizar este cambio.

Cabe notar que si bien con estas salvedades la biblioteca compilará en Windows, existen otros problemas funcionales que son tratados en el capítulo 6 'Dificultades'. No obstante a continuación se muestra la situación actual en cuanto a la compatibilidad con diferentes sistemas operativos y compiladores.

		Plataforma/Compilador				
		Linux			Windows	
Modulo	Status	gcc 2.95.3	gcc 2.96	gcc 3.2.1	VC++ 6	VC++ .NET
3D	Alpha					
Core	Beta					
GUI	80%					
Image	20%					
Logic						
Math	30%					
OS	Beta					
Threads	Beta					
PatternRec						
QMake	Beta				n/a	n/a
SourceTools	Beta					

Los rectángulos en gris oscuro indican compatibilidad total, en gris claro indican algún tipo de incompatibilidad y los negros incompatibilidad total. Los demás aún no han sido probados.

Adicionalmente la RAVL utiliza algunas bibliotecas externas, particularmente para manejo de imágenes. Estas bibliotecas deben estar instaladas en el sistema para su correcto funcionamiento:

Biblioteca	Uso
LibPNG	Manejo de Imágenes
Zlib	Manejo de Imágenes
LibJPEG	Manejo de Imágenes
LibTIFF	Manejo de Imágenes
GTK	GUI
cURL	Manejo URL

Para los sistemas Windows, son necesarios como mínimo los siguientes archivos:

- **RAVL-SRC-0.7.zip**
- **MSVCPP-ExternalLibs-0.7.zip**

Y para desarrollar aplicaciones sobre Linux que utilicen la RAVL son necesarios como mínimo los siguientes archivos:

- **RAVL-(version).rpm**
- **RAVL-devel-(version).i386.rpm**
- **local-gcc-2.95.3.i386.rpm**

En cambio, si solo se quisiera correr aplicaciones basadas en la RAVL, solo seria necesario:

- **RAVL-(version).rpm**

Estos archivos, se encuentran disponibles para ser descargados en la pagina web de la RAVL.

Los archivos disponibles para descargar de la web actualmente son:

RAVL-SRC-0.7.tar.bz2 **RAVL-SRC-0.7.tar.gz** **RAVL-SRC-0.7.zip**	Código fuente completo para la RAVL, incluyendo los archivos de proyecto para Visual C++.
RAVL-Contrib-SRC-0.7.tar.bz2 **RAVL-Contrib-SRC-0.7.tar.gz** **RAVL-Contrib-SRC-0.7.zip**	Fuente para drivers específicos de algún componente de hardware en particular.
RAVL-(version).src.rpm	El código fuente.
RAVL-(version).i386.rpm	Bibliotecas binarias y compartidas.
RAVL-devel-(version).i386.rpm	Bibliotecas de desarrollo.
RAVL-doc-(version).i386.rpm	Documentación de desarrollo.
RAVL-debug-(version).i386.rpm	Bibliotecas de debug estático. Este código contiene chequeos extra y información de debug completa.
RAVL-opt-(version).i386.rpm	Bibliotecas estáticas optimizadas.
MSVCPP-ExternalLibs-0.7.zip	Bibliotecas externas necesitadas por RAVL. Deben ser descomprimidas dentro de la raíz de la estructura de directorios del código.

Existe también una version compilada de gcc-2.95.3.

Local-gcc-2.95.3-1.i386.rpm	gcc-2.95.3 compilada dentro de /usr/local/gcc-2.95.3, es necesario ya que el compilador 2.96 que contienen muchas distribuciones Linux no manejan C++ adecuadamente.

Una de las cualidades de la RAVL es su simple método de compilación. Esto se realiza a través del modulo de QMake.

El QMake esta basado en el 'make' GNU. Tiene la ventaja también de que puede ser usado no solo para compilar la RAVL sino cualquier otro proyecto de cualquier magnitud.

QMake se apoya en un simple archivo 'defs.mk' que debe estar presente en cada directorio del proyecto. Cada 'defs.mk' describe que archivo de ese directorio debe ser compilado.

Dentro de este archivo debe haber una línea que indique cual es el programa a compilar (el main del proyecto), otra línea que indique los archivos .cc a relacionar (linking), y otro indicando las bibliotecas necesarias para compilar el código.

Un ejemplo es el siguiente:

```
MAINS = main.cc

MUSTLINK = RecAM.cc Imagen.cc ImagenAuto.cc

USESLIBS = Auto
```

A continuación se muestra a modo de ejemplo como escribir y compilar un programa simple con RAVL:

Primeramente se debe crear un directorio y dentro de el dos archivos. El primero, 'micodigo.cc' con el siguiente código.

```
// micodigo.cc

#include "Ravl/Option.hh"
#include "Ravl/IO.hh"
#include "Ravl/Image/Image.hh"
#include "Ravl/Image/ImgIO.hh"

using namespace RavlImageN;
using namespace RavlN;

int main(int nargs,char **argv) {
  OptionC opt(nargs,argv);
  StringC infile = opt.String("i","infile.ppm","Archivo de entrada");
  opt.Check();

  ImageC<ByteT> img;
  if(!Load(infile,img)) {
    cerr << "Fallo al cargar imagen " << infile << "\n";
    return 1;
  }

  //.... Procesamiento de la imagen aqui ....

  return 0;
}
```

A continuación se le debe indicar al QMake que se quiere crear un archivo ejecutable. QMake usa un archivo llamado 'defs.mk' que define los archivos a utilizar en la compilación.
En este ejemplo, el archivo 'defs.mk' debería estar compuesto por la línea: MAINS = micodigo.cc indicando que el programa principal del código es 'micodigo.cc'

A continuación, sobre el directorio se debe correr qm y el sistema de compilación creara el archivo ejecutable como la salida del proyecto.
Al correr qm, se debería observar algo parecido a lo siguiente:

```
% qm
--- Making dir
/ejemplo/prog/share/RAVL/Admin/linux/depend/local/local/None
--- Making dir
/ejemplo/prog/qm//ejemplo/prog//linux/local/None/check/objs
--- Compile check micodigo.cc
--- Binary dependency micodigo
--- Making dir /ejemplo/prog/lib/RAVL/linux/bin
--- Making dir /ejemplo/prog/bin
--- Creating redirect for micodigo.
--- Linking check micodigo (/ejemplo/prog/lib/RAVL/linux/bin/micodigo)
```

Finalmente, ya se esta en condiciones de ejecutar el programa. La manera mas sencilla de hacerlo es cortar y pegar el path dado en la etapa de linking anterior, en este caso:

```
/ejemplo/prog/lib/RAVL/linux/bin/micodigo
```

Lo que hace el programa anterior es tomar argumentos desde la línea de comandos y cargar una imagen.

Bibliografía

[1] gti - Curso de Grado de Tratamiento de Imágenes por Computadora.
Facultad de Ingeniería – Universidad de la República (UDELAR)
Instituto de Ingeniería Eléctrica (IIE)
http://iie.fing.edu.uy/investigacion/grupos/gti/index.php3

[2] Charles Galambos. "RAVL, Recognition And Vision Library". http://ravl.sourceforge.net

[3] F. Martín Rodríguez, X. Fernández Hermida. "Reconocedor Automático de Matrículas de Automóviles". Proceedings of TIARP-01, (2001).

[4] F. Martín Rodríguez, X. Fernández Hermida. "An OCR for VLP's (Vehicle License Plate)". Proceedings of ICSPAT-97. San Diego. (Setiembre 1997).

[5] V. Sergey, R. Alexander, S. Roman. "Moving Car License Plate Recognition, Final Report". Technion-Israel Institute of Technology.

[6] J. Brown, A. Harris, M. Noury, A. Patel, J. Rafferty. "License Plate Recognition System. Formal Proposal". Georgia Institute of Technology. (Setiembre 2002).

[7] J. Molina, J. M. Mossi, A. Albiol. "Development of a Plate. Reader for Surveillance System". Universidad Politécnica de Valencia.

[8] N. Ketelaars. "Final project: Automated License Plate Recognition". (Enero 2001).

[9] P. Ponce, S. Wang, D. Wang. "Final Report: License Plate Recognition"

[10] J. F. Canny. "A Computational Approach to Edge Detection" IEEE Trans. Pattern Analysis and Machine Intelligence,8:679-698. 1986.

[11] Sorin Draghici. "A Neural Network Based Artificial Vision System for License Plate Recognition". Dept. of Computer Science, Wayne State University. Proceedings of IJNS-97, (1997).

[12] Y. V. Linnik. "Méthode des Moindres Carrés. Éléments de la Théorie du Traitement Statistique des Observations". Dunod. (Paris 1963).

[13] Bjarne Stroustrup, "The C++ Programming Language". Second Edition. Addison-Wesley publishing company. 1991.

[14] Deitel y Deitel, "C++ Cómo Programar". Segunda Edición. Prentice may. 1999.

[15] M. C. José Jaime Esqueda Elizondo. "Fundamentos de Procesamiento de Imágenes". Universidad Autónoma de Baja California, Unidad Tijuana. Noviembre 2002.

[16] E.O. Piedrahíta, J.P. Urrea Duque "Implementación de la Transformada de Hough para la detección de líneas para un sistema de visión de bajo nivel". Universidad Nacional de Colombia. Facultad de Ingeniería y Arquitectura. Depto. De Ing. Eléctrica, Electrónica y Computación. (2002)

[17] Kok Kiaw Teo. "Low Cost Number Plate Recognition". The University of Queensland, School of Information Technology and Electrical Engineering. October 2003.

[18] A. Alahi, A. Menghrajani. "Pattern Recognition Theory: License Plate Recognition". Carnegie Mellon University (2003)

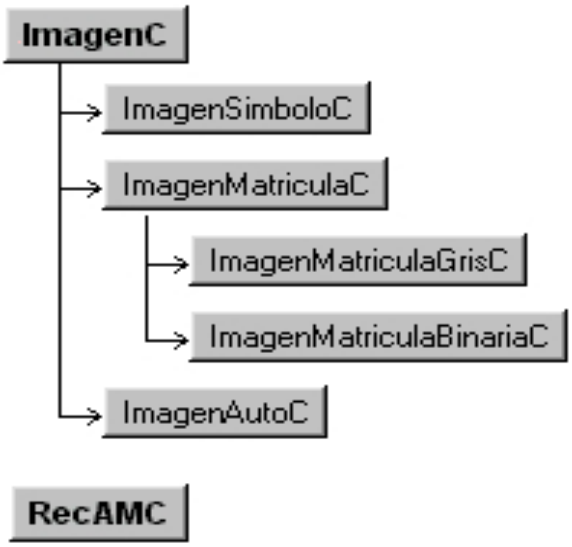
[19] H. Cohen, G.Bergman. "Car License Plate Recognition". Faculty of Electrical Engineering, Israel.

[20] D.Chanson, T. Roberts. "License Plate Recognition System". Manukau Institute of Technology, Auckland.

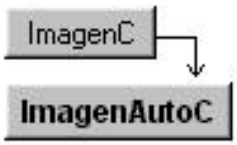
[21] V. P. Thokal y Prof. J. S. Deshpande. "Neural Network Based Optical Character Recognition", Information Sheet. B.N. College of Engg, Pusad Dist. Yavatmal And College of Engg, Badnera Dist. Amravati.

Indice

Herencia de clases



1. Clase ImagenAutoC



Métodos Públicos

- **ImagenAutoC**(ImageC<ByteT> img)
  Constructor 1, con argumentos
- **ImagenAutoC**()
  Constructor 2
- **~ImagenAutoC**()
  Destructor
- ImagenMatriculaGrisC **Segmentacion**(int numeroDeSimbolos, RealT porcentajeBinarizacion)
  Metodo para segmentar la matricula

Heredado de ImagenC:

Campos Públicos

- ImageC<ByteT> **imagen**
- ImageC<ByteT> **imgBackup**
- SizeT **col**
- SizeT **fil**
- ImageC<ByteT> **sobelX**
- ImageC<ByteT> **sobelY**
- ImageC<ByteT> **sobelYBin**

Métodos Públicos

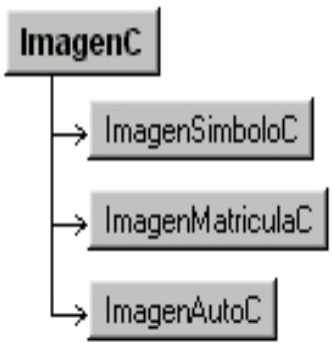
- int **MaximoArray**(int array[], int size)
- int **MinimoArray**(int array[], int size)
- int **Maximo**(ImageC<ByteT> imag)
- int **Minimo**(ImageC<ByteT> imag)
- void **LeerImagen**(StringC archivo, ImageC<ByteT> img)
- void **GuardarImagen**(StringC archivo, ImageC<ByteT> img)
- ImageC<IntT> **Sobel**(RealT porcentajeBinarizacion)
- void **FiltroSobel**()
- int **interpolacionBilineal**(ImageC<ByteT> img, RealT i, RealT j)
- void **Rotacion**(float angulo)

Documentación

- **`ImagenAutoC(ImageC<ByteT> img)`**
  Constructor 1, con argumentos
- **`ImagenAutoC()`**
  Constructor 2
- **`~ImagenAutoC()`**
  Destructor
- **ImagenMatriculaGrisC `Segmentacion(int numeroDeSimbolos, RealT porcentajeBinarizacion)`**
  Metodo para segmentar la matricula

No exiten clases que hereden de esta clase.

2. Clase ImagenC



Campos Públicos

- ImageC<ByteT> **imagen**
  La Imagen
- ImageC<ByteT> **imgBackup**
  La Imagen de respaldo, se utiliza por ejemplo cuando se aplica morfología
- SizeT **col**
  Cantidad de columnas en la imagen
- SizeT **fil**
  Cantidad de filas en la imagen
- ImageC<ByteT> **sobelX**
  Guarda el Sobel segun X
- ImageC<ByteT> **sobelY**
  Guarda el Sobel segun Y
- ImageC<ByteT> **sobelYBin**
  Guarda el Sobel segun Y Binarizado

Métodos Públicos

- **ImagenC**()
  Constructor
- **~ImagenC**()
  Destructor
- int **MaximoArray**(int array[], int size)
  Calcula el máximo elemento de un array
- int **MinimoArray**(int array[], int size)
  Calcula el mínimo elemento de un array
- int **Maximo**(ImageC<ByteT> imag)
  Calcula el máximo valor de la matriz de la imagen
- int **Minimo**(ImageC<ByteT> imag)
  Calcula el mínimo valor de la matriz de la imagen
- void **LeerImagen**(StringC archivo, ImageC<ByteT> img)
  Lee la imagen desde un archivo y la guarda en img
- void **GuardarImagen**(StringC archivo, ImageC<ByteT> img)
  Salva la imagen img en un archivo
- ImageC<IntT> **Sobel**(RealT porcentajeBinarizacion)

Aplica el filtro Sobel sobre imagen y da como salida el Sobel con valores de 1, -1 o 0.

- void **FiltroSobel**()
  Aplica el filtro Sobel sobre imagen y guarda los resultados en sobelX y sobelY
- int **interpolacionBilineal**(ImageC<ByteT> img, RealT i, RealT j)
  Aplica interpolación bilineal
- void **Rotacion**(float angulo)
  Aplica una rotación de un ángulo en grados a la imagen y guarda el resultado en imagen

---

Documentación

- **`ImageC<ByteT> imagen`**
  La Imagen
- **`ImageC<ByteT> imgBackup`**
  La Imagen de respaldo, se utiliza por ejemplo cuando se aplica morfología
- **`SizeT col`**
  Cantidad de columnas en la imagen
- **`SizeT fil`**
  Cantidad de filas en la imagen
- **`ImageC<ByteT> sobelX`**
  Guarda el Sobel segun X
- **`ImageC<ByteT> sobelY`**
  Guarda el Sobel segun Y
- **`ImageC<ByteT> sobelYBin`**
  Guarda el Sobel segun Y Binarizado
- **`ImagenC()`**
  Constructor
- **`~ImagenC()`**
  Destructor
- **`int MaximoArray(int array[], int size)`**
  Calcula el máximo elemento de un array
- **`int MinimoArray(int array[], int size)`**
  Calcula el mínimo elemento de un array
- **`int Maximo(ImageC<ByteT> imag)`**
  Calcula el máximo valor de la matriz de la imagen
- **`int Minimo(ImageC<ByteT> imag)`**
  Calcula el mínimo valor de la matriz de la imagen
- **`void LeerImagen(StringC archivo, ImageC<ByteT> img)`**
  Lee la imagen desde un archivo y la guarda en img
- **`void GuardarImagen(StringC archivo, ImageC<ByteT> img)`**
  Salva la imagen img en un archivo
- **`ImageC<IntT> Sobel(RealT porcentajeBinarizacion)`**
  Aplica el filtro Sobel sobre imagen y da como salida el Sobel con valores de 1, -1 o 0. Además guarda el resultado en sobelY y sobelYBin.
- **`void FiltroSobel()`**
  Aplica el filtro Sobel sobre imagen y guarda los resultados en sobelX y sobelY
- **`int interpolacionBilineal(ImageC<ByteT> img, RealT i, RealT j)`**
  Aplica interpolación bilineal

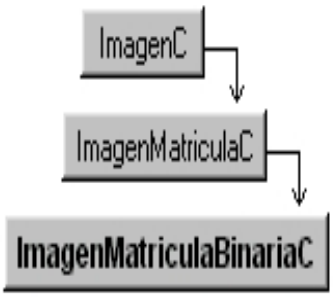
- **`void Rotacion(float angulo)`**
  Aplica una rotación de un ángulo en grados a la imagen y guarda el resultado en imagen

---

Clases que heredan directamente:

ImagenSimboloC
ImagenMatriculaC
ImagenAutoC

3. Clase ImagenMatriculaBinariaC



Métodos Públicos

- **ImagenMatriculaBinariaC**(ImageC<ByteT> img)
  Constructor 1, con argumentos
- **ImagenMatriculaBinariaC**()
  Constructor 2
- **~ImagenMatriculaBinariaC**()
  Destructor
- void **Morfologia**()
  Morfología, consiste en una Erosion, toma la imagen y realiza un backup, sobreescribe el atributo imagen.
- float **Angulo**()
  Calcula el ángulo en grados a rotar la matrícula para dejarla horizontal
- ImagenSimboloC* **SegmentacionSimbolos**(int numeroDeSimbolos)
  Actua sobre la imagen, segmenta los simbolos y los guarda en el array simbolos

---

Heredado de ImagenMatriculaC:

Heredado de ImagenC:

Campos Públicos

- ImageC<ByteT> **imagen**
- ImageC<ByteT> **imgBackup**
- SizeT **col**
- SizeT **fil**
- ImageC<ByteT> **sobelX**
- ImageC<ByteT> **sobelY**
- ImageC<ByteT> **sobelYBin**

Métodos Públicos

- int **MaximoArray**(int array[], int size)
- int **MinimoArray**(int array[], int size)
- int **Maximo**(ImageC<ByteT> imag)
- int **Minimo**(ImageC<ByteT> imag)

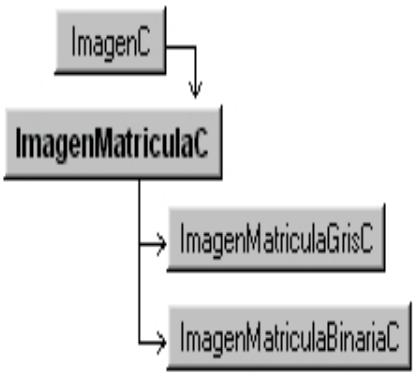
- void **LeerImagen**(StringC archivo, ImageC<ByteT> img)
- void **GuardarImagen**(StringC archivo, ImageC<ByteT> img)
- ImageC<IntT> **Sobel**(RealT porcentajeBinarizacion)
- void **FiltroSobel**()
- int **interpolacionBilineal**(ImageC<ByteT> img, RealT i, RealT j)
- void **Rotacion**(float angulo)

Documentación

- **`ImagenMatriculaBinariaC(ImageC<ByteT> img)`**
  Constructor 1, con argumentos
- **`ImagenMatriculaBinariaC()`**
  Constructor 2
- **`~ImagenMatriculaBinariaC()`**
  Destructor
- **`void Morfologia()`**
  Morfología, consiste en una Erosion, toma la imagen y realiza un backup, sobreescribe el atributo imagen
- **`float Angulo()`**
  Calcula el ángulo en grados a rotar la matrícula para dejarla horizontal
- **`ImagenSimboloC* SegmentacionSimbolos(int numeroDeSimbolos)`**
  Actua sobre la imagen, segmenta los símbolos y los guarda en el array símbolos.

No existen clases que hereden de esta clase.

4. Clase ImagenMatriculaC



Métodos Públicos

- **ImagenMatriculaC**()
  Constructor
- **~ImagenMatriculaC**()
  Destructor

Heredado de ImagenC:

Campos Públicos

- ImageC<ByteT> **imagen**
- ImageC<ByteT> **imgBackup**
- SizeT **col**
- SizeT **fil**
- ImageC<ByteT> **sobelX**
- ImageC<ByteT> **sobelY**
- ImageC<ByteT> **sobelYBin**

Métodos Públicos

- int **MaximoArray**(int array[], int size)
- int **MinimoArray**(int array[], int size)
- int **Maximo**(ImageC<ByteT> imag)
- int **Minimo**(ImageC<ByteT> imag)
- void **LeerImagen**(StringC archivo, ImageC<ByteT> img)
- void **GuardarImagen**(StringC archivo, ImageC<ByteT> img)
- ImageC<IntT> **Sobel**(RealT porcentajeBinarizacion)
- void **FiltroSobel**()
- int **interpolacionBilineal**(ImageC<ByteT> img, RealT i, RealT j)
- void **Rotacion**(float angulo)

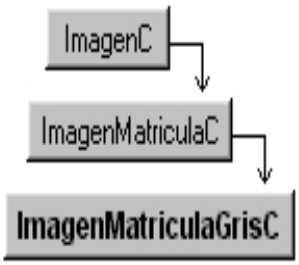
Documentación

- **`ImagenMatriculaC()`**
  Constructor
- **`~ImagenMatriculaC()`**
  Destructor

Clases que heredan:

ImagenMatriculaGrisC
ImagenMatriculaBinariaC

5. Clase ImagenMatriculaGrisC



Campos Públicos

- bool **segmentoBien**
  Variable que indica si fue posible segmentar bien
- int **maximoDeImagen**
  Variable que contiene el máximo de la imagen

Métodos Públicos

- **ImagenMatriculaGrisC**(ImageC<ByteT> img)
  Constructor 1, con argumentos
- **ImagenMatriculaGrisC**()
  Constructor 2
- **~ImagenMatriculaGrisC**()
  Destructor
- void **Homotecia**()
  Homotecia para uniformizar el tamaño de las matrículas
- ImagenMatriculaBinariaC **Binarizacion**(RealT porcentajeBinMatricula)
  Se aplica sobre la imagen de la clase y devuelve la matrícula binaria

---

Heredado de ImagenMatriculaC:

Heredado de ImagenC:

Campos Públicos

- ImageC<ByteT> **imagen**
- ImageC<ByteT> **imgBackup**
- SizeT **col**
- SizeT **fil**
- ImageC<ByteT> **sobelX**
- ImageC<ByteT> **sobelY**
- ImageC<ByteT> **sobelYBin**

Métodos Públicos

- int **MaximoArray**(int array[], int size)
- int **MinimoArray**(int array[], int size)

- int **Maximo**(ImageC<ByteT> imag)
- int **Minimo**(ImageC<ByteT> imag)
- void **LeerImagen**(StringC archivo, ImageC<ByteT> img)
- void **GuardarImagen**(StringC archivo, ImageC<ByteT> img)
- ImageC<IntT> **Sobel**(RealT porcentajeBinarizacion)
- void **FiltroSobel**()
- int **interpolacionBilineal**(ImageC<ByteT> img, RealT i, RealT j)
- void **Rotacion**(float angulo)

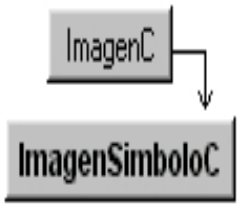
Documentación

- **`bool segmentoBien`**
  Variable que indica si fue posible segmentar bien
- **`int maximoDeImagen`**
  Variable que contiene el máximo de la imagen
- **`ImagenMatriculaGrisC(ImageC<ByteT> img)`**
  Constructor 1, con argumentos
- **`ImagenMatriculaGrisC()`**
  Constructor 2
- **`~ImagenMatriculaGrisC()`**
  Destructor
- **`void Homotecia()`**
  Homotecia para uniformizar el tamaño de las matrículas. Se aplica sobre la matrícula en niveles de gris. Se utiliza una interpolación bilineal.
- **`ImagenMatriculaBinariaC Binarizacion(RealT porcentajeBinMatricula)`**
  Se aplica sobre la imagen de la clase y devuelve la matrícula binaria

---

No existen clases que hereden de esta clase.

6. Clase ImagenSimboloC



Métodos Públicos

- **ImagenSimboloC**()
  Constructor
- **~ImagenSimboloC**()
  Destructor
- float **Correlacion**(ImageC<ByteT> imgsimbolo, ImageC<IntT> imgpatron)
  Calcula la correlacion
- void **OcrLetra**()
  Algoritmo de reconocimiento de caracteres para las letras
- void **OcrNumero**()
  Algoritmo de reconocimiento de caracteres para los números

Heredado de ImagenC:

Campos Públicos

- ImageC<ByteT> **imagen**
- ImageC<ByteT> **imgBackup**
- SizeT **col**
- SizeT **fil**
- ImageC<ByteT> **sobelX**
- ImageC<ByteT> **sobelY**
- ImageC<ByteT> **sobelYBin**

Métodos Públicos

- int **MaximoArray**(int array[], int size)
- int **MinimoArray**(int array[], int size)
- int **Maximo**(ImageC<ByteT> imag)
- int **Minimo**(ImageC<ByteT> imag)
- void **LeerImagen**(StringC archivo, ImageC<ByteT> img)
- void **GuardarImagen**(StringC archivo, ImageC<ByteT> img)
- ImageC<IntT> **Sobel**(RealT porcentajeBinarizacion)
- void **FiltroSobel**()
- int **interpolacionBilineal**(ImageC<ByteT> img, RealT i, RealT j)
- void **Rotacion**(float angulo)

Documentación

- **`char simbolo;`**
  Variable con el caracter del símbolo
- **`ImagenSimboloC()`**
  Constructor
- **`~ImagenSimboloC()`**
  Destructor
- **`float Correlacion(ImageC<ByteT> imgsimbolo, ImageC<IntT> imgpatron)`**
  Calcula la correlacion
- **`void OcrLetra()`**
  Algoritmo de reconocimiento de caracteres para las letras
- **`void OcrNumero()`**
  Algoritmo de reconocimiento de caracteres para los números

No existen clases que hereden de esta clase.

7. Clase RecAMC

Campos Públicos

- float **angulo**
  Se define el ángulo a rotar
- char* **numeroMatricula**
  Se define un puntero a un string que va a tener el número de la matrícula

Métodos Públicos

- **RecAMC**(ImageC<ByteT> img, int letras, int numeros)
  Constructor 1
- **RecAMC**(int letras, int numeros)
  Constructor 2
- **~RecAMC**()
  Destructor
- bool **Reconocer**(RealT porcentajeBinarizacion, RealT porcentajeBinMatricula)
  Reconoce la matrícula y la guarda en numeroMatricula

Documentación

- **`float angulo`**
  Se define el ángulo a rotar
- **`char* numeroMatricula`**
  Se define un puntero a un string que va a tener el número de la matrícula
- **`RecAMC(ImageC<ByteT> img, int letras, int numeros)`**
  Constructor 1
- **`RecAMC( int letras, int numeros)`**
  Constructor 2
- **`~RecAMC()`**
  Destructor
- **`bool Reconocer(RealT porcentajeBinarizacion, RealT porcentajeBinMatricula)`**
  Reconoce la matrícula y la guarda en numeroMatricula

No existen clases que hereden de esta clase.