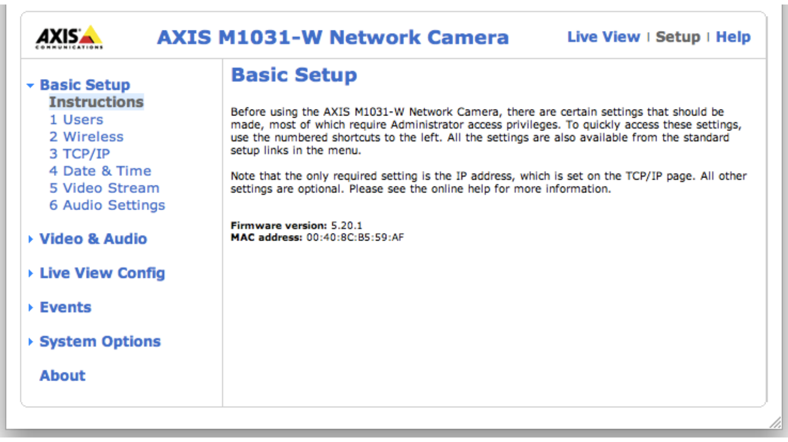
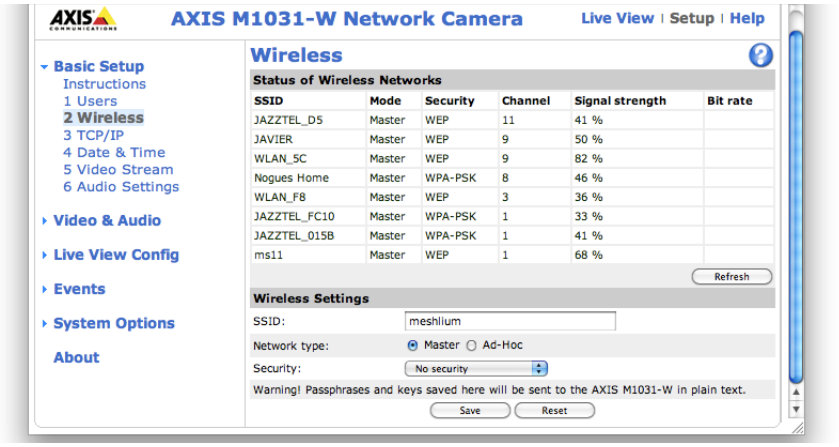
ANEXOS

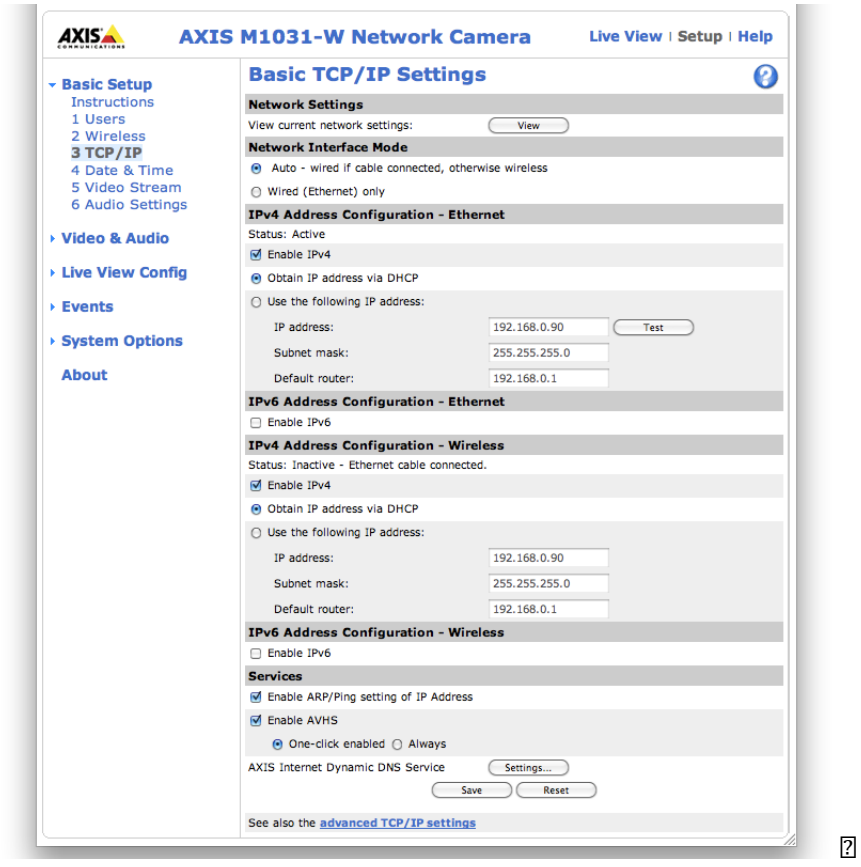
INDICE ANEXOS

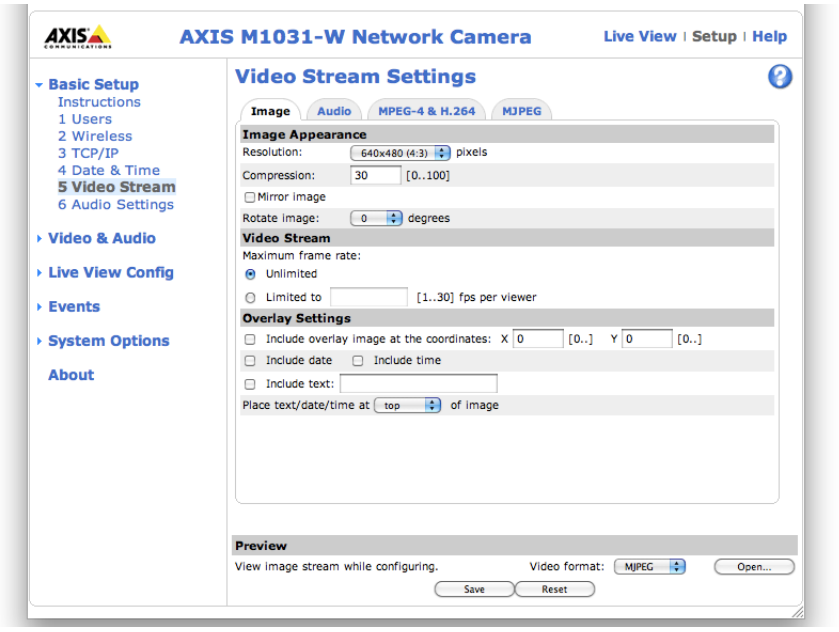
Anexo

Configuración de la cámara IP inalámbrica


AXIS
AXIS M1031-W Network Camera
Live View | Setup | Help
Basic Setup
Instructions
1 Users
2 Wireless
3 TCP/IP
4 Date & Time
5 Video Stream
6 Audio Settings
Video & Audio
Live View Config
Events
System Options
About
Basic Setup
Before using the AXIS M1031-W Network Camera, there are certain settings that should be made, most of which require Administrator access privileges. To quickly access these settings, use the numbered shortcuts to the left. All the settings are also available from the standard setup links in the menu.
Note that the only required setting is the IP address, which is set on the TCP/IP page. All other settings are optional. Please see the online help for more information.
Firmware version: 5.20.1
MAC address: 00:40:8C:B5:59:AF


AXIS M1031-W Network Camera
Live View | Setup | Help
Basic Setup
Instructions
1 Users
2 Wireless
3 TCP/IP
4 Date & Time
5 Video Stream
6 Audio Settings
Video & Audio
Live View Config
Events
System Options
About
Wireless
Status of Wireless Networks
SSID Mode Security Channel Signal strength Bit rate
JAZZTEL_D5 Master WEP 11 41 %
JAVIER Master WEP 9 50 %
WLAN_5C Master WEP 9 82 %
Nogues Home Master WPA-PSK 8 46 %
WLAN_F8 Master WEP 3 36 %
JAZZTEL_FC10 Master WPA-PSK 1 33 %
JAZZTEL_015B Master WPA-PSK 1 41 %
ms11 Master WEP 1 68 %
Refresh
Wireless Settings
SSID: meshlium
Network type: Master Ad-Hoc
Security: No security
Warning! Passphrases and keys saved here will be sent to the AXIS M1031-W in plain text.
Save Reset

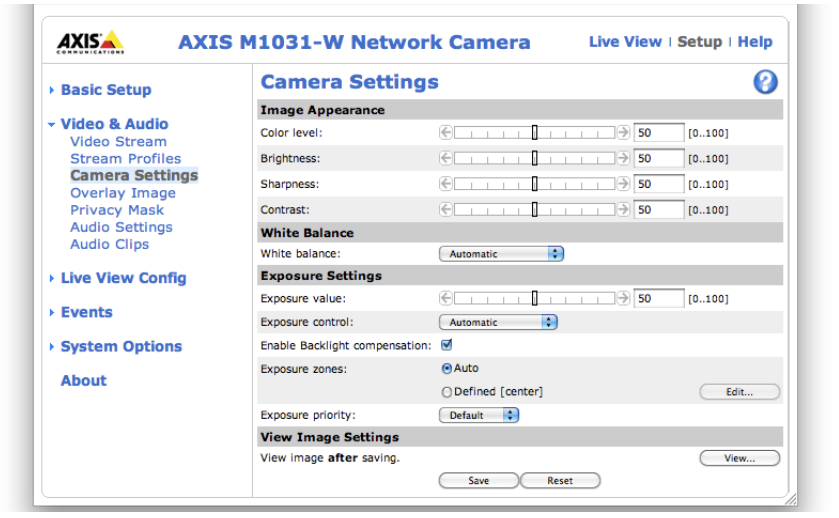

AXIS M1031-W Network Camera
Live View | Setup | Help
Basic Setup
Instructions
1 Users
2 Wireless
3 TCP/IP
4 Date & Time
5 Video Stream
6 Audio Settings
Video & Audio
Live View Config
Events
System Options
About
Basic TCP/IP Settings
Network Settings
View current network settings: View
Network Interface Mode
Auto - wired if cable connected, otherwise wireless
Wired (Ethernet) only
IPv4 Address Configuration - Ethernet
Status: Active
Enable IPv4
Obtain IP address via DHCP
Use the following IP address:
IP address: 192.168.0.90 Test
Subnet mask: 255.255.255.0
Default router: 192.168.0.1
IPv6 Address Configuration - Ethernet
Enable IPv6
IPv4 Address Configuration - Wireless
Status: Inactive - Ethernet cable connected.
Enable IPv4
Obtain IP address via DHCP
Use the following IP address:
IP address: 192.168.0.90
Subnet mask: 255.255.255.0
Default router: 192.168.0.1
IPv6 Address Configuration - Wireless
Enable IPv6
Services
Enable ARP/Ping setting of IP Address
Enable AVHS
One-click enabled Always
AXIS Internet Dynamic DNS Service Settings...
Save Reset
See also the advanced TCP/IP settings


AXIS M1031-W Network Camera
Live View | Setup | Help
Basic Setup
Instructions
1 Users
2 Wireless
3 TCP/IP
4 Date & Time
5 Video Stream
6 Audio Settings
Video & Audio
Live View Config
Events
System Options
About
Video Stream Settings
Image
Audio
MPEG-4 & H.264
MJPEG
Image Appearance
Resolution:
640x480 (4:3)
pixels
Compression:
30
[0..100]
Mirror image
Rotate image:
0
degrees
Video Stream
Maximum frame rate:
Unlimited
Limited to
[1..30] fps per viewer
Overlay Settings
Include overlay image at the coordinates: X
[0..]
Y
[0..]
Include date
Include time
Include text:
Place text/date/time at
top
of image
Preview
View image stream while configuring.
Video format:
MJPEG
Open...
Save
Reset

[illegible]

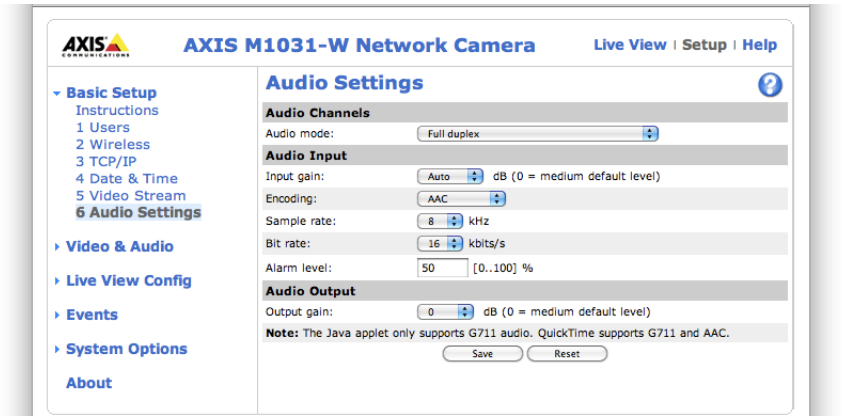
[illegible]

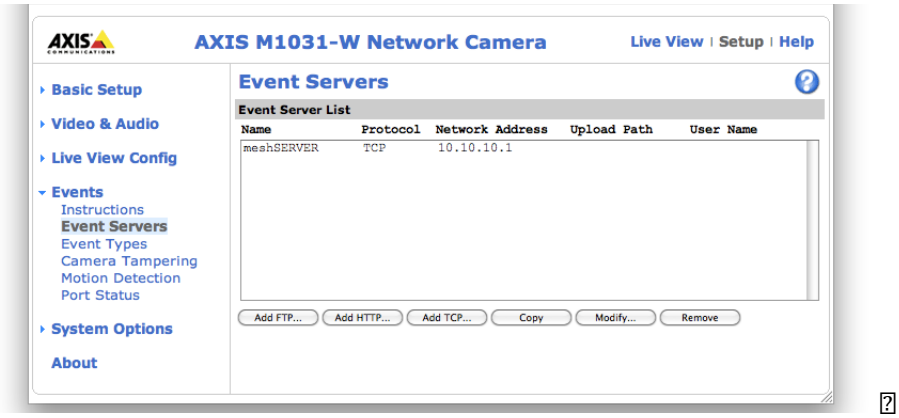
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]

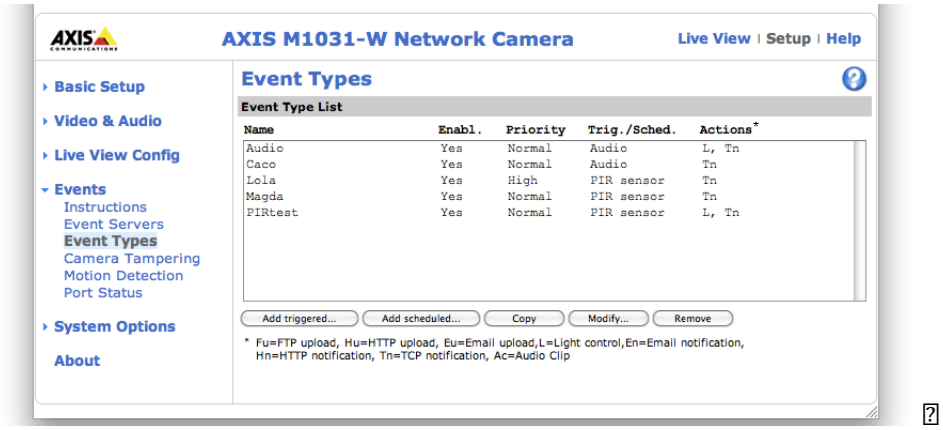

AXIS M1031-W Network Camera
Live View | Setup | Help
Basic Setup
Video & Audio
Video Stream
Stream Profiles
Camera Settings
Overlay Image
Privacy Mask
Audio Settings
Audio Clips
Live View Config
Events
System Options
About
Camera Settings
Image Appearance
Color level:
50
[0..100]
Brightness:
50
[0..100]
Sharpness:
50
[0..100]
Contrast:
50
[0..100]
White Balance
White balance:
Automatic
Exposure Settings
Exposure value:
50
[0..100]
Exposure control:
Automatic
Enable Backlight compensation:
Exposure zones:
Auto
Defined [center]
Edit...
Exposure priority:
Default
View Image Settings
View image after saving.
View...
Save
Reset

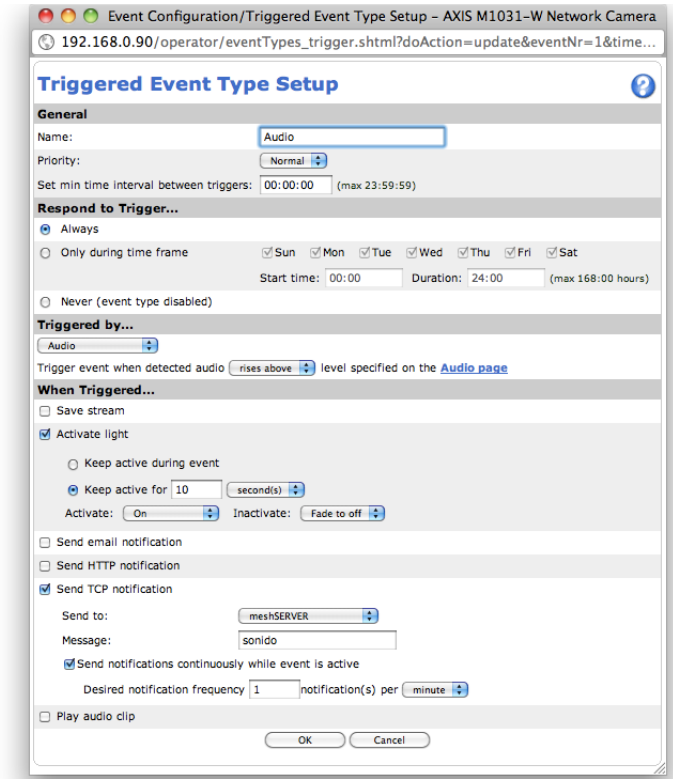
[illegible]

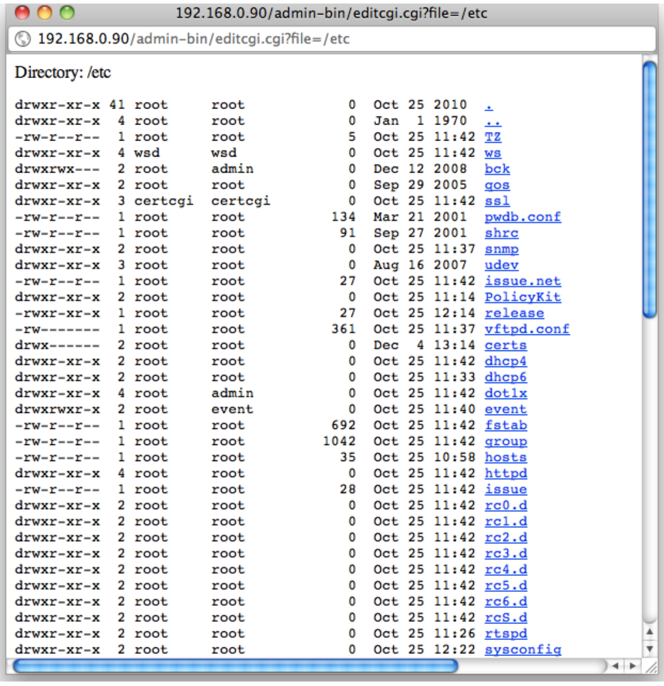
[illegible]


AXIS M1031-W Network Camera
Live View | Setup | Help
Basic Setup
Instructions
1 Users
2 Wireless
3 TCP/IP
4 Date & Time
5 Video Stream
6 Audio Settings
Video & Audio
Live View Config
Events
System Options
About
Audio Settings
Audio Channels
Audio mode: Full duplex
Audio Input
Input gain: Auto dB (0 = medium default level)
Encoding: AAC
Sample rate: 8 kHz
Bit rate: 16 kbits/s
Alarm level: 50 [0..100] %
Audio Output
Output gain: 0 dB (0 = medium default level)
Note: The Java applet only supports G711 audio. QuickTime supports G711 and AAC.
Save
Reset


AXIS M1031-W Network Camera
Live View | Setup | Help
Basic Setup
Video & Audio
Live View Config
Events
Instructions
Event Servers
Event Types
Camera Tampering
Motion Detection
Port Status
System Options
About
Event Servers
Event Server List
Name
Protocol
Network Address
Upload Path
User Name
meshSERVER
TCP
10.10.10.1
Add FTP...
Add HTTP...
Add TCP...
Copy
Modify...
Remove


AXIS M1031-W Network Camera
Live View | Setup | Help
Basic Setup
Video & Audio
Live View Config
Events
Instructions
Event Servers
Event Types
Camera Tampering
Motion Detection
Port Status
System Options
About
Event Types
Event Type List
Name
Enabl.
Priority
Trig./Sched.
Actions*
Audio
Caco
Lola
Magda
PIRtest
Yes
Normal
High
PIR sensor
Audio
L, Tn
Tn
Add triggered...
Add scheduled...
Copy
Modify...
Remove
* Fu=FTP upload, Hu=HTTP upload, Eu=Email upload,L=Light control,En=Email notification,
Hn=HTTP notification, Tn=TCP notification, Ac=Audio Clip


Event Configuration/Triggered Event Type Setup – AXIS M1031-W Network Camera
192.168.0.90/operator/eventTypes_trigger.shtml?doAction=update&eventNr=1&time...
Triggered Event Type Setup
General
Name:
Audio
Priority:
Normal
Set min time interval between triggers:
00:00:00
(max 23:59:59)
Respond to Trigger...
Always
Only during time frame
Sun
Mon
Tue
Wed
Thu
Fri
Sat
Start time:
00:00
Duration:
24:00
(max 168:00 hours)
Never (event type disabled)
Triggered by...
Audio
Trigger event when detected audio
rises above
level specified on the
Audio page
When Triggered...
Save stream
Activate light
Keep active during event
Keep active for
10
second(s)
Activate:
On
Inactivate:
Fade to off
Send email notification
Send HTTP notification
Send TCP notification
Send to:
meshSERVER
Message:
sonido
Send notifications continuously while event is active
Desired notification frequency
1
notification(s) per
minute
Play audio clip
OK
Cancel



OX

1e E1b s x t1s sti 1 e1 te1 eE t
tI p í 1e x t1 1 s e a e1 1 s
Ee1E s st b e1 s 1 v s a
u e1 e g s x t1í

s Ee1E s s i s s a te e1
1 s Ee1E s f s x l s e1
1 te1 s e1 s t e s t s 1 p 1

e 1 a s st g e p 1 1 s e1 s
st 1 9 e P p EEá 1 1 E1 s x e p 1 e s
Ee1E s i e te1 a Ee1 e í 1 Ee1 1 a a 1 t1
E e e s 1 a g Eta e i e u 1 E e1 s a E e a
1 s 1 s t t1 p a e e i e í e 1 i e
1 t1 g 1 1 e 1 1 e a

c E1 1 s e a E e ea 1 s 1 a ste1 Ee1 e p o m

e a l m t , a s e Ue11t , E sso 1e Ue11t ttE I . 2í. 2í. 2í. . I v s,
1 I t 1 te1 L t 1 U . m 22

E e a l m t , a s e Ue11t , E sso 1e Ue11t ttE I . 2í. 2í. 2í. . I v s,
1 I t 1 te1 L t 1 U . m 2

Ee1 a e 1 1m t , a s e Ue11t , E sso 1e Ue11t ttE I . 2í. 2í. 2í. . I v s,
E p Eí L EUh

1 1s e Ee1 a e e t s 1 1s st 1 a st a e h E1e b 1 qí

Anexo

Comunicación con la cámara

ANEXO – COMUNICACIÓN CON LA CÁMARA IP INALÁMBRICA

Aquí voy a desglosar el código del 'receptor + preparador'.

Hablo de preparador porque la parte de análisis es muy extensa y profunda y la trato por separado en otros anexos.

Básicamente consta de dos partes principales:

1. Interacción con la cámara IP Inalámbrica.
    a. Recepción de señales TCP procedentes de la cámara
    b. Descarga de imágenes de la cámara.
    c. Comunicación con el proceso de análisis
2. Preparación de las imágenes.
    a. Recepción de mensaje con datos de imagen descargada
    b. Cambio de nombre y ubicación

En resumen se trata de dos procesos, uno para tramitar las señales enviadas por la cámara y otro para el análisis. Para comunicar dichos procesos uso una 'cola de mensajes'. Comunico entre ambos procesos el nombre de la imagen, que esta guardada en la carpeta /IMÁGENES.

La parte principal del proceso de interacción con la cámara IP Inalámbrica es un servidor de peticiones TCP. Dentro del bucle servidor, cada vez que recibimos una alarma nos descargamos la imagen de la cámara y la guardamos en la carpeta imágenes dentro de su correspondiente fecha y mediante una cola de mensajes comunicamos el nombre de la imagen al proceso de análisis.

Código del programa principal:

```
int main(int argc, char** argv)
{
      pid_t idProceso;        // Identificador del proceso creado

      int estadoHijo;         // Estado devuelto por el hijo

      // Se crea el proceso hijo.
      idProceso = fork();

      // Si fork() devuelve -1, es que hay un error
      if (idProceso == -1)
      {
        cerr << "No se puede crear proceso" << endl;
        return -1;
      }

      // fork() devuelve 0 al proceso hijo.
      if (idProceso == 0)
      {
        //-------------------------------------------------------
        //     R      E      C      E      P      T      O      R
        //-------------------------------------------------------

        emisor rcp;

        bool correcto = false;

        logs_debug("", 8);
        logs_debug("--> EJECUCIÓN RECEPTOR.", 8);
        logs_debug("", 8);
```

```
        logs("", 8);
        logs("--> EJECUCION RECEPTOR.", 8);
        logs("", 8);

        rcp.asignar_estado(0);

        // Parte Cola de Mensajes - Iniciamos cola
        //-----------------------------------------
        rcp.crear_clave();

        rcp.crear_cola();

        // Parte Socket - Recepción de peticiones http
        //---------------------------------------------
        rcp.iniciar_socket();

        correcto = rcp.servidor();

        rcp.cerrar_socket();

        // Contestación - Sólo se hace en las pruebas como mera comprobación.
        //      mensaje.recibir_mensaje1();
        //      mensaje.mostrar_mensaje( 1 );

        rcp.cerrar_cola();

        if( correcto )
        {
                cout << "Todo perfecto." << endl;
        }
        else
        {
                cout << "Ha ocurrido un error." << endl;
        }

        return 32;
      }

      // fork() devuelve un número positivo al padre. Este número es el id del hijo.
      if (idProceso > 0)
      {
        //-------------------------------------------------------
        //      A       N       A       L       I       Z       A       R
        //-------------------------------------------------------

        logs_debug("", 8);
        logs_debug("--> EJECUCIÓN ANALIZADOR.", 8);
        logs_debug("", 8);
        logs("", 8);
        logs("--> EJECUCION ANALIZADOR.", 8);
        logs("", 8);

        // variables de las clases de COLA DE MENSAJES
        cola_mensajes message;

        // Comandos
        message.crear_clave();
        message.crear_cola();

        IplImage* src;

        // Bucle infinito
        //-----------------
        while( 1 )
        {
                message.recibir_mensaje2();

                if( message.buffer.size() != 0 )
                {
                        // Tratamiento para el detección de personas
                        //-------------------------------------------
                        string ruta;
                        ruta = ruta_imagen();
                        ruta += message.Un_Mensaje.Nombre;
                        ruta += ".jpg";
```

```
                        src = cvLoadImage( ruta.c_str() ); // Cargamos la imagen

                        logs("HIJO -> Imagen guardada.", 3 );
                        message.buffer.pop_back();

                        string save;
                        save = ruta_fichero_debug();
                        save += "bien.jpg";

                        cvSaveImage(save.c_str(), src);

                        // Mera comprobación
                        // ver_imagen( src );
                }
        }

        logs("--> FIN TRATAMIENTO DE LA IMAGEN.", 3);
        logs_debug("--> FIN TRATAMIENTO DE LA IMAGEN.", 3);

        // Espera que el hijo muera.
        wait (&estadoHijo); // --> ESTO IRÁ DENTRO DEL BUCLE

        // Comprueba la salida del hijo.
        if (WIFEXITED(estadoHijo) != 0)
        {
                printf ("Padre : Mi hijo ha salido. Devuelve %d\n",
WEXITSTATUS(estadoHijo));
                // break; // Salgo del bucle para terminar el programa.
        }

      }

      return 0;
}
```

En este código podemos ver como el proceso padre es donde realizamos las labores de recepción y en el proceso hijo donde realizaremos el análisis. En el código que vemos arriba no esta el código de análisis porque ya se analiza en otra parte (ANEXOS detección de personas y detección de matrículas), pero básicamente consiste en introducir la imagen '*src*' a las secuencias de detección de gente y detección de matrículas.

En los siguientes puntos se explica tanto el servidor de mensajes TCP como la cola de mensajes. El servidor TCP se usa en el proceso de comunicación con la cámara, proceso padre, mientras que la cola de mensajes se utiliza en ambos procesos.

Servidor de mensajes TCP

Selecciono como puerto a escuchar el 32032 y el mismo en la configuración de la cámara AXIS.

A partir de este punto comenzamos con la descripción del código C/C++ de los sockets. Los pasos que debe seguir un programa servidor son los siguientes:

- **Apertura de un socket**, mediante la función **socket()**. Esta función devuelve un descriptor de fichero normal, como puede devolverlo open(). La función socket() no hace absolutamente nada, salvo devolvernos y preparar un descriptor de fichero que el sistema posteriormente asociará a una conexión en red.

- **Avisar al sistema operativo** de que hemos abierto un socket y queremos que asocie nuestro programa a dicho socket, mediante la función **bind()**. El sistema todavía no atenderá a las conexiones de clientes, simplemente anota que cuando empiece a hacerlo, tendrá que avisarnos a nosotros. Es en esta llamada cuando se debe indicar el número de servicio al que se quiere atender.
- Avisar al sistema de que **comience a atender dicha conexión** de red. Se consigue mediante la función **listen()**. A partir de este momento el sistema operativo anotará la conexión de cualquier cliente para pasárnosla cuando se lo pidamos. Si llegan clientes más rápido de lo que somos capaces de atenderlos, el sistema operativo hace una "buffer" con ellos y nos los irá pasando según vayamos pidiéndolo.
- Pedir y **aceptar las conexiones** de clientes al sistema operativo. Para ello hacemos una llamada a la función **accept()**. Esta función le indica al sistema operativo que nos dé la siguiente petición dl buffer. Si no hay peticiones se quedará bloqueado hasta que llegue otra petición.
- **Recibir datos** del cliente (cámara) por medio de la funcio **read()**, que es exactamente la misma que usamos para leer de un fichero.
- **Cierre de la comunicación** y del socket, por medio de la función **close()**, que es la misma que sirve para cerrar un fichero.

```
bool emisor::servidor()
{
      const int MAX_MSG = 19;
      const int LINE_ARRAY_SIZE = ( MAX_MSG + 1);

      int listenSocket, connectSocket, i;
      unsigned short int listenPort;
      socklen_t clientAddressLength;
      struct sockaddr_in clientAddress, serverAddress;
      char line[LINE_ARRAY_SIZE];

      if( (estado == DEBUG) || (estado_num == 2) )
      {
               cout << "EMISOR - Servidor." << endl;
        logs_debug("EMISOR - Servidor", 4);
      }
    else if(estado_num != 2)
      {
        logs_debug("EMISOR - Servidor.", 4);
      }

      // Puerto por el que escuchar.
      //--------------------------------
      if ( !ejecucion )
      {
        cout << "Enter port number to listen on (between 1500 and 65000): ";
        cin >> listenPort;
      }
      else
        listenPort = 32032;

      // ESP Crea un socket de escucha para las peticiones de conexión del cliente.
      listenSocket = socket(AF_INET, SOCK_STREAM, 0); // -> TCP
      //listenSocket = socket(AF_INET, SOCK_DGRAM, 0);     // --> UDP
      if (listenSocket < 0)
      {
        if ( estado_num != 0 )
        {
               cerr << "Cannot create listen socket" << endl;
```

```
                logs_debug("Cannot create listen socket", 1);
                return 1;
        }
        else
        {
                logs_debug("Cannot create listen socket", 1);
                return 1;
        }
      }

      // ESP Bind escucha el socket hasta el puerto.  Primero rellena varios campos en
la dirección del servidor (estructura serverAddress), entonces llama a bind().
      // htonl() y htons() convierten enteros largos a enteros cortos (respectivamente)
de byte de orden del cliente (en x86 este es el primer byte menos significativo
      // a orden de byte de red (Primer byte más significativo).
      serverAddress.sin_family = AF_INET;
      serverAddress.sin_addr.s_addr = htonl(INADDR_ANY);
      serverAddress.sin_port = htons(listenPort);

      if (bind(listenSocket, (struct sockaddr *) &serverAddress, sizeof(serverAddress))
< 0)
      {
        if ( estado_num != 0 )
        {
                cerr << "Cannot bind socket" << endl;
                logs_debug( "Cannot bind socket", 3);
                return false;
        }
        else
        {
                logs_debug( "Cannot bind socket", 3);
                return false;
        }
      }

      // ESP Espera por conexiones de los clientes. Es una llamada no bloqueante;
      listen(listenSocket, 5);

      string nombre;
      gestion webcam;

      webcam.construir();

      // Bucle para recibir las peticiones HTTP
      //-----------------------------------------
      while (1)
      {
        if( estado_num != 0 )
        {
                cout << "Waiting for TCP connection on port " << listenPort << " ...\n";

                logs_debug( ("Waiting for TCP connection on port " + int2s(listenPort) +
" …").c_str(), 3);
        }
        else
                logs_debug( ("Waiting for TCP connection on port " + int2s(listenPort) +
" …").c_str(), 3);

        // ESP Acepta una conexión cuando el cliente la requiere. La llamadaThe accept()
es una llamada bloquenate.
        clientAddressLength = sizeof(clientAddress);
        connectSocket = accept(listenSocket, (struct sockaddr *) &clientAddress,
&clientAddressLength);
        if (connectSocket < 0)
        {
                if( estado_num != 0 )
                {
                        cerr << "Cannot accept connection " << endl;
                        logs_debug("Cannot accept connection ", 1);
                        return false;
                }
                else
                {
                        logs_debug("Cannot accept connection ", 1);
                        return false;
```

```
                }
        }

        if( estado_num != 0 )
        {
                // ESP muestra la direccion IP del cliente. inet_ntoa() convierte la
direccion IP de binario a la numeración estandar de número y puntos.
                cout << "  connected to " << inet_ntoa(clientAddress.sin_addr);

                // ESP Muestra el número de puerto del cliente. ntohs() convierte un
integer pequeño de un byte the orden de la red (que es el primer byte más significativo)
hasta el byte del cliente (que en x86, por ejemplo, es el primer byte menos
significativo).
                cout << ":" << ntohs(clientAddress.sin_port) << "\n";

//              logs_debug( ("  connected to " +
int2s(inet_ntoa(clientAddress.sin_addr)) ).c_str(), 3);
                logs_debug( (":" +  int2s(ntohs(clientAddress.sin_port) ) ).c_str(), 3);
        }
        else
        {
//              logs_debug( ("  connected to " +
int2s(inet_ntoa(clientAddress.sin_addr)) ).c_str(), 3);
                logs_debug( (":" + int2s(ntohs(clientAddress.sin_port) ) ).c_str(), 3);
        }

        // ESP Lee las lineas del socket, usando recv(), guardadolas en un vector. Si no
hay mensjaes disponibles, recv() bloquea
        // hasta que llega uno. Primero pone la línea todo a cero, entonces sabremos
donde está el final de la cadena.
        memset(line, 0x0, LINE_ARRAY_SIZE);
        while (recv(connectSocket, line, MAX_MSG, 0) > 0)
        {
                if( estado_num != 0 )
                {
                        cout << "  --  " << line << "\n";
                        logs_debug(line, 3);

                        // Meter mensaje en la cola para que ANALIZADOR lo analice.
                        //---------------------------------------------------------------

                        nombre = dar_hora2();

                        // Capturamos la imagen de la Webcam
                        webcam.capturar_imagen( nombre );

                        rellenar_mensaje( 1, 4, line, strdup(nombre.c_str()) ); // -->
                        enviar_mensaje();
                }
                else
                {
                        logs_debug(line, 3);
                        nombre = dar_hora2();

                        // Capturamos la imagen de la Webcam
                        webcam.capturar_imagen( nombre );

                        // Meter mensaje en la cola para que ANALIZADOR lo analice.
                        //---------------------------------------------------------------
                        rellenar_mensaje( 1, 4, line, strdup(nombre.c_str()) ); // -->
                        enviar_mensaje();
                }

                memset(line, 0x0, LINE_ARRAY_SIZE);  // set line to all zeroes
        }
      }

      return true;
}
```

Destacar el método *webcam.capturar_imgen(nombre)* que es el encargado de descargar la imagen de la cámara AXIS, renombrarla a la hora actual y moverla a la carpeta correspondiente a la fecha actual.

Cola de Mensajes

Explicación del método para comunicar los 2 procesos. Primero defino la estructura de los mensajes que voy a mandar de un proceso a otro.

```
struct Mi_Tipo_Mensaje
      {
       long Id_Mensaje;
       int Dato_Numerico;
       char Mensaje[20];
       char Nombre[10];
       };
```

De este *struct* nos interesa principalmente el nombre, que corresponderá a la hora en que ha saltado la alarma en la cámara y que dará nombre a un archivo de imagen. El nombre es la unión de hora-minutos-segundos sin separación entre cada uno de los marcadores, por ejemplo si la alarma salta a las 10.53.15 el archivo de imagen se llamará *105315.jpg* y el atributo nombre del *struct* contendrá la cadena de caracteres *105315.*

Explicación en detalle de una cola de mensajes.

Pasos a seguir para trabajar con una cola de mensajes:

- En primer lugar necesitamos conseguir una clave, de tipo **key_t**, que sea común para todos los programas que quieran compartir la cola de mensajes. Para ello existe la función **key_t ftok (char *, int)**. A dicha función se le pasa un fichero que exista y sea accesible y un entero. Normalmente como primer parámetro se pasa algún fichero del sistema que sepamos seguro de su existencia, como por ejemplo "/bin/ls", que es el "ls" del unix.

- Una vez obtenida la clave, se crea la cola de mensajes. Para ello está la función **int msgget (key_t, int)**. Con dicha función creamos la cola y nos devuelve un identificador para la misma. Si la cola correspondiente a la Clave key_t ya estuviera creada, simplemente nos daría el identificador de la misma (siempre y cuando los parámetros no indiquen lo contrario). El primer parámetro es la clave key_t obtenida anteriormente y que debería ser la misma para todos los programas. El segundo parámetro son unos flags. Aunque hay más posibilidades, lo imprescindible es:
  - 9 bits menos significativos, son permisos de lectura/escritura/ejecución para propietario/grupo/otros, como sucede con los ficheros. Para obtener una cola con todos los permisos para todo el mundo, debemos poner como parte de los flags el número **0777**. Es importante el cero delante, para que el número se interprete en octal y queden los bits en su sitio (En C, cualquier número que empiece por cero, se considera octal). El de ejecución no tiene sentido y se ignora.
  - **IPC_CREAT**. Junto con los bits anteriores, este bit indica si se debe crear la cola en caso de que no exista. Si está puesto, la cola se creará

si no lo está ya y se devolverá el identificador. Si no está puesto, se intentará obtener el identificador y se obtendrá un error si no está ya creada.

En resumen, los flags deberían ser algo como 0777 | IPC_CREAT

- Ya estamos en condiciones de utilizar la cola de mensajes. Para meter un mensaje en la cola, se utiliza la función **msgsnd (int, struct msgbuf *, int, int)**
  - El primer parámetro entero es el identificador de la cola obtenido con **msgget()**.
  - El segundo parámetro es el mensaje en sí. El mensaje debe ser una estructura cuyo primer campo sea un **long**. En dicho *long* se almacena el tipo de mensaje. El resto de los campos pueden ser cualquier cosa que se desee enviar (otra estructura, campos sueltos, etc.). Al pasar el mensaje como parámetro, se pasa un puntero al mensaje y se le hace un "cast" a **struct msgbuf ***. No hay ningún problema en este "cast" siempre y cuando el primer campo del mensaje sea un **long**.
  - El tercer parámetro es el tamaño en bytes del mensaje exceptuando el **long**, es decir, el tamaño en bytes de los campos con la información.
  - El cuarto parámetro son flags. Aunque hay varias opciones, la más habitual es poner un 0 o bien **IPC_NOWAIT**. En el primer caso la llamada a la función queda bloqueada hasta que se pueda enviar el mensaje. En el segundo caso, si el mensaje no se puede enviar, se vuelve inmediatamente con un error. El motivo habitual para que el mensaje no se pueda enviar es que la cola de mensajes esté llena.
- Para recoger un mensaje de la cola se utiliza la función **msgrcv (int, struct msgbuf *, int, int, int)**.
  - El primer parámetro es el identificador de la cola obtenido con **msgget()**.
  - El segundo parámetro es un puntero a la estructura donde se desea recoger el mensaje. Puede ser, como en la función anterior, cualquier estructura cuyo primer campo sea un long para el tipo de mensaje.
  - El tercer parámetro entero es el tamaño de la estructura exceptuando el **long**.
  - El cuarto parámetro entero es el tipo de mensaje que se quiere retirar. Se puede indicar un entero positivo para un tipo concreto o un 0 para cualquier tipo de mensaje.
  - El quinto parámetro son flags, que habitualmente puede ser 0 o bien **IPC_NOWAIT**. En el primer caso, la llamada a la función se queda bloqueada hasta que haya un mensaje del tipo indicado. En el

segundo caso, se vuelve inmediatamente con un error si no hay mensaje de dicho tipo en la cola.

- Una vez terminada de usar la cola, se debe liberar. Para ello se utiliza la función **msgctl (int, int, struct msqid_ds *)**. Es una función genérica para control de la cola de mensajes con posibilidad de varios comandos. Aquí sólo se explica como utilizarla para destruir la cola.
    - El primer parámetro es el identificador de la cola de mensajes, obtenido con **msgget()**.
    - El segundo parámetro es el comando que se desea ejecutar sobre la cola, en este caso **IPC_RMID**.
    - El tercer parámetro son datos necesarios para el comando que se quiera ejecutar. En este caso no se necesitan datos y se pasará un **NULL**.

Método para generar la clave, imprescindible para poder crear la cola.

```
bool emisor::crear_clave()
{
      //---------------------------------------------------------------------------
      // Igual que en cualquier recurso compartido (memoria compartida, semaforos
      //  o colas) se obtien una clave a partir de un fichero existente cualquiera
      //  y de un entero cualquiera. Todos los procesos que quieran compartir este
      //  semaforo, deben usar el mismo fichero y el mismo entero.
      //---------------------------------------------------------------------------
      if( (estado == DEBUG) || (estado_num == 2) )
      {
               cout << "EMISOR + COLA - Crear clave." << endl;
        logs_debug("EMISOR + COLA - Comprobar cámara.", 4);
      }
    else if(estado_num != 2)
      {
        logs_debug("EMISOR + COLA - Comprobar cámara.", 4);
      }

      Clave1 = ftok ("/bin/ls", 33);
      if (Clave1 == (key_t)-1)
      {
        cout << "Error al obtener clave para cola mensajes" << endl;
        return false;
      }

      return true;
}
```

Método para crear la cola:

```
bool emisor::crear_cola()
{
      //---------------------------------------------------------------------------
      //       Se crea la cola de mensajes y se obtiene un identificador para ella.
      // El IPC_CREAT indica que cree la cola de mensajes si no lo está ya.
      // el 0600 son permisos de lectura y escritura para el usuario que lance
      // los procesos. Es importante el 0 delante para que se interprete en
      // octal.
      //---------------------------------------------------------------------------
      if( (estado == DEBUG) || (estado_num == 2) )
      {
               cout << "EMISOR + COLA - Crear cola." << endl;
        logs_debug("EMISOR + COLA - Crear cola.", 4);
      }
    else if(estado_num != 2)
      {
        logs_debug("EMISOR + COLA - Crear cola.", 4);
      }
```

```
      Id_Cola_Mensajes = msgget (Clave1, 0600 | IPC_CREAT);
      if (Id_Cola_Mensajes == -1)
      {
        cout << "Error al obtener identificador para cola mensajes" << endl;
        // Aqu'  debemos escribir tambi☻n mensaje a log y log_debug.

        return false;
      }

      return true;
}
```

Método para llenar el mensaje antes de enviarlo:

```
bool emisor::rellenar_mensaje(int tipo, int num, char* cadena, char* name)
{
      //------------------------------------------------------------------------
      //       Se rellenan los campos del mensaje que se quiere enviar.
      //       El Id_Mensaje es un identificador del tipo de mensaje. Luego se podrá
      //       recoger aquellos mensajes de tipo 1, de tipo 2, etc.
      //       Dato_Numerico es un dato que se quiera pasar al otro proceso. Se pone,
      //       por ejemplo 29.
      //       Mensaje es un texto que se quiera pasar al otro proceso.
      //------------------------------------------------------------------------
      if( (estado == DEBUG) || (estado_num == 2) )
      {
              cout << "EMISOR + COLA - Rellenar mensaje." << endl;
        logs_debug("EMISOR + COLA - Rellenar mensaje.", 4);
      }
    else if(estado_num != 2)
      {
        logs_debug("EMISOR + COLA - Rellenar mensaje.", 4);
      }

      if( (tipo == 1) || (tipo == 2) )
      {
        Un_Mensaje.Id_Mensaje = tipo;
      }
      else
        return false;

      Un_Mensaje.Dato_Numerico = num;
      strcpy (Un_Mensaje.Mensaje, cadena);
      strcpy (Un_Mensaje.Nombre, name);

      return true;
}
```

Método para enviar el mensaje:

```
bool emisor::enviar_mensaje()
{
      //------------------------------------------------------------------------
      //       Se envia el mensaje. Los parámetros son:
      //       - Id de la cola de mensajes.
      //       - Dirección al mensaje, convirtiéndola en puntero a (struct msgbuf *)
      //       - Tamaño total de los campos de datos de nuestro mensaje, es decir
      //       de Dato_Numerico y de Mensaje
      //       - Unos flags. IPC_NOWAIT indica que si el mensaje no se puede enviar
      //       (habitualmente porque la cola de mensajes esta llena), que no espere
      //       y de un error. Si no se pone este flag, el programa queda bloqueado
      //       hasta que se pueda enviar el mensaje.
      //------------------------------------------------------------------------
      if( (estado == DEBUG) || (estado_num == 2) )
      {
              cout << "EMISOR + COLA - Enviar mensaje." << endl;
        logs_debug("EMISOR + COLA - Enviar mensaje.", 4);
      }
    else if(estado_num != 2)
      {
        logs_debug("EMISOR + COLA - Enviar mensaje.", 4);
```

```
      }

      int devuelto;

      devuelto =      msgsnd (Id_Cola_Mensajes,
                              (struct msgbuf *)&Un_Mensaje,
                              sizeof(Un_Mensaje.Dato_Numerico) +
sizeof(Un_Mensaje.Mensaje) + sizeof(Un_Mensaje.Nombre),
                              IPC_NOWAIT);

      if(devuelto == 0)
        return true;
      else
        return false;
}
```

Método para recibir el mensaje:

```
bool emisor::recibir_mensaje2()
{
      if( (estado == DEBUG) || (estado_num == 2) )
      {
               cout << "EMISOR + COLA - Recibir mensaje 2." << endl;
        logs_debug("EMISOR + COLA - Recibir mensaje 2.", 4);
      }
    else if(estado_num != 2)
      {
        logs_debug("EMISOR + COLA - Recibir mensaje 2.", 4);
      }

      int devuelto;

      devuelto = msgrcv (Id_Cola_Mensajes,
                              (struct msgbuf *)&Un_Mensaje,
                              sizeof(Un_Mensaje.Dato_Numerico) +
sizeof(Un_Mensaje.Mensaje) + sizeof(Un_Mensaje.Nombre),
                              1,
                              0);

      // Meter el mensaje recibido en el buffer.
      buffer.push_back( Un_Mensaje );

      if(devuelto == 0)
        return true;
      else
        return false;
}
```

Unos **comandos** de UNIX que nos pueden resultar de utilidad son ipcs y ipcrm.

ipcs nos da un listado de recursos compartidos que están creados en ese momento, es decir, listado de memorias compartidas como las que hemos tratado en esta página, de semáforos y de colas.

ipcrm nos permite eliminar algunos de estos recursos. Si paramos el programa con un Ctrl-C o simplemente sale de forma anormal, el recurso (la memoria compartida) no se libera y queda en el sistema. La forma de borrarla sería con este comandos.

Es bastante normal mientras desarrollamos y depuramos nuestro programa que se nos caiga, lo abortemos, etc. El número de memorias compartidas que podemos crear está limitado en UNIX, así que a la cuarta o quinta prueba empezaremos a

obtener errores de que no se pueden crear las memorias. ipcrm nos permite eliminar las memorias que se nos han quedado "pendientes" en nuestras pruebas.

Anexo

Detección de personas

Anexo – Detección de personas

En esta parte se va a explicar la parte de detección de personas. Algo que a priori parece muy sencillo pero que posee mucha complejidad implícita. El ser humano diferencia fácilmente a otros seres de su misma especie por un proceso de aprendizaje. Un sistema de visión por computador es como un recién nacido, a priori no sabe nada y tienen que ser agentes externos los que le digan las pautas para reconocer los diferentes objetos y seres vivos que nos rodean. Para diferenciar a una persona de cualquier otra cosa usamos la detección de rostros o caras dado que es una de las partes más características del ser humano.

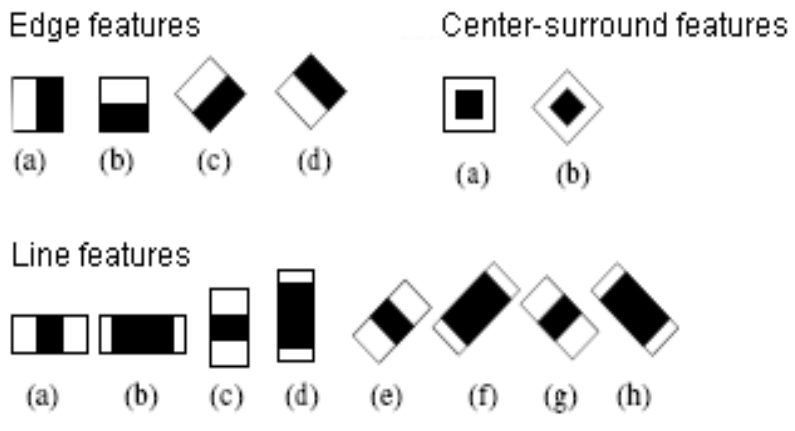
Características de las caras humanas

El rostro humano es un objeto dinámico que tiene un alto grado de variabilidad en su apariencia lo cual hace que su detección sea un problema difícil de tratar en visión por computador.

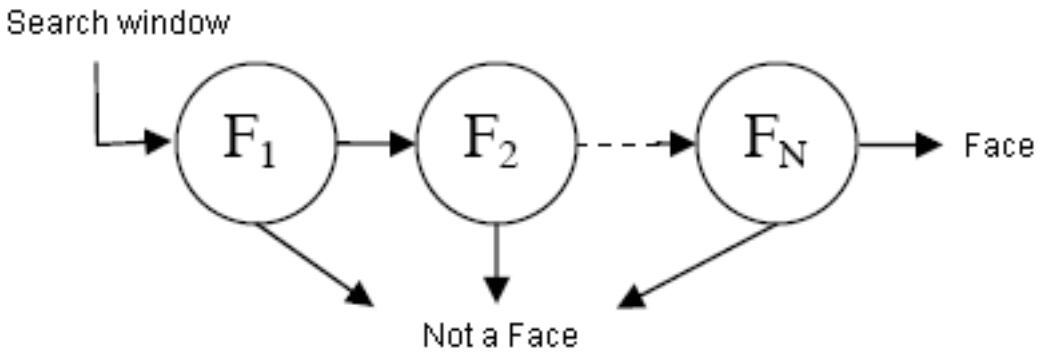
Estado del Arte sobre el reconocimiento facial

Inicialmente el problema de detección del rostro en los sistemas de reconocimiento no recibió la atención necesaria y se partía de que el rostro ya había sido detectado, fue solo en la década de los ochenta que surgieron los primeros algoritmos, basados en técnicas heurísticas y antropométricas, y en la década de los noventa cuando el desarrollo de algoritmos de detección rostros inició su crecimiento, proponiéndose una gran variedad de técnicas, desde algoritmos básicos de detección de bordes hasta algoritmos compuestos de alto nivel que utilizan métodos avanzados de reconocimiento de patrones.

Estas técnicas de detección se han abordado desde diferentes enfoques: Enfoques basados en rasgos faciales o características locales, en los que se buscan determinados elementos que componen el rostro, como los ojos, la nariz, la boca, Enfoques holísticos o basados en la imagen, en este caso los métodos trabajan con la imagen completa o zonas concretas de la misma de la cual se extraen características que puedan representar el objeto buscado, Enfoques híbridos, estos métodos usan tanto la información local como la global para la detección, basándose en el hecho de que el sistema de percepción humano distingue tanto las características locales como globales del rostro. Alrededor de los enfoques mencionados se han planteado diferentes trabajos en los que se usa la información del color de la piel para realizar la detección, obteniendo resultados alrededor del 90%, usa redes neuronales para segmentar el rostro alcanzando porcentajes de detección entre 77,9% y 90,3% para las diferentes configuraciones de la red, usa una base Haar para la extracción de características y Adaboost para la selección de estas y clasificación, alcanzando un porcentaje de detección del 94,1%, este método propuesto por Paul Viola y Michael Jones, es uno de los métodos mas usados hoy en día ya que ha permitido segmentar múltiples rostros en una imagen con tiempos de procesamiento bajos. Alrededor de esta investigación se han realizado otros trabajos como los de, que han aplicando el método para segmentar rostros y han adicionado otras etapas como la detección de características faciales (ojos y boca), corrección de pose y seguimiento del rostro.


Edge features
Center-surround features
(a)
(b)
(c)
(d)
(a)
(b)
Line features
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)

Básicamente, el proceso de detección de rostros desliza una "ventana de búsqueda" a través de la imagen, comprobar si una región de la imagen puede ser considerado como un "objeto de la cara" o no. El detector asumes una escala fija para el objeto, pero como la cara de una imagen puede ser diferente de la escala asumida, la "ventana de búsqueda" va a través de la imagen varias veces, buscando el objeto a través de un rango de tamaños.



Clasificación

Esta etapa dentro del algoritmo de detección se encarga de asignar un conjunto de características dado a una clase con la que se encuentra una mayor similitud, de acuerdo a un modelo inducido durante el entrenamiento [13].

Boosting es un método de clasificación que combina varios clasificadores básicos para formar un único clasificador más complejo y preciso. La idea se basa en la afirmación de que varios clasificadores sencillos, cada uno de ellos con una precisión ligeramente superior a una clasificación aleatoria, pueden combinarse para formar un clasificador de mayor precisión, siempre y cuando se disponga de un número suficiente de muestras de entrenamiento. La aplicación de clasificadores en cascada ha permitido obtener buenos resultados.

Para aplicar la técnica de boosting primero se debe establecer un algoritmo de aprendizaje sencillo (clasificador débil o base), que será llamado repetidas veces para crear diversos clasificadores base. Para el entrenamiento de los clasificadores base se emplea, en cada iteración, un subconjunto diferente de muestras de entrenamiento y una distribución de pesos diferente sobre las muestras de entrenamiento. Finalmente, estos clasificadores base se combinan en un único clasificador que se espera sea mucho más preciso que cualquiera de los clasificadores base por separado.

En función de los clasificadores base que se utilicen, las distribuciones que se empleen para entrenarlos y el modo de combinarlos, podrán crearse distintas clases del algoritmo genérico de boosting. El algoritmo de boosting empleado por Viola y Jones en su trabajo es conocido como AdaBoost.

En OpenCV hay definida una clase para los clasificadores e igualmente hay ya creados algunos ficheros que contienen las características propias de rasgos humanos. Aquí podemos ver una referencia completa de las características de esta clase:

```
class CascadeClassifier
{
public:
        // structure for storing tree node
    struct CV_EXPORTS DTreeNode
    {
        int featureIdx; // feature index on which is a split
        float threshold; // split threshold of ordered features only
        int left; // left child index in the tree nodes array
        int right; // right child index in the tree nodes array
    };

    // structure for storing desision tree
    struct CV_EXPORTS DTree
    {
        int nodeCount; // nodes count
    };

    // structure for storing cascade stage (BOOST only for now)
    struct CV_EXPORTS Stage
    {
        int first; // first tree index in tree array
        int ntrees; // number of trees
        float threshold; // treshold of stage sum
    };

    enum { BOOST = 0 }; // supported stage types

    // mode of detection (see parameter flags in function
HaarDetectObjects)
    enum { DO_CANNY_PRUNING = CV_HAAR_DO_CANNY_PRUNING,
           SCALE_IMAGE = CV_HAAR_SCALE_IMAGE,
           FIND_BIGGEST_OBJECT = CV_HAAR_FIND_BIGGEST_OBJECT,
           DO_ROUGH_SEARCH = CV_HAAR_DO_ROUGH_SEARCH };

    CascadeClassifier(); // default constructor
    CascadeClassifier(const string& filename);
    ~CascadeClassifier(); // destructor

    bool empty() const;
    bool load(const string& filename);
    bool read(const FileNode& node);

    void detectMultiScale( const Mat& image, vector<Rect>& objects,
                           double scaleFactor=1.1, int minNeighbors=3,
                           int flags=0, Size minSize=Size());

    bool setImage( Ptr<FeatureEvaluator>&, const Mat& );
    int runAt( Ptr<FeatureEvaluator>&, Point );

    bool is_stump_based; // true, if the trees are stumps

    int stageType; // stage type (BOOST only for now)
    int featureType; // feature type (HAAR or LBP for now)
    int ncategories; // number of categories (for categorical features
only)
    Size origWinSize; // size of training images
```

```
    vector<Stage> stages; // vector of stages (BOOST for now)
    vector<DTree> classifiers; // vector of decision trees
    vector<DTreeNode> nodes; // vector of tree nodes
    vector<float> leaves; // vector of leaf values
    vector<int> subsets; // subsets of split by categorical feature

    Ptr<FeatureEvaluator> feval; // pointer to feature evaluator
    Ptr<CvHaarClassifierCascade> oldCascade; // pointer to old cascade
};
```

Las última versiones de la librería OpenCV ofrece una completa serie de clasificadores y ficheros XML, que incluyen, entre otros, los clasificadores para las caras frontal, el perfil caras, ojos, boca, nariz, parte superior del cuerpo, parte inferior del cuerpo, etc ...

Busco la detección de personas en un principio con múltiples marcadores para diferentes partes del cuerpo:

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

En principio se busca usar el mayor número de marcadores para obtener una detección lo más fiable posible. Realizamos un análisis en profundidad para descartar los marcadores menos precisos y ver que clasificadores nos aportan más información y de cuales podemos prescindir.

Tras diferentes pruebas con múltiples imágenes se llegó a la conclusión de que la gran mayoría de los marcadores eran ineficaces en la mayor parte de las imágenes analizadas y que el que obtenía el mejor rendimiento era un marcador referido a la cara de las personas, contenido en el fichero *haarcascade_frontalface_alt.xml*. Además y para describir los posibles falsos positivos que pueda dar este clasificador añadimos una detección de ojos dentro de las caras ya detectadas. el clasificador de los ojos lo encontramos en el fichero de nombre *haarcascade_eye.xml*. Es decir volvemos a realizar un análisis grueso para posteriormente afinarlo. Esto tiene por contrapunto que no se detectaran las personas que den la espalda a la cámara. Que si pensamos un poco sólo podrían ser reconocidas por los marcadores de las piernas o del cuerpo entero.

Total de clasificadores analizados:

- Clasificadores primarios:
    - Cara → *haarcascade_frontalface_alt.xml*
    - Ojos → *haarcascade_eye.xml*
    - Busto → *haarcascade_upperbody.xml*
    - Piernas → *haarcascade_lowerbody.xml*
    - Cuerpo Entero → *haarcascade_fullbody.xml*

- Clasificadores alternativos y secundarios

- Cara
  - Cara Frontal Alternativa → *haarcascade_frontalface_alt_tree.xml*
  - Cara Frontal Alternativa 2 → *haarcascade_frontalface_alt2.xml*
  - Cara Frontal por defecto → *haarcascade_frontalface_default.xml*
- Ojos
  - Ojo izquierdo → *haarcascade_mcs_lefteye.xml*
  - Ojo derecho → *haarcascade_mcs_lefteye.xml*
  - Ambos ojos pequeños → *haarcascade_mcs_lefteye.xml*
- Otras partes de la cara
  - Nariz → *haarcascade_mcs_nose.xml*
  - Boca → *haarcascade_mcs_mouth.xml*

Resumiendo, los clasificadores secundarios y alternativos no nos aportan prácticamente información relevante y nos basta con utilizar dos clasificadores primarios para tener una muy buena detección de personas.

Como resultado final tenemos algo que a priori parece muy simple pero que llega como conclusión de un análisis mucho más intensivo. Además debido a las limitaciones de hardware que tenemos creo que cumplimos un buen ratio de detección/consumo de recursos.

Explicación del código

Partes del código:

1. Cargar los ficheros con las características Haar (XML)
2. Tratar la imagen de entrada
3. Aplicar los clasificadores de Haar.
4. Recuadrar los resultados (opcional, en meshlium desactivado)
5. Guardado de la imagen

Cargar los ficheros XML con las característica de las caras:

```
bool gente::cargar_xml_cara()
{
      // Impresión de valores de DEBUG
      if( (estado == DEBUG) || (estado_num == 2) )
      {
        cout << "Punto: Cargar XML cara." << endl;
        logs_debug("Cargar XML cara.", 0);
      }
      else if((estado_num != 2) && (estado_num != 0))
      {
        logs_debug("Cargar XML cara.", 0);
      }

      // Cargamos el fichero con las características
      string CaraCascadeName = "haarcascades/haarcascade_frontalface_alt.xml";

      // Cargamos los ficheros para con los datos a detectar; cara y ojos
    if( !Caracascade.load( CaraCascadeName ) )
    {
        cerr << "ERROR: No se ha podido cargar el clasificador en cascada de cara. " <<
endl;
        return false;
    }

      return true;
}
```

Lo mismo para cargar el XML de los ojos:

```
bool gente::cargar_xml_ojo()
{
      // Impresión de valores de DEBUG
      if( (estado == DEBUG) || (estado_num == 2) )
      {
        cout << "Punto: Cargar XML ojo." << endl;
        logs_debug("Cargar XML ojo.", 0);
      }
      else if((estado_num != 2) && (estado_num != 0))
      {
        logs_debug("Cargar XML ojo.", 0);
      }

      // Cargamos el fichero con las características
      string OjoCascadeName = "haarcascades/haarcascade_eye.xml";

      // Cargamos los ficheros para con los datos a detectar; cara y ojos
    if( !Ojocascade.load( OjoCascadeName ) )
    {
        cerr << "ERROR: No se ha podido cargar el clasificador en cascada de ojo. " <<
endl;
        return false;
    }

      return true;
}
```

Antes de aplicar el clasificar HAAR hemos de tratar la imagen de entrada. Es necesario pasarla a escala de grises y equalizarla.

```
bool gente::tratamiento_imagen()
{
      // Impresión de valores de DEBUG
      if( (estado == DEBUG) || (estado_num == 2) )
      {
        cout << "Punto: Tratamiento de imagen." << endl;
        logs_debug("Tratamiento de imagen.", 0);
      }
      else if((estado_num != 2) && (estado_num != 0))
      {
        logs_debug("Tratamiento de imagen.", 0);
      }

      gris = cvCreateImage(cvGetSize( frame ), IPL_DEPTH_8U, 1);

      // Tratamiento de la imagen.
    cvCvtColor( frame, gris, CV_BGR2GRAY );
    cvEqualizeHist( gris, gris );

    return true;
}
```

Ahora es el momento para ejecutar el clasificador, esto se realiza con la función detectMultiScale. Esta función devuelve los objetos detectados como una lista de los rectángulos. En este caso, un vector se guarda los datos devueltos. Explicación de los parámetros de la función del clasificador de Haar:

void CascadeClassifier::**detectMultiScale**(const Mat& *imagen*, vector<Rect> &*objectos*, double *factor de escala=1.1*, int *minNeighbors=3*, int *flags=0*, Size *minSize=Size()*)

- **imagen**: Matriz del tipoCV_8U que contiene la imagen donde vamos a detectar los objetos.

- **objetos**: Vector de rectángulos que contendrá los rectángulos que rodean los objetos detectados.
- **factor de escala**: cuanto se reduce el tamaño de la imagen en cada escala de la imagen.
- **minNeighbors**: cómo muchos vecinos deben cada rectángulo candidato tiene que mantener **la bandera *CV_HAAR_SCALE_IMAGE***, que dice el algoritmo para escalar la imagen en lugar de detector.
- **minSize**: el tamaño mínimo requerido objeto posible, los objetos más pequeño se ignoran.

Este método requiere para ser increíblemente rápido que se reduzca la imagen tanto en tamaño como en cantidad de colores, como hemos visto debemos convertir la imagen de entrada en escala de grises. Luego se utiliza el detector de Haar para reconocer las caras y los ojos.

Función de detección:

```
bool gente::buscar_caras()
{
      // Variables de contador
      int i = 0;
    // Variables del bucle
    CvPoint center, center2;
    CvSize size;
    int radius, radius2;

    vector<cv::Rect> faces;
    vector<cv::Rect> ojos_sup;

    // Impresión de valores de DEBUG
      if( (estado == DEBUG) || (estado_num == 2) )
      {
        cout << "Punto: Buscar caras." << endl;
        logs_debug("BUSCAR caras.", 0);
      }
    else if((estado_num != 2) && (estado_num != 0))
      {
        logs_debug("BUSCAR caras.", 0);
      }

      // Aplicamos el clasificador Haar
    Caracascade.detectMultiScale( gris, faces, 1.1, 2, 0 | CV_HAAR_SCALE_IMAGE,
cvSize(20, 20) );

    // Bucle
    for( vector<cv::Rect>::const_iterator r = faces.begin(); r != faces.end(); r++,
i++ )
    {
        center.x = cvRound( (r->x + r->width * 0.5) * scale);
        center.y = cvRound( (r->y + r->height * 0.5) * scale);
        radius = cvRound( (r->width + r->height) * 0.25 * scale);
        size.height = r->height;
        size.width = r->width * 0.8;

        // Asignamos las coordenadas y pintamos un rectángulo
        coord_caras.push_back( cvRect(center.x - radius, center.y - radius, 2*radius,
2*radius) );

        Ojocascade.detectMultiScale( gris, ojos_sup, 1.1, 2, 0 |CV_HAAR_SCALE_IMAGE,
cvSize(30, 30) );

        for( vector<cv::Rect>::const_iterator nr = ojos_sup.begin(); nr !=
ojos_sup.end(); nr++ )
```

```
        {
            center2.x = cvRound( (r->x + nr->x + nr->width * 0.5) * scale);
            center2.y = cvRound((r->y + nr->y + nr->height * 0.5) * scale);
            radius2 = cvRound( (nr->width + nr->height) * 0.25 * scale);

            // Cogemos las coordenadas de la cara no de los ojos
            cvSetImageROI(frame, cvRect(center.x, center.y, size.width, size.height) );

              // Extraer ROI a una imagen o vector de imagenes.
              caras.push_front( frame );
              cvResetImageROI( frame );
        }
    }

    faces.clear();
    ojos_sup.clear();

    // Zona para PINTAR las caras en la imagen
      if( ((estado == DEBUG) || (estado_num == 2)) && (coord_caras.size() != 0) )
      {
          cout << "Punto: Pintar caras detectadas." << endl;
        logs_debug("Pintar caras detectadas.", 0);

        for(unsigned int i = 0; i < coord_caras.size(); i++)
        {
                cvRectangle(img, cvPoint(coord_caras[i].x, coord_caras[i].y),
cvPoint(coord_caras[i].x + coord_caras[i].width, coord_caras[i].y +
coord_caras[i].height), cvScalar(0, 0, 255, 0), 3, 8, 0);
        }

        // Guardamos la imagen final
           guardar_imagen( 1, coord_caras.size(), img );
      }

    // Imprimir número de personas detectadas en el log
      logs( ( "Número de personas detectadas: " + int2s(coord_caras.size() ) ).c_str(),
3);
      logs_debug( ( "Número de personas detectadas: " +
int2s(coord_caras.size() ) ).c_str(), 3);

      return true;
}
```

La función guardar imagen es muy sencilla y básicamente consiste en guardar la imagen que pasamos como parámetro y nombrarla según el primer número que introducimos como parámetro al método.

Anexo

Detección de matrículas

ANEXO - Detección de matrículas

En esta parte voy a comentar más en profundidad partes del código de la parte de detección de matrículas.

Para la detección de matrículas se busca un método de detección lo más generalista posible de matrículas de vehículos a motor. Se basa en la detección de matrículas a partir de sus características.

Esta parte esta divida en tres partes principales:

1. Búsqueda de matrículas
2. Homography
3. Identificación de caracteres

Búsqueda de matrículas

Esta es la parte inicial y la parte más crítica y menos definida de toda la detección de matrículas. Primero porque es la parte que se guarda con más recelo por las empresas y particulares y por otra parte porque resulta complejo diferenciar matrículas de otros objetos a partir de un análisis de fotografías., lo normal y quizá la opción a implementar en un futuro sería un sistema de aprendizaje supervisado de matrículas.

Bueno pero ciñéndonos a la explicación de nuestro método utilizado. Comentar las ideas generales, buscamos mediante diferentes métodos de detección de bordes la detección por una parte de caracteres y por otra de marcos o matrículas. Posteriormente seleccionamos los marcos que contengan un número mínimo de caracteres, eliminando los otros. Es decir buscamos blobs grandes (matrícula) que contengan otros blobs más pequeños (caracteres). A priori esto suena sencillo porque el concepto en que se basa es sencillo, lo complicado es al final obtener 1 o 2 blobs candidatos para continuar el análisis. En caso de obtener muchos blobs candidatos a ser matrícula lo único que haríamos sería mucho análisis para nada y por lo tanto una ralentización general del sistema de detección de matrículas.

Búsqueda de caracteres mediante el método de detección de borde *Laplace*.

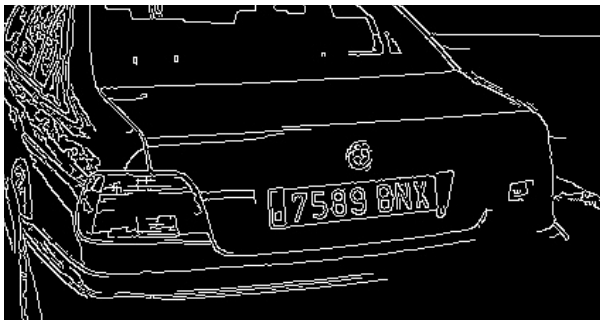
```
deque< pair<CvLabel, CvBlob*> > blobList;

laplace = cvCreateImage(cvGetSize( img ), IPL_DEPTH_16S ,1);
IplImage *laplace8 = cvCreateImage(cvGetSize( img ),IPL_DEPTH_8U ,1);
binaria = cvCreateImage(cvGetSize( img ), IPL_DEPTH_8U ,1);

cvCvtColor(img, gris, CV_RGB2GRAY);  // --> Método sencillo pero que lleva a equívocos
en la detección

cvLaplace(gris, laplace, 3);
cvConvertScale(laplace, laplace8);

IplImage* labelImg = cvCreateImage(cvGetSize( img ), IPL_DEPTH_LABEL, 1);

cvThreshold(laplace8, binaria, 100, 255, CV_THRESH_BINARY );

result = cvLabel(binaria, labelImg, blobs1);

cvFilterByArea(blobs1, 20, 3000); // Elimino los blobs cuya área este fuera del rango.

copy(blobs1.begin(), blobs1.end(), back_inserter( blobList ) );
sort(blobList.begin(), blobList.end(), cmpY);  // Ordeno los blobs de izquierda a
```

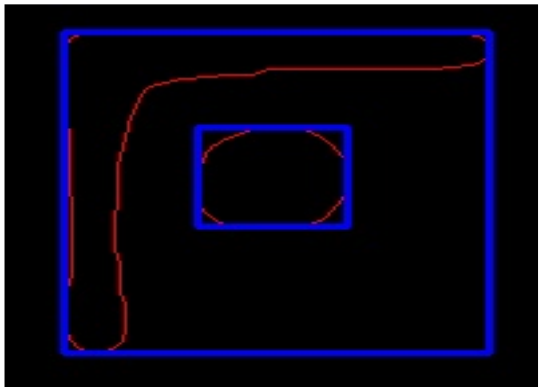
```
derecha

cout << "Número de blobs-matriculas detectados " << blobList.size() << endl;

// Tratamiento de los blobs para encontrar la matrícula
for(unsigned int i = 0; i < blobList.size(); i++)
{
      CvScalar meanColor = cvBlobMeanColor( blobList[i].second, labelImg, img);

      // Guardamos los blobs posibles caracteres para matriculas no detectadas con
laplace pero si con Canny(con este método no vemos los caracteres)
      if(rgb2grayscale( (unsigned int) meanColor.val[0], (unsigned int)
meanColor.val[1], (unsigned int) meanColor.val[2] ) < 95 )
       {
              int altura, anchura;

              altura = anchura = 0;
              // Miramos altura y anchura != 1
              altura = blobList[i].second->maxy - blobList[i].second->miny ;    //
Altura
              anchura = blobList[i].second->maxx - blobList[i].second->minx ;   //
Anchura

              if( (anchura != 1) || (altura != 1) )
                     blobList_color.push_back( blobList[i] );
       }

} // Final i

// Ordenos los blobs por su altura
sort(blobList_color.begin(), blobList_color.end(), cmpAltura);
```

Búsqueda de candidatos a matrícula mediante el método de detección de borde *Canny*.

```
// Recargar la imagen original
      img2 = cvLoadImage( i_imagen, 0);
//    cvCvtColor(frame, img2, CV_RGB2GRAY);

      binaria = cvCreateImage(cvGetSize( img2 ), IPL_DEPTH_8U ,1);

      // Usamos la canny para destacar los bordes.
      cvCanny( img2, binaria, 50, 200, 3 );

      IplImage* labelImg = cvCreateImage(cvGetSize( img ), IPL_DEPTH_LABEL, 1);

      unsigned int result;
      result = cvLabel(binaria, labelImg, blobs2);

      cvFilterByArea(blobs2, 10, 3000); // Elimino los blobs cuya área este fuera del
rango.

      copy(blobs2.begin(), blobs2.end(), back_inserter( ListaBlob ) );
      sort(ListaBlob.begin(), ListaBlob.end(), cmpY);  // Ordeno los blobs de izquierda
a derecha

       cout << "Número de blobs detectados en Canny: " << ListaBlob.size() << endl;
```


7589 BNX


7589 BNX



Como vemos al mostrarlos los rectángulos exteriores a los blobs menos como aparece una inclusión de un rectángulo dentro del otro. Es decir vemos un falso positivo. Es por esto que sólo con este método no obtendríamos una buena selección de los blobs candidatos a ser matrícula.

Ahora se aprecia la necesidad de aplicar un método más preciso, el método de análisis más fino y lento. Este método se basa es ver si hay inclusión dentro de un blob dentro de otro usando los bordes de los blobs. Este método es más lento porque se hacen más comprobaciones. Para esto he creado un método propio basado en una instrucción muy interesante de OpenCV.

cvPointPolygonTest (const CvArr contour, CvPoint2D32f pt, int measure_dist)*

Parámetros:

- *contour* – contorno de entrada
- *pt* – El punto a comparar con el contorno
- *measure_dist* – Si no es 0, mide la distancia desde el punto hasta el contorno.

La función determina si el punto está dentro de un contorno, fuera de, o se encuentra en un borde (o coincide con un vértice). Devuelve un valor positivo, negativo o cero, según corresponda. Cuando, el valor devuelto es +1, -1 y 0, respectivamente. Cuando se trata de una distancia firmado entre el punto y el borde más cercano de contorno.

Ahora vemos el método creado a tal efecto.

```
bool prueba_metodo_interior(CvBlob* b1, CvBlob* b2, bool type)
{
      bool interior = false;
      double dist = 0.0;
      int size = 0;

      bool salida = false;

      deque<CvPoint> borde_ext;
      deque<CvPoint> borde_int;

      borde_ext.clear();
      borde_int.clear();

      CvContourPolygon *polygon_ext = cvConvertChainCodesToPolygon( &b1->contour );
      copy(polygon_ext->begin(), polygon_ext->end(), back_inserter( borde_ext ) );

      CvContourPolygon *polygon_int = cvConvertChainCodesToPolygon( &b2->contour );
      copy(polygon_int->begin(), polygon_int->end(), back_inserter( borde_int ) );

      CvMemStorage* storage = cvCreateMemStorage(0);
      CvSeq* seq = cvCreateSeq( CV_32SC2, /* sequence of integer elements */
                                sizeof(CvSeq), /* header size - no extra fields */
                                sizeof(CvPoint), /* element size */
                                storage /* the container storage */ );

      for(unsigned int i = 0; i < borde_ext.size(); i++ )
      {
        cvSeqPush( seq, &borde_ext[i] );
      }

      // Tama–o del contorno del blob interior
      size = borde_int.size();
```

```
      // Si type = 'true' an‡lisis usando 10 puntos aleatorios de las 4 zonas del borde,
si type = false analizo todos los puntos del borde.
      if( type )
      {
        vector<int> random;

        size = size / 4;

        srand (time (NULL));
        int contador = 1;
        int i = myRandom ( size );

        while (contador < 11)
        {
                random.push_back( i );
                i = myRandom (-1);
                contador++;
        }
        for(int j = 0; j < 4; j++)
        {
                for(unsigned int i = 0; i < random.size(); i++ )
                {
                        dist = cvPointPolygonTest(seq, cvPoint2D32f(borde_int[ (j*size) +
i ].x, borde_int[ (j*size) + i ].y) , false);
                        if( dist == -100 )
                        {
                                salida = true;
                                break;
                        }
                }
                if( salida )
                        break;
        }
      }
      else
      {
        for(unsigned int i = 0; i < borde_int.size(); i++ )
        {
                dist = cvPointPolygonTest(seq, cvPoint2D32f(borde_int[i].x,
borde_int[i].y) , false);
                if( dist == -100 )
                {
                        salida = true;
                        break;
                }
        }
      }

      /* release memory storage in the end */
      cvReleaseMemStorage( &storage );

      delete polygon_ext;
      delete polygon_int;

      return !salida;
}
```

A partir de aquí sólo seleccionamos como candidatos a ser matrículas aquellos blobs que contienen en su interior 2 o más blobs. Posteriormente eliminamos de forma ordenada los blobs que no han cumplido la condición anterior. Esto es sencillo pero hay que tener en cuenta una peculiaridad. Al trabajar con un 'deque' o con un 'vector', definidos en las librerías estándar de C/C++, hay que saber que al eliminar un elemento que ocupa el índice 'j' implica que todos los elementos cuyo índice son mayores a 'j' se corren una posición. Esto es relevante porque sino tenemos en cuenta esto podemos estar eliminando elementos que no queremos.

```
// Ordenamos de mayor a menor. Para una correcta eliiminación posteriormente
sort(eliminar.begin(), eliminar.end(), mayormenor);
```

```
for(unsigned int i = 0; i < eliminar.size(); i++)
{
      ListaBlob.erase( ListaBlob.begin() + eliminar[i] );
}

eliminar.clear();
```

Aparte de este método hay unos cuantos métodos para mostrar y pintar los blobs detectados en diferentes imágenes intermedias. Pero sólo están disponibles si hemos activado el estado de 'debug'. Tenemos la opción de guardar estas imágenes intermedias en la carpeta de debug o de visualizarlas por pantalla. Para visualizar las imágenes intermedias por pantalla es necesario haber compilado la librería OpenCV con Gtk+, Windows o Carbon (librerías gráficas correspondientes a los principales Sistemas Operativos).

Por último y antes de pasar al siguiente paso es importante liberar las variables y vaciar los vectores para liberar la mayor cantidad de memoria posible. Trabajamos con un hardware muy imitado y es necesario ser limpio y ordenado en el uso de la memoria.

```
bool matricula::liberar_parte_busqueda()
{
      // Impresión de valores de DEBUG
      if( (estado == DEBUG) || (estado_num == 2) )
      {
        cout << "Punto LIBERAR PARTE DE BUSQUEDA." << endl;
        logs_debug("LIBERAR PARTE DE BUSQUEDA.", 0);
      }
      else if(estado_num != 2)
      {
        logs_debug("LIBERAR PARTE DE BUSQUEDA.", 0);
      }

      // Liberamos variables.
      //--------------------------
      cvReleaseImage( &gris );

      cvReleaseBlobs( blobs1 );
      cvReleaseBlobs( blobs2 );

      blobList_mat.clear();
      blobList_char.clear();

      blobList_color.clear();
      ListaBlob.clear();

      eliminar.clear();

      if( (estado == DEBUG) || (estado_num == 2) )
      {
        cout << "Fin LIMPIEZA." << endl;
        logs_debug("Fin LIMPIEZA.", 0);
      }
      else if(estado_num != 2)
      {
        logs_debug("Fin LIMPIEZA.", 0);
      }

    return true;
}
```

Homography

Explicación del concepto, no es posible una traducción al castellano porque el significado de homografía en castellano es únicamente el de igual grafía y dista mucho del concepto que queremos explicar aquí.

Homografía es un concepto matemático del área de la geometría. Una "homografía", si podemos traducirlo así, es una transformación invertible de un espacio de proyección (por ejemplo, el plano proyectivo real) igual que los mapas de líneas rectas con líneas rectas. Los sinónimos son co-alineación, transformación proyectiva y proyectividad.

Formalmente, una transformación de proyección en un plano es una transformación utilizada en la geometría proyectiva: es la composición de un par de proyecciones de perspectiva. En él se describe lo que ocurre con la percepción de las posiciones de los objetos observados en el punto de vista de los cambios de observador. Las transformaciones proyectivas no preservan los tamaños y ángulos, pero si conservan la incidencia y el ratio de cruce: dos propiedades que son importantes en la geometría proyectiva.

Veamos un ejemplo para comprender rápidamente la acción que realiza la operación de Homography:

Ilustración 5 - Imagen original

Ilustración 6 - Imagen tras aplicar Homography

Para aplicar esta operación debemos seleccionar las 4 esquinas del objeto que se encuentra proyectado y queremos transformar. Primero se realizan las pruebas introduciendo de manera manual las esquinas de las imágenes que quiero transformar hasta que se afina el método.

En la imagen de arriba los 4 puntos que corresponden a las 4 esquinas de la matrícula son los siguientes puntos:

X	Y
65	340
549	401
518	172
195	136

Ilustración 7 - Imagen original con las esquinas marcadas

Matriz P, puntos de origen. Las columnas de la matriz son las coordenadas *x* e *y* de las 4 esquinas seleccionadas. Por eso tenemos una matriz 2x4

$$P = \begin{pmatrix} 65 & 549 & 518 & 195 \\ 340 & 401 & 172 & 136 \end{pmatrix}$$

Matriz H, puntos de destino:

$$H = \begin{pmatrix} 0 & 640 & 640 & 0 \\ 480 & 480 & 0 & 0 \end{pmatrix}$$

Ahora aplico la instrucción para encontrar la matriz de transformación.

cvFindHomography(&p, &h, homo);

Obteniendo como resultado

$$\text{homo} = \begin{pmatrix} 2'612 & 1'664 & -735'632 \\ -0'506 & 4'541 & -518'881 \\ 0'000 & 0'003 & 1'000 \end{pmatrix}$$

Ahora aplicamos la matriz de transformación a la imagen original

cvWarpPerspective(im, out, &homo);

Las variables de entrada de esta instrucción son sencillas de identificar:

1. im: imagen original (IplImage*)
2. out: Variable de imagen de sálida donde se guardará la imagen resultante.

3. Homo: Matriz de transformación.

Finalmente redimensionamos la imagen resultante para tener la imagen final del mismo tamaño que la imagen original.

Además de esto hay implementado un método para ordenar los puntos de las esquinas por si fueran introducidos en otro orden al establecido. Y también un método para mostrar las matrices correctamente en la pantalla del terminal que además escribe las matrices en los ficheros de debug de log por si hubiera que algún error.

No obstante voy a poner el código del método con el que cálculo la matriz de transformación.

```
void matricula::calc( CvMat* p2h, int real_width, int real_height, int width, int
height)
{
      int x, y;
    CvMat p;
    CvMat h;

    double dos_cuatro_nueva[] = { 0, real_width, real_width, 0,
                                  real_height, real_height, 0, 0 };

    x = y = 0;

    // Impresión de valores de DEBUG
      if( (estado == DEBUG) || (estado_num == 2) )
      {
          cout << "Dentro de 'calc'" << endl;
          logs_debug("Etapa de 'calc'.", 0);
      }
    else if(estado_num != 2)
      {
        logs_debug("Etapa de 'calc'.", 0);
      }

    order_points(width, height);

    cvInitMatHeader( &p, 2, 4, CV_64FC1, dos_cuatro);
    cvInitMatHeader( &h, 2, 4, CV_64FC1, dos_cuatro_nueva);

    if( (estado == DEBUG) || (estado_num == 2) )
      {
          cout << "Imprimimos matriz P inicializada: " << endl;
          logs_debug("Imprimimos matriz P inicializada: ", 6);
          PrintMat( &p );

          cout << "Imprimimos matriz H inicializada: " << endl;
          logs_debug("Imprimimos matriz H inicializada: ", 6);
          PrintMat( &h );
      }

      // Matriz con los puntos del objeto a modificar
    for (int i = 0; i < 4; i++)
    {
        cvSet2D( &p, 0, i, cvRealScalar( puntos[i].x) );
        cvSet2D( &p, 1, i, cvRealScalar( puntos[i].y) );
    }

     // Impresión de valores de DEBUG
      if( (estado == DEBUG) || (estado_num == 2) )
      {
        cout << "Imprimimos matriz P modificadas v1: " << endl;
        logs_debug("Imprimimos matriz P modificadas v1: ", 6);
          PrintMat( &p );
          cout << "Imprimimos matriz H modificadas: " << endl;
          logs_debug("Imprimimos matriz H modificadas: ", 6);
          PrintMat( &h );
```

```
          cout << "Imprimimos matriz P modificadas v2: " << endl;
          logs_debug("Imprimimos matriz P modificadas v2: ", 6);
          PrintMat( &p );
      }

    cvFindHomography(&p, &h, p2h);

    if( (estado == DEBUG) || (estado_num == 2) )
      {
          cout << "Imprimimos matriz Homo modificadas: " << endl;
          PrintMat( p2h );
      }
}
```

Por último comentar un método muy interesante encontrado y que se ha modificado para visualizar en el terminal y escribir en un fichero de manera correcta una matriz.

```
void PrintMat(CvMat* A)
{
    // Aun hay que meterle mucho trabajo porque no se guardan bien los datos.
    int i,j;
    float valor;
    char buffer[100];
    string cadena;

    //printf("\nMatrix = :");
    for(i=0;i<A->rows;i++)
    {
        printf("\n");

        switch( CV_MAT_DEPTH(A->type) )
        {
              case CV_32F:
              case CV_64F:
                  for(j=0; j<A->cols; j++)
                  {
                      valor = (float) cvGetReal2D( A, i, j );
                      printf("%9.3f ", valor);
                      sprintf(buffer, "%9.3f ", valor);
                      cadena += string(buffer);
                  }
                  logs_debug( cadena, 6);
                   cadena = "";
                  break;
              case CV_8U:
              case CV_16U:
                  for(j=0; j<A->cols; j++)
                  {
                      printf("%6d ",(int)cvGetReal2D( A, i, j ));
                      cadena += int2s( (int) cvGetReal2D( A, i, j ) );
                  }
                  logs_debug( cadena, 6);
                   cadena = "";
                  break;
              default:
                  break;
        }
    }
    printf("\n");
    logs_debug( cadena, 6);
}
```

Por último y como ya se ha comentado en el bloque anterior al finalizar la operación de Homography libero toda la memoria que puedo vaciando los vectores y liberando las variables de imagen.

Identificación de caracteres

Esta es la última fase y la que mostrará el resultado final de nuestro análisis para la detección de matrículas.

Pasos en la identificación de caracteres:

- Primero debemos extraer los caracteres de la matrícula.

  Pasos a seguir para extraer los caracteres de la imagen. Vamos a distinguir los caracteres porque contrastan mucho con el fondo de la matrícula y porque normalmente suelen ser de color negro o al menos de un tono oscuro.

  Se binariza la imagen de forma dinámica, es decir sin fijar un umbral dado que puede cambiar de una imagen a otra. Para eso uso el método de *Binarización Otsu*.

```
tratamiento = binarizacion_otsu( matriculas[ num_matricula ] );
```

  Posteriormente extraemos la matrícula eliminado los posibles blob exteriores a ella. Para ello busco los blobs y me quedo con el más grande, que será la matrícula.

```
      IplImage *labelImg = cvCreateImage( cvGetSize( tratamiento ), IPL_DEPTH_LABEL, 1);
      unsigned int result = cvLabel(tratamiento, labelImg, extraccion);
      // Nos quedamos con el blob más grande
      cvFilterByLabel(extraccion, cvGreaterBlob(extraccion));

      CvBlobs::const_iterator it = extraccion.begin();
      int ancho, alto;
      ancho = it->second->maxx - it->second->minx;
      alto = it->second->maxy - it->second->miny;

      // Definimos la Región de Interes.
      cvSetImageROI(tratamiento, cvRect(it->second->minx, it->second->miny, ancho,
alto));

      /* crear imagen de destino
         Note that cvGetSize will return the width and the height of ROI */
      IplImage *final = cvCreateImage(cvGetSize(tratamiento), tratamiento->depth,
tratamiento->nChannels);

      /* copiar subimage */
      cvCopy(tratamiento, final, NULL);
```

  Una vez seleccionada la matrícula objetivo extraigo todos los blobs que contiene. Posteriormente los ordeno de menor a mayor ordenada x del centro de los blobs.

```
      cvNot( final,  final);

      labelImg = cvCreateImage(cvGetSize( final ), IPL_DEPTH_LABEL, 1);
      result = cvLabel(final, labelImg, blobs_caracteres);

      cvFilterByArea(blobs_caracteres, 1000, 30000);

      // Guardo los blos en un vector
      copy(blobs_caracteres.begin(), blobs_caracteres.end(),
back_inserter(blobList_caracteres));
      sort(blobList_caracteres.begin(), blobList_caracteres.end(), cmpX);
      cvReleaseBlobs( blobs_caracteres );
```

Se realizan comprobaciones por si no hemos detectado ningún blob y en caso de querer más pruebas guardo imágenes con los resultados intermedios. Posteriormente busco una secuencia de blobs que tenga una altura muy similar. Con esto elimino todos los blobs detectados que no sean caracteres. Trabajo con vectores y tras la selección de blobs marcados como caracteres elimino los que son clasificados como no caracteres.

```
// Proceso de Selección
      int altura;
      int contador = 0;

      deque<int> eliminacion;
      deque<int> altura_vector;

      eliminacion.clear();

      // Guardar caracteristicas de los BLOGS
      if( (estado == DEBUG) || (estado_num == 2) )
      {
        cout << "Punto: SELECCION CARACTERES MaTrIcUlA." << endl;
        logs_debug("SELECCION CARACTERES MaTrIcUlA.", 0);
      }
      else if(estado != DEBUG)
      {
        logs_debug("SELECCION CARACTERES MaTrIcUlA.", 0);
      }

      // Análisis de las características de los blobs
      //-----------------------------------------------------
      for(unsigned int i = 0; i < blobList_caracteres.size(); i++)
      {
        altura = blobList_caracteres[i].second->maxy - blobList_caracteres[i].second-
>miny;

        altura_vector.push_back( altura );
      }

      // Mostrar las diferentes alturas
      if( (estado == DEBUG) || (estado_num == 2) )
      {
        for(unsigned int i = 0; i < blobList_caracteres.size(); i++)
        {
                cout << " Altura blob #" << i << " -> " << altura_vector[i] << endl;
                logs_debug( (" Altura blob #" + int2s(i) + " -> " +
int2s(altura_vector[i]) ).c_str(), 0);
        }

        cout << endl;
      }
      else if(estado != DEBUG)
      {
        for(unsigned int i = 0; i < blobList_caracteres.size(); i++)
        {
                logs_debug( (" Altura blob #" + int2s(i) + " -> " +
int2s(altura_vector[i]) ).c_str(), 0);
        }
      }

      // Elimino lo blos que no corresponden a caracteres.
      for(unsigned int i = 0; i < blobList_caracteres.size(); i++)
      {
        for(unsigned int j = 0; j < blobList_caracteres.size(); j++)
        {
                if( i == j )
                        continue;
                else
                {
                        if( aprox( altura_vector[i], altura_vector[j], 0.10 ) )
                                contador++;
                }
```

```
        }

        if( contador < 4 )
                eliminacion.push_back( i );

        contador = 0;
      }

      sort (eliminacion.begin(), eliminacion.end(), mayormenor);

      // Algoritmo de eliminación de los blobs no-caracteres
      for(unsigned int r = 0; r <  eliminacion.size() ; r++)
      {
        blobList_caracteres.erase( blobList_caracteres.begin() + eliminacion[r] );
      }
```

```
                        14, 286, 90, 398, // J
                        94, 286, 165, 397, // K
                        166, 287, 240, 398, // L
                        236, 287, 313, 397, // M
                        314, 286, 387, 398, // N
                        388, 287, 456, 396, // O
                        455, 287, 529, 395, // P
                        532, 289, 606, 402, // Q
                        603, 288, 684, 398, // R
                        47, 430, 122, 537, // S
                        119, 424, 191, 536, // T
                        195, 423, 271, 536, // U
                        267, 426, 339, 535, // V
                        340, 422, 412, 538, // W
                        413, 427, 489, 538, // X
                        484, 423, 557, 535, // Y
                        562, 428, 645, 538}; // Z
```

Redimensionado de la imagen modelo para tener ambas imágenes del mismo tamaño para una mejor aplicación del método de identificación. Además añadimos una pequeña operación de erosión a la imagen con el carácter objetivo para tratar de eliminar posibles imperfecciones. Usamos una máscara en forma de cruz para hacer esta erosión.

```
bool matricula::ajustar_imagen()
{
      IplImage *patron, *simbolo, *modelo;
      IplImage *img_b;
      IplImage *p_b;

      // Erode kernel variables
      int km[] = {0, 1, 0, 1, 1, 1, 0, 1, 0};
      IplConvKernel* kernel = 0;
      kernel = cvCreateStructuringElementEx( 3, 3, 0, 0, CV_SHAPE_RECT, km );

      img_b = cvCreateImage(cvGetSize(imgsec), IPL_DEPTH_8U, 1);
      p_b = cvCreateImage(cvGetSize( img2 ), IPL_DEPTH_8U, 1);

      // ------------------------------------------------
      // Suavizado y conversion a imágenes binarias
      // ------------------------------------------------
      cout << "Carac. prof: " << imgsec->depth << " canales " << imgsec->nChannels << "
step " << imgsec->widthStep << endl;
      cvSmooth(imgsec, imgsec, CV_GAUSSIAN, 3, 0, 0, 0);

      cvCvtColor(imgsec, img_b, CV_RGB2GRAY);
      cvCvtColor(img2, p_b, CV_RGB2GRAY);

      cvThreshold(img_b, img_b, 155, 255, CV_THRESH_BINARY);
      cvThreshold(p_b, p_b, 200, 255, CV_THRESH_BINARY);

      cvNot(img_b, img_b);
      cvNot(p_b, p_b);

      cvErode(img_b, img_b, kernel, 1);

      // ------------------------------------------------
      // Variables de los blobs
      // ------------------------------------------------
      CvBlobs blobs_patron;
      CvBlobs blobs_caracter;
      unsigned int result, result2;
      IplImage *labelImg = cvCreateImage(cvGetSize(imgsec), IPL_DEPTH_LABEL, 1);
      CvSize tamano;

      // Deteccion de los Blobs
      result2 = cvLabel(img_b, labelImg, blobs_caracter);
      labelImg = cvCreateImage(cvGetSize( p_b ), IPL_DEPTH_LABEL, 1);
      result = cvLabel(p_b, labelImg, blobs_patron);
```

```
	// Acceso directo a los elementos del blob
	CvBlobs::const_iterator it = blobs_caracter.begin();
	CvBlob *blob_c = it->second;
	CvBlobs::const_iterator jt = blobs_patron.begin();
	CvBlob *blob_p = jt->second;

	cvNot(img_b, img_b);
	cvNot(p_b, p_b);

	tamano.width = blob_c->maxx - blob_c->minx;
	tamano.height = blob_c->maxy - blob_c->miny;
	simbolo = cvCreateImage(tamano, IPL_DEPTH_8U, 1);

	cvSetImageROI(img_b, cvRect(blob_c->minx, blob_c->miny, tamano.width,
tamano.height));

	/* Copiar subimagen */
	cvCopy(img_b, simbolo, NULL);
	cvResetImageROI(img_b);

	modelo = cvCreateImage(cvGetSize(simbolo), IPL_DEPTH_8U, 1);

	tamano.width = blob_p->maxx - blob_p->minx;
	tamano.height = blob_p->maxy - blob_p->miny;
	patron = cvCreateImage(tamano, IPL_DEPTH_8U, 1);

	cvSetImageROI(p_b, cvRect(blob_p->minx, blob_p->miny, tamano.width,
tamano.height));

	/* Copiar subimagen */
	cvCopy(p_b, patron, NULL);
	cvResetImageROI(p_b);

	// Redimension
	cvResize(patron, modelo);

	imagen_cb = cvCloneImage( simbolo );
	imagen_pb = cvCloneImage( modelo );

	// liberar memoria
	cvReleaseImage( &img_b );
	cvReleaseBlobs( blobs_patron );
	// faltan de liberar muchas más imágenes.

	return true;
}
```

La parte crucial de la identificación se basa en la comparativa de patrones a partir de la correlación normalizada entre imágenes. Para realizar esta operación utilizamos el siguiente método de OpenCV:

- cvMatchTemplate(const CvArr image, const CvArr* templ, CvArr* result, int method)*

- *image* – Imagen que queremos comparar.
- *templ* – Imagen con el patrón para comparar.
- *result* – variable para contener el resultado; Variable decimal de 32bits.
- *method* – Especifica el método a aplicar en la comparación. Los métodos definidos son los siguientes:
  - method = CV_TM_SQDIFF *

$$R(x,y) = \sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2$$

  - method = CV_TM_SQDIFF_NORMED

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

- method = CV_TM_CCORR

$$R(x,y) = \sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))$$

- method = CV_TM_CCORR_NORMED

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

- method = CV_TM_CCOEFF

$$R(x,y) = \sum_{x',y'}(T'(x',y') \cdot I(x+x',y+y'))$$

donde

$$T'(x',y') = T(x',y') - \frac{1}{(w \cdot h)} \sum_{x'',y''} T(x'',y'')$$

$$I'(x+x',y+y') = I(x+x',y+y') - \frac{1}{(w \cdot h)} \sum_{x'',y''} I(x+x'',y+y'')$$

- method = CV_TM_CCOEFF_NORMED

$$R(x,y) = \frac{\sum_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}}$$

Realizamos la comparativa con los 26 caracteres modelo y definimos el carácter detectado como aquel que ha obtenido el valor más alto en la comparativa. Para guardar los resultados

Bucle donde se realiza la identificación de los caracteres.

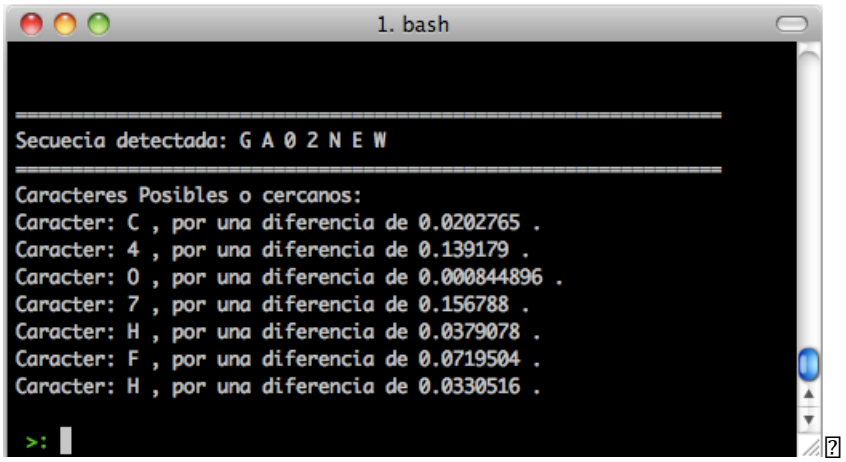
```
bool matricula::bucle_comparar()
{
      // Variables internas del algoritmo
      // ---------------------------------
      int i, j; // i->Secuencia del patrón, j->Caracteres en la matrícula
      int num ;
      int k, k_store; // Para guardar la componente y componente del máximo
      double max_value;
      // Definición de variables locales al bucle
      int iwidth ;
      int iheight ;
      IplImage *temp;
      IplImage *img2_re;  // Para los casos excepcionales de letras que no detecto con
el ajuste
```

```
      CvPoint minloc, maxloc;
      double minval, maxval;
      double diff;

      struct probabilidad datos;

      // Utilizar esta variable para testear que todo dentro del bucle vaya perfecto
      bool estado_comparativa = false;

      // Inicializo variables
      //-------------------------
      i = j = 0;
      num = 1;
      k = k_store = 0;
      max_value = 0.0;
      datos.inicializar();
      diff = 0.0;

      // Impresión de valores de DEBUG
      if( (estado == DEBUG) || (estado_num == 2) )
      {
        cout << "Punto: Inicio de BUCLE PRINCIPAL." << endl;
        logs_debug("Inicio de BUCLE PRINCIPAL.", 0);
      }
      else if( (estado_num != 2) && (estado_num != 0) )
      {
        logs_debug("Inicio de BUCLE PRINCIPAL.", 0);
      }

      // Bucle principal
      //-----------------------
      while(1)
      {
        cargar_caracter_imagen(j);
        cargar_caracter_patron(i);

        if( !estado_comparativa )
        {
              ajustar_imagen();
        }
        else
        {
              // Kernel variables
              int km[] = {0, 1, 0, 1, 1, 1, 0, 1, 0};
              IplConvKernel* kernel = 0;
              kernel = cvCreateStructuringElementEx( 3, 3, 0, 0, CV_SHAPE_RECT, km );

              //-----------------------------------------
              // Redimension
              //-----------------------------------------
              img2_re = cvCreateImage(cvGetSize(imgsec), imgsec->depth, imgsec-
>nChannels);
              imagen_cb = cvCreateImage(cvGetSize(imgsec), IPL_DEPTH_8U, 1);
              imagen_pb = cvCreateImage(cvGetSize(imgsec), IPL_DEPTH_8U, 1);
              cvResize(img2, img2_re);
              //-----------------------------------------
              // Tratamiento imagen & Binarizacion
              //-----------------------------------------
              cvSmooth(imgsec, imgsec, CV_GAUSSIAN, 3, 0, 0, 0);

              cvCvtColor(imgsec, imagen_cb, CV_RGB2GRAY);
              cvCvtColor(img2_re, imagen_pb, CV_RGB2GRAY);

              cvSmooth(imagen_cb, imagen_cb, CV_GAUSSIAN, 3, 0, 0, 0);

              cvThreshold(imagen_cb, imagen_cb, 150, 255, CV_THRESH_BINARY);
              cvThreshold(imagen_pb, imagen_pb, 200, 255, CV_THRESH_BINARY);
              cvNot(imagen_cb, imagen_cb);
              cvErode(imagen_cb, imagen_cb, kernel, 1);
              cvNot(imagen_cb, imagen_cb);

              cvReleaseImage(&img2_re);
              cvReleaseStructuringElement( &kernel);
        }
```

```
        //-----------------------------------------
        // Comparativa
        //-----------------------------------------
        iwidth = imagen_pb->width - imagen_cb->width + 1;
        iheight = imagen_pb->height - imagen_cb->height + 1;
        temp = cvCreateImage(cvSize( iwidth, iheight ), 32, 1);
        cvMatchTemplate(imagen_cb, imagen_pb, temp, CV_TM_CCORR_NORMED);  //
CV_TM_SQDIFF or CV_TM_CCORR_NORMED, both with _NORMED versions
        cvMinMaxLoc(temp, &minval, &maxval, &minloc, &maxloc, 0);

        // Ploteo de los resultados
        //-------------------------
        if(minval > 1)
        {
                // Impresión de valores de DEBUG
                if( (estado == DEBUG) || (estado_num == 2) )
                {
                        cout << "ENCONTRADO !!" << endl;
                        cout << caracter[k] << endl << ". Valor min: " << minval << ".
Valor max: " << maxval <<  endl;
                        // Guardado de las imagenes
                        guardar_imagen(1, num, imgsec);
                }
                j = j + 4;
                i = k = 0;
                num++;
                // Introducimos el elemento detectado en el resultado
                resultado.push_back( caracter[k] );
                resultado_int.push_back( k );
                // Liberamos memoria
                cvReleaseImage(&imgsec);
                cvReleaseImage(&img2);
                cvReleaseImage(&temp);
                if( num == num_caracteres_imagen ) // Si hemos anadlizado todos los
caracteres de la imagen.
                        break;
        }
        else
        {
                // Impresión de valores de DEBUG
                if( (estado == DEBUG) || (estado_num == 2) )
                {
                        cout << "No hay semejanza: " << caracter[k] << ". Valor min: " <<
minval << ". Valor max: " << maxval<< endl;
                }

                if( minval > max_value )
                {
                        datos.k_store_second = k_store;
                        k_store = k;
                        datos.maxvalue_second = max_value;
                        max_value = minval;
                        datos.diferencia = max_value - datos.maxvalue_second;
                        if( (estado == DEBUG) || (estado_num == 2) )
                        {
                                guardar_imagen(3, num, imagen_pb);
                        }
                }
                else
                {
                        // Guardo el valor coincidente si la distancia es menor que la
guardada.
                        if( datos.diferencia > ( max_value - maxval ) )
                        {
                                datos.k_store_second = k;
                                datos.maxvalue_second = maxval;
                                datos.diferencia = max_value - maxval;
                                if( (estado == DEBUG) || (estado_num == 2) )
                                {
                                        guardar_imagen(4, num, imagen_pb);
                                }
                        }
                }
                // Actualizamos variables
                i = i + 4;
```

```
                k++;
                if(i == 144) // Comprobamos si ya hemos comparado con todos los
caracteres del patrón
                {
                        i = 0;
                        k = 0;

                        if( ( k_store == 0 ) || ( k_store == 18 ) || ( k_store == 23 ) ||
( k_store == 24 ) ) // 0, H, M, N y W
                        {
                                if( estado_comparativa == false      )
                                {
                                        // Impresión de valores de DEBUG
                                        if( (estado == DEBUG) || (estado_num == 2) )
                                        {
                                                cout << "Detectado uno de los caracteres
ESPECIALES: 0, H, M, N o W " << endl << endl;
                                        }
                                        estado_comparativa = true;
                                        continue;
                                }
                                if( estado_comparativa == true)
                                {
                                        estado_comparativa = false;
                                }
                        }

                        // Condición especial para el CERO y la D
                        if( k_store == 13)
                        {
                                // Impresión de valores de DEBUG
                                if( (estado == DEBUG) || (estado_num == 2) )
                                {
                                        cout << " Caracter D  DETECTADO !!!! " << endl;
                                }
                                if(     datos.diferencia < 0.001 )
                                        k_store = 0;
                        }

                        // Impresión de valores de DEBUG
                        if( (estado == DEBUG) || (estado_num == 2) )
                        {
                                cout << "====================================" << endl;
                                cout << "CARACTER NO DETECTADO" << endl;
                                cout << "El caracter más parecido es: " <<
caracter[k_store] << " Valor: " << max_value << endl;
                                cout << "====================================" << endl
<< endl;
                                cout << "Valor de num: " << num << " y valor del número de
caracteres: " << num_caracteres_imagen << endl;
                                cout << "------------------------------------" << endl;
                                // Guardado de los carácteres de la imagen

                                guardar_imagen(1, num, imgsec);
                                guardar_imagen(5, num, imagen_cb);
                        }
                        resultado.push_back( caracter[k_store] );
                        resultado_int.push_back( k_store );

                        datos.caracter = caracter[datos.k_store_second];
                        prob.push_back( datos );
                        k_store = 0;
                        max_value = 0.0;
                        datos.inicializar();

                        // Actualizaicón de variables del bucle
                        j = j + 4;
                        // Comprobar que no hemos comprobado
                        if( num == num_caracteres_imagen )
                                break;

                        num++;
                }

                // Liberar memoria imágenes
```

```
                cvReleaseImage(&temp);
        } // Final Comparativa MAX y MIN del cvMatchTemplate

      } // Fin BUCLE PRINCIPAL

      // Liberamos memoria
      cvReleaseImage( &imgsec );
      cvReleaseImage( &img2 );
      cvReleaseImage( &imagen_cb );
      cvReleaseImage( &imagen_pb );

      return true;
}
```



Anexo

Logs e Imágenes

ANEXO - LOGS e IMAGENES

Este anexo trata de explicar el funcionamiento de un conjunto de método creados para guardar de forma ordenada unos mensajes de texto. Se vana guardar dos tipos de logs:

- Logs finales: mensajes con información útil para el usuario final. Pocos en cuanto a cantidad pero deben aportar información final de relevancia.
- Logs debug: mensajes con información útil para saber en caso de error o fallo en que parte del programa se ha producido este error. Información no muy importante para el cliente final pero si para los creadores/mantenedores de este programa.

Características internas de los logs:

- En todos los los logs se escribe la hora con hh:mm:ss
- Los logs se encuentran ordenados temporalmente de los más recientes (parte superior) a los más antiguos en la parte inferior.
- Posteriormente entre corchetes la temática del log
    - Para los logs finales:
        - #1 [ERROR]
        - #2 [MENSAJE]
        - #3 [INCIDENCIA]
        - #4 [ESTADO]
        - #5 [DEBUG]
        - #6 [OTRO]
    - Para los logs de debug:
        - #0 [ETAPA]
        - #1 [ERROR]
        - #2 [MENSAJE]
        - #3 [INCIDENCIA]
        - #4 [ESTADO]
        - #5 [DEBUG]
        - #6 [OTRO]
        - #8
        - #9 [ESTADISTICA]
- A continuación el grueso del mensaje de log.

Características externas de los logs:

- Los ficheros se crean con el nombre de la fecha completa del día y la extensión *.LG*.
- Los ficheros de log están escritos en texto plano y se pueden abrir con cualquier programa que abra ficheros de texto.
- Asimismo los ficheros de logs se encuentran guardados en la carpeta LOGS/AÑO/MES/DIA/ desde la raíz desde la que se ejecuta el programa que va a hacer uso del sistema de logs.
- Además los logs de debug se encuentran dentro de la carpeta 'debug' en la misma ruta antes mencionada junto con algunas imágenes que se guardan a modo de comprobación.

Comentar que de cara al exterior no están declarados todos los métodos definidos en el fichero 'CPP'. Básicamente tenemos dos tipos de funciones declaradas:

1. Los 2 métodos de escritura de logs
    i. void logs (string , int);
    ii. void logs_debug (string , int);
2. Métodos para obtener la fecha/hora y la ruta completa para poder guardar imágenes de muestra.
    i. string fecha_fichero2();
    ii. string dar_hora2();
    iii. int dar_minutos();
    iv. int dar_segundos();

v. string ruta_fichero_debug();
vi. string ruta_fichero_debug_caracteres();

Ejemplo de ficheros de logs

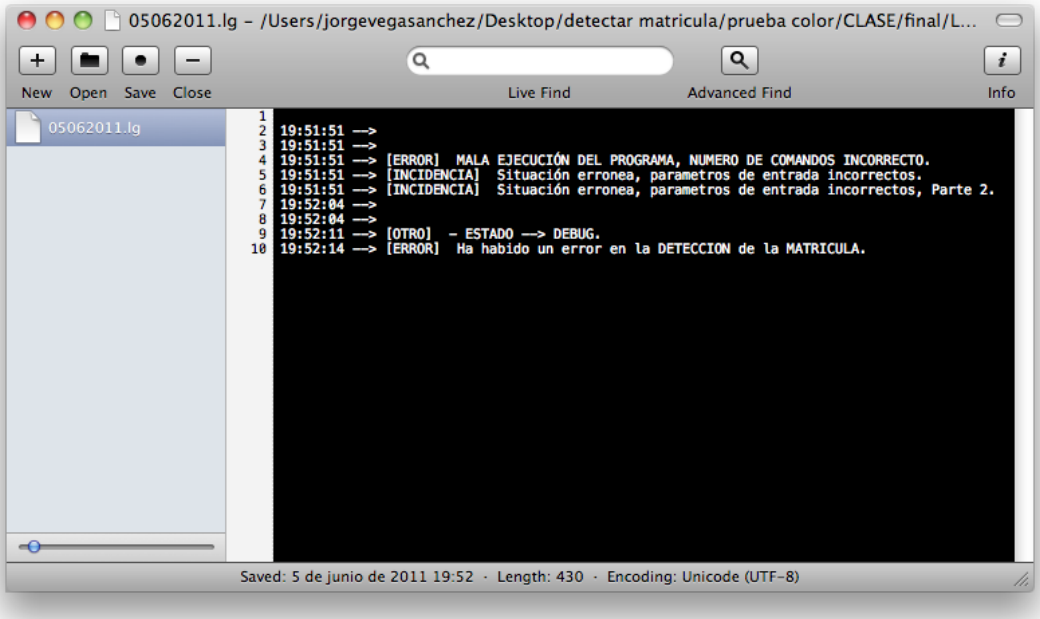


Ilustración 1 - Ejemplo fichero de log

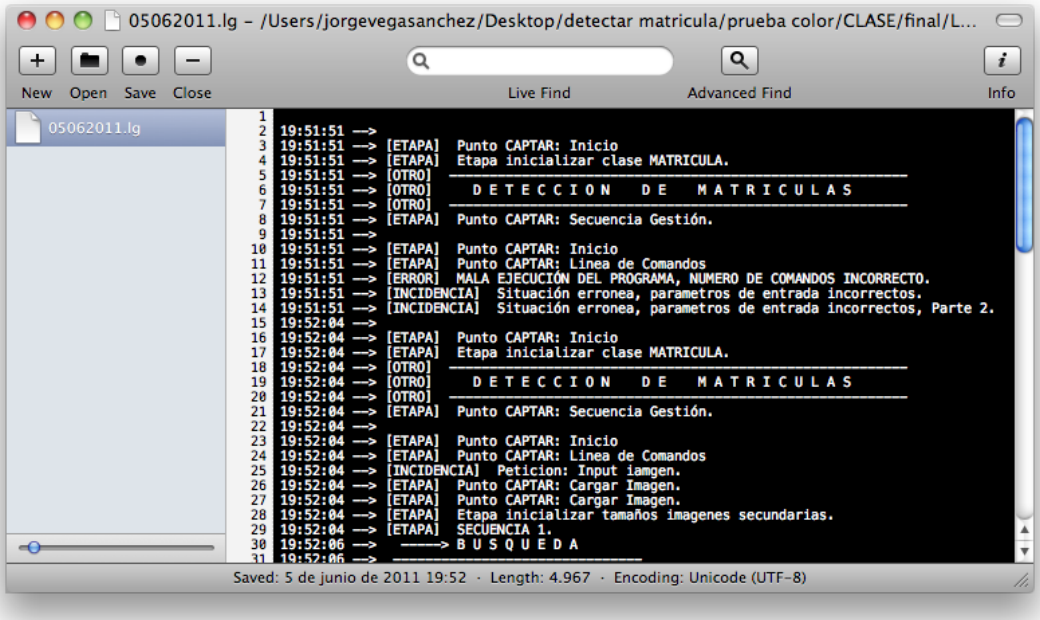


Ilustración 2 - EJemplo fichero de log de debug

Una muestra muy pequeña del código fuente:

```
// Metodo principal para la ejecuci—n del m©todo
void logs(string cadena, int tipo)
{
	bool marcador = false;
	/* Pasos a seguir para los logs del sistema:
	1.- Primero saber el nombre del programa que se esta ejecutando
	2.- Segundo, crear el fichero donde se escribiran los logs de ese programa
	3.- Tercero, escribir los logs en el fichero. */
	// string nombre=busqueda_programa();
	marcador = comprobar_carpetas();
	if( !marcador )
	{
		crear_directorios();
		crear_fichero_log();
	}

	escribir_log(tipo, cadena);
}
```

```
// Metodo principal para la ejecuci—n del m©todo
void logs_debug(string cadena, int tipo)
{
	bool marcador = false;
	/* Pasos a seguir para los logs del sistema:
	1.- Primero saber el nombre del programa que se esta ejecutando
	2.- Segundo, crear el fichero donde se escribiran los logs de ese programa
	3.- Tercero, escribir los logs en el fichero. */
	// string nombre=busqueda_programa();

//	cout << " Debug." << endl;
	marcador = comprobar_carpetas();
	if( !marcador )
	{
		crear_directorios();
		crear_fichero_log();
	}

	escribir_log_debug(tipo, cadena);
}
```

Ejemplo de la colocación en disco de los ficheros de logs.

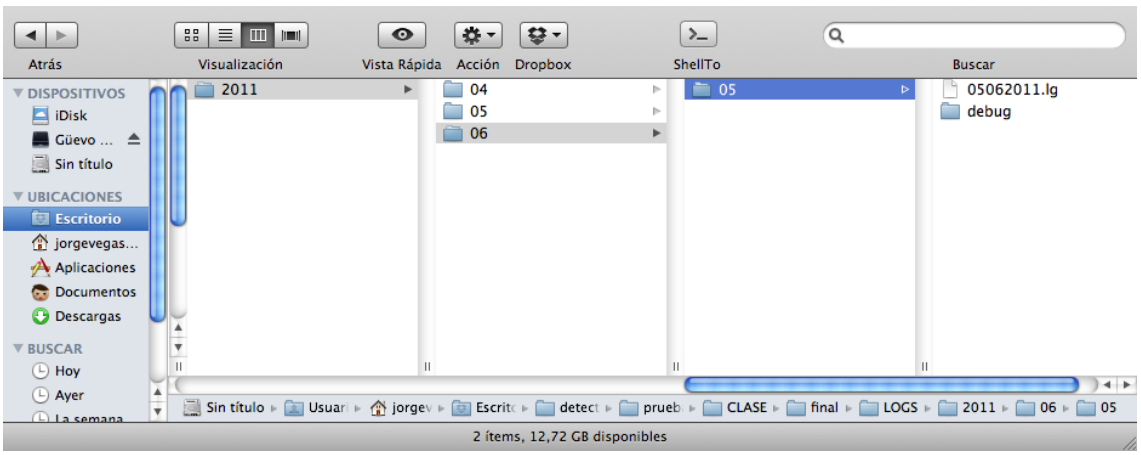


Ilustración 3 - Ejemplo localizacion ficheros de log

IMÁGENES

Creo que es relevante guardar las imágenes que capturamos de la cámara y guardarlas de una forma ordenada. Para ello y dada la estructura creada con los logs creo que es posible reutilizar dicha estructura con unas leves modificaciones. Los 'logs' los guardamos en una carpeta llamada LOGS y dentro de un entramado de carpetas correspondientes a la fecha del día. Para las imágenes he pensado en una estructura igual en sustituyendo la carpeta 'raíz' por *IMÁGENES* y el mismo entramado de carpetas referentes a la fecha de realización. Por último las imágenes tendrán un nombre correspondiente a su hora, minuto y segundo todo junto, aparte claro está la extensión *jpg* con la que guardamos los ficheros de imagen.

```
bool crear_directorios_imagen()
{
      // Para el tratamiento de la carpeta
      bool flag = false;
      string folder;
      string folder_save;
      string base;
      string aux;
      int carpeta;

      // Tratamiento de las carpetas
      flag = comprobar_directorio( "IMAGENES" );
      if( !flag)
      {
        system( "mkdir IMAGENES" );
      }

      carpeta = fecha_fichero_carpeta( 3 );
      base = "mkdir ";
      aux += "IMAGENES/";
      folder = itos( carpeta );
      folder_save = folder;
      flag = comprobar_directorio( aux+folder );
      if( !flag )
      {
        folder = base + aux + folder;
        aux = aux + folder_save;
        system( folder.c_str() );
      }
      else
      {
        aux = aux + folder;
      }

      carpeta = fecha_fichero_carpeta( 2 );
      aux += "/";
      folder = itos( carpeta );
      folder_save = folder;
      flag = comprobar_directorio( aux+folder );
      if( !flag )
      {
        folder = base + aux + folder;
        aux = aux + folder_save;
        system( folder.c_str() );
      }
      else
      {
        aux = aux + folder;
      }

      carpeta = fecha_fichero_carpeta( 1 );
      aux += "/";
      folder = itos( carpeta );
      folder_save = folder;
      flag = comprobar_directorio( aux+folder );
      if( !flag )
```

```
      {
       folder = base + aux + folder;
       aux = aux + folder_save;
       system( folder.c_str() );
      }
      else
      {
       aux = aux + folder;
      }

      return flag;
}
```

```
string ruta_imagen()
{
      string final;

      string folder;
      int carpeta;

      final = "IMAGENES/";

      carpeta = fecha_fichero_carpeta( 3 );

      folder = itos( carpeta );
      final = final + folder;

      carpeta = fecha_fichero_carpeta( 2 );

      final += "/";
      folder = itos( carpeta );
      final = final + folder;

      carpeta = fecha_fichero_carpeta( 1 );

      final += "/";
      folder = itos( carpeta );
      final = final + folder;
      final += "/";

      return final;
}
```

Anexo

Librerías externas requeridas

ANEXO – Librerías externas requeridas

Las librerías requeridas para ejecutar nuestro código son las siguientes:

- OpenCV
- Cvblob

Hay que destacar que *cvblob* es una librería complementaria a *OpenCV* y que su objetivo es facilitar el trabajo con blobs. Es decir para poder compilar la librería *cvblob* es requisito imprescindible tener ya compilada e instalada la librería *OpenCV*.

La librería se compilará en Ubuntu 8.04 para no tener problemas con las versiones de C/C++ instaladas. La compilación se realiza en una instalación de dicha distribución limpia sin ninguna actualización.

La versión más alta de la librería que podemos ejecutar en esta versión de Linux es la versión 2.2. La librería utiliza *cmake* para desempaquetar y compilar los ficheros. Los comandos a ejecutar son los siguientes:

- Entramos en el directorio de la librería *OpenCV*
- Creamos un directorio llamado */build* para contener de manera ordenada los resultados de la compilación.
- Ejecutamos *cmake ..* para configurar la compilación y generar el fichero de Makefile.
- Ejecutamos el *make*
- Instalamos la librería *sudo make instal.*

El termino *instalar* la librería se basa en copiar los archivos finales de la compilación a una ruta preestablecido del Sistema Operativo, en nuestro caso:

- Los ficheros de cabecera (.h) se copian a */usr/local/include/* . Los ficheros están contenidos dentro de las carpetas de nombre *opencv* y *opencv2*.
- Los ficheros de librería (.so por ser librería dinámica) se copian a la ruta */usr/local/lib/*

Destacar que por defecto la librería OpenCV se compila como un librería dinámica. En caso de desear obtener la librería de forma estática basta con seguir uno de estos dos pasos:

1. Modificar el fichero CMakeList.txt, cambiando el ON original por un OFF.
   # --
   # Build static or dynamic libs?
   # Default: dynamic libraries
   # --
   set(BUILD_SHARED_LIBS OFF CACHE BOOL "Build shared libraries (.dll/.so) instead of static ones (.lib/.a)")

2. Ejecutar como primer comando de la compilación de la librería OPenCV bajo CMAKE:
   cmake -DBUILD_SHARED_LIBS=OFF

En lo referente a la biblioteca complementaria *cvblob,* esta viene también empaquetada con *cmake* pero este no funciona correctamente en Mac y por lo tanto cambie esto por un *makefile* propio que realiza la misma labor de manera correcta en ambos sistemas operativos. Esta biblioteca se instala de la misma manera que *OpenCV* y en la misma ruta. La única diferencia es que esta biblioteca la compilo como una biblioteca estática (.lib“nombre”.a).

Anexo

Cross-compiling

ANEXO - CROSS COMPILING

Para ejecutar nuestro programa en el dispositivo Meshlium es mucho más eficaz compilar el código en un ordenador externo mucho más potente y posteriormente mandar el ejecutable y las librerías compiladas a Meshlium. Lo que se suele hacer en estos casos es "cross compiling". En nuestro caso se compila todo en un ordenador x86 en una distro de Linux (32bits) y luego se copian las librerías y el ejecutable a Meshlium y se dejan en los directorios correspondientes.

Comprobar que el directorio /usr/local/lib está incluido en /etc/ld.so.conf

En este apartado, no todas las distribuciones incluyen los directorios de las librerías compartidas a cachear por ***ldconfig*** en el archivo de configuración, ***/etc/ld.so.conf***, también utilizan el directorio ***/etc/ld.so.conf.d***, por lo que su edición sólo será necesaria en el caso de que el comando ***ldconfig*** no cachee las librerías compartidas ubicadas en ***/usr/local/lib***, en ese caso, abrimos con un editor de texto, el archivo de configuración ***/etc/ld.so.conf*** y añadimos la ruta correspondiente.

Un ejemplo:

```
/usr/X11R6/lib/Xaw3d
/usr/X11R6/lib
/usr/lib/Xaw3d
/usr/i386-suse-linux/lib
/usr/local/lib
/opt/kde3/lib
include /etc/ld.so.conf.d/*.conf
```

Posteriormente, ejecutamos el comando ***ldconfig*** para actualizar la caché de las librerías compartidas.

```
$ su
# ldconfig -v
```

Esto actualiza las librerías utilizadas por el sistema, recomendable ejecutarlo cada vez que se instale un programa.

ld es el programa que carga otros programas junto con las librerías compartidas que requieran y resuelve los símbolos. Para buscar librerías compartidas emplea la información de */var/run/ld.so.hints* (que puede examinarse con **ldconfig -r**).
Con **ldd** pueden examinarse las librerías compartidas que un programa requiere, por ejemplo:

```
ldd /usr/bin/less
```

Cuando se agregan librerías compartidas o se requiere que las librerias de nuevos directorios sean referenciados en /var/run/ld.so.hints puede ejecutar **ldconfig** (que por defecto busca nuevas librerías en /usr/lib).

Incluso un usuario puede establecer otros directorios donde buscar librerías compartidas especificándolos en la variable de entorno LD_LIBRARY_PATH (separar un directorio de otro con ':').

Primero pruebo con dos ejemplos sencillos en Meshlium la correcta configuración:

- Un programa que sólo usa la librería *OpenCV*
- Un programa que usa las librerías *OpenCV* y *cvblob*.

Ambos programas sólo buscan ser un modelo a pequeña escala de lo que luego requerirá nuestro proyecto de mayor entidad y tamaño.

Primero miro cuales son las dependencias del programa que sólo usa la librería OpenCV.
Dependencias del primer programa sencillo:

linux-gate.so.1 => (0xb7785000)
libopencv_core.so.2.3 => /usr/local/lib/libopencv_core.so.2.3 (0xb757e000)
libopencv_imgproc.so.2.3 => /usr/local/lib/libopencv_imgproc.so.2.3 (0xb7313000)
libopencv_highgui.so.2.3 => /usr/local/lib/libopencv_highgui.so.2.3 (0xb72e3000)
libopencv_ml.so.2.3 => /usr/local/lib/libopencv_ml.so.2.3 (0xb726e000)
libopencv_video.so.2.3 => /usr/local/lib/libopencv_video.so.2.3 (0xb7230000)
libopencv_features2d.so.2.3 => /usr/local/lib/libopencv_features2d.so.2.3 (0xb7148000)
libopencv_calib3d.so.2.3 => /usr/local/lib/libopencv_calib3d.so.2.3 (0xb70ac000)
libopencv_objdetect.so.2.3 => /usr/local/lib/libopencv_objdetect.so.2.3 (0xb704f000)
libopencv_contrib.so.2.3 => /usr/local/lib/libopencv_contrib.so.2.3 (0xb6ff3000)
libopencv_legacy.so.2.3 => /usr/local/lib/libopencv_legacy.so.2.3 (0xb6f42000)
libopencv_flann.so.2.3 => /usr/local/lib/libopencv_flann.so.2.3 (0xb6edc000)
libstdc++.so.6 => /usr/lib/i386-linux-gnu/libstdc++.so.6 (0xb6df1000)
libm.so.6 => /lib/i386-linux-gnu/libm.so.6 (0xb6dca000)
libgcc_s.so.1 => /lib/i386-linux-gnu/libgcc_s.so.1 (0xb6dae000)
libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb6c4d000)
libdl.so.2 => /lib/i386-linux-gnu/libdl.so.2 (0xb6c49000)
libpthread.so.0 => /lib/i386-linux-gnu/libpthread.so.0 (0xb6c30000)
librt.so.1 => /lib/i386-linux-gnu/librt.so.1 (0xb6c26000)
libz.so.1 => /lib/i386-linux-gnu/libz.so.1 (0xb6c11000)
libjpeg.so.62 => /usr/lib/i386-linux-gnu/libjpeg.so.62 (0xb6bf1000)
libpng15.so.15 => /usr/local/lib/libpng15.so.15 (0xb6bc8000)
libtiff.so.4 => /usr/lib/i386-linux-gnu/libtiff.so.4 (0xb6b6d000)
libjasper.so.1 => /usr/lib/i386-linux-gnu/libjasper.so.1 (0xb6b20000)
/lib/ld-linux.so.2 (0xb7786000)

Dependencias del segundo programa sencillo, que u:

linux-gate.so.1 => (0xb7763000)
libopencv_core.so.2.3 => /usr/local/lib/libopencv_core.so.2.3 (0xb755c000)
libopencv_imgproc.so.2.3 => /usr/local/lib/libopencv_imgproc.so.2.3 (0xb72f1000)
libopencv_highgui.so.2.3 => /usr/local/lib/libopencv_highgui.so.2.3 (0xb72c1000)
libopencv_ml.so.2.3 => /usr/local/lib/libopencv_ml.so.2.3 (0xb724c000)
libopencv_video.so.2.3 => /usr/local/lib/libopencv_video.so.2.3 (0xb720e000)
libopencv_features2d.so.2.3 => /usr/local/lib/libopencv_features2d.so.2.3 (0xb7126000)
libopencv_calib3d.so.2.3 => /usr/local/lib/libopencv_calib3d.so.2.3 (0xb708a000)
libopencv_objdetect.so.2.3 => /usr/local/lib/libopencv_objdetect.so.2.3 (0xb702d000)
libopencv_contrib.so.2.3 => /usr/local/lib/libopencv_contrib.so.2.3 (0xb6fd1000)
libopencv_legacy.so.2.3 => /usr/local/lib/libopencv_legacy.so.2.3 (0xb6f20000)
libopencv_flann.so.2.3 => /usr/local/lib/libopencv_flann.so.2.3 (0xb6eba000)
libstdc++.so.6 => /usr/lib/i386-linux-gnu/libstdc++.so.6 (0xb6dcf000)
libm.so.6 => /lib/i386-linux-gnu/libm.so.6 (0xb6da8000)
libgcc_s.so.1 => /lib/i386-linux-gnu/libgcc_s.so.1 (0xb6d8c000)
libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb6c2b000)

libpthread.so.0 => /lib/i386-linux-gnu/libpthread.so.0 (0xb6c12000)
libdl.so.2 => /lib/i386-linux-gnu/libdl.so.2 (0xb6c0e000)
librt.so.1 => /lib/i386-linux-gnu/librt.so.1 (0xb6c04000)
libz.so.1 => /lib/i386-linux-gnu/libz.so.1 (0xb6bef000)
libjpeg.so.62 => /usr/lib/i386-linux-gnu/libjpeg.so.62 (0xb6bcf000)
libpng15.so.15 => /usr/local/lib/libpng15.so.15 (0xb6ba6000)
libtiff.so.4 => /usr/lib/i386-linux-gnu/libtiff.so.4 (0xb6b4b000)
libjasper.so.1 => /usr/lib/i386-linux-gnu/libjasper.so.1 (0xb6afe000)
/lib/ld-linux.so.2 (0xb7764000)

Ahora hay que tener en cuenta la siguiente circunstancia. En Meshlium no es necesario copiar los ficheros de cabecera de la librería, basta con copiar los ficheros propios de la librería. Los ficheros de cabecera sólo son necesarios en la compilación y enlazado. Por lo tanto basta con copiar los ficheros que están dentro de /usr/local/lib/.

Los ficheros de la librería se copiab en la misma ruta en Meshlium, es decir en */usr/local/lib/*. Posteriormente basta con ejecutar un *ldconfig –v* para indexarlos.

PROBLEMAS CON EL CROSS-COMPILING

Los problemas principales que aparecen es que estamos trabajando con dos distribuciones diferentes de Linux y que por lo tanto es muy posible que tengan distintas versiones de las librerías por defecto instaladas. Hablo de librerías del sistema en general, por ejemplo las librerías de C o C++ por ejemplo.

Trabajando con la última versión de Ubuntu compilamos nuestros programas perfectamente pero al tratar de ejecutarlos en Meshlium obtenemos mensajes de error.
Comprobamos las dependencias de librerías dinámicas con errores para un programa sencillo que utiliza OpenCV.

```
meshlium:/vision/pruebas-cc/normal# ldd paint
./paint: /usr/lib/libjpeg.so.62: no version information available (required by
/usr/local/lib/libopencv_highgui.so.2.3)
./paint: /lib/libc.so.6: version `GLIBC_2.11' not found (required by
/usr/local/lib/libopencv_highgui.so.2.3)
./paint: /usr/lib/libstdc++.so.6: version `GLIBCXX_3.4.11' not found (required by
/usr/local/lib/libopencv_features2d.so.2.3)
./paint: /usr/lib/libstdc++.so.6: version `GLIBCXX_3.4.14' not found (required by
/usr/local/lib/libopencv_calib3d.so.2.3)
./paint: /usr/lib/libstdc++.so.6: version `GLIBCXX_3.4.11' not found (required by
/usr/local/lib/libopencv_contrib.so.2.3)
./paint: /usr/lib/libstdc++.so.6: version `GLIBCXX_3.4.11' not found (required by
/usr/local/lib/libopencv_flann.so.2.3)
        linux-gate.so.1 =>  (0xffffe000)
        libopencv_core.so.2.3 => /usr/local/lib/libopencv_core.so.2.3 (0xb7ee7000)
        libopencv_imgproc.so.2.3 => /usr/local/lib/libopencv_imgproc.so.2.3 (0xb7c7c000)
        libopencv_highgui.so.2.3 => /usr/local/lib/libopencv_highgui.so.2.3 (0xb7c4d000)
        libopencv_ml.so.2.3 => /usr/local/lib/libopencv_ml.so.2.3 (0xb7bd8000)
        libopencv_video.so.2.3 => /usr/local/lib/libopencv_video.so.2.3 (0xb7b9a000)
        libopencv_features2d.so.2.3 => /usr/local/lib/libopencv_features2d.so.2.3 (0xb7ab1000)
        libopencv_calib3d.so.2.3 => /usr/local/lib/libopencv_calib3d.so.2.3 (0xb7a15000)
        libopencv_objdetect.so.2.3 => /usr/local/lib/libopencv_objdetect.so.2.3 (0xb79b9000)
        libopencv_contrib.so.2.3 => /usr/local/lib/libopencv_contrib.so.2.3 (0xb795d000)
        libopencv_legacy.so.2.3 => /usr/local/lib/libopencv_legacy.so.2.3 (0xb78ac000)
        libopencv_flann.so.2.3 => /usr/local/lib/libopencv_flann.so.2.3 (0xb7845000)
        libstdc++.so.6 => /usr/lib/libstdc++.so.6 (0xb7757000)
```

libm.so.6 => /lib/libm.so.6 (0xb7731000)
libgcc_s.so.1 => /lib/libgcc_s.so.1 (0xb7724000)
libc.so.6 => /lib/libc.so.6 (0xb75e6000)
libdl.so.2 => /lib/libdl.so.2 (0xb75e1000)
libpthread.so.0 => /lib/libpthread.so.0 (0xb75c9000)
librt.so.1 => /lib/librt.so.1 (0xb75c0000)
libz.so.1 => /usr/lib/libz.so.1 (0xb75ab000)
libjpeg.so.62 => /usr/lib/libjpeg.so.62 (0xb758c000)
libpng15.so.15 => /usr/local/lib/libpng15.so.15 (0xb7564000)
libtiff.so.4 => /usr/lib/libtiff.so.4 (0xb750e000)
libjasper.so.1 => /usr/lib/libjasper.so.1 (0xb74bf000)
/lib/ld-linux.so.2 (0xb80e1000)

Problemas:

- Problema con libJPEG
- Versión de LIBC++ → requiere versión 3.4.11 y 3.4.14. Versión más reciente instalada en MESHLIUM es *GLIBCXX_3.4.10*
- *Versión de LIBC – requiere versión 2.11 y la versión instalada en Meshlium es*

meshlium:/vision/pruebas-cc/normal# aptitude show libc6
Package: libc6
State: installed
Automatically installed: no
Version: 2.7-18lenny7

Para tratar de solucionar este problema es importante comprobar que tanto en Meshlium como en un nuestra distro de Linux tengamos las mismas versiones de GLIBC y GLIBCXX. Para ello he sustituido mi Ubuntu 11.04 por la versión 10.04 LTS para realizar pruebas junto con un miembro del Equipo de Libelium que me está ayudando en esta tarea, Ernesto Sánchez.

Para comprobar las versiones instaladas de dichas librerías hay que utilizar los siguientes comandos de Linux.

- strings /usr/lib/libstdc++.so.6 | grep GLIBCXX
- strings /lib/libc.so.6 | grep GLIBC

Con esta versión de Ubuntu aún tengo versiones más actuales que las que tiene el dispositivo Meshlium. La distro elegida fue la anterior versión LTS de Ubuntu (8.04) y que utilizo virtualizada con VirtualBox desde mi sistema operativo de trabajo (MAC OS X 10.6 Snow Leopard). Para transferir los ficheros entre mi entorno de trabajo y la distro de Linux virtualizada uso el servicio de almacenamiento en la nube DropBox.

Además de estos problemas de versión de las librerías de C y C++ se requiere instalar en Meshlium otras librerías externas que son dependencias de OpenCV, son:

- libjasper
- libtiff
- libjpeg
- libzlib
- libpng

Posteriormente compiló el código con el makefile creado a tal efecto y consiguió un ejecutable que transferiré al dispositivo Meshlium mediante la siguiente orden:

scp [ejcutable] root@10.10.10.1:/vision/[carpeta correspondiente al ejemplo]

El dispositivo Meshlium siempre tiene asignada la misma dirección de IP y accedo a él como el usuario root. Anteriormente debo haber dado permiso para escribir en el dispositivo mediante la instrucción *remountrw* .

Para poder visualizar archivos de imágenes y otros ficheros de resultados es necesario transferir estos ficheros a Ubuntu. No es posible mandar directamente estos ficheros al SO de trabajo, en mi caso OSX, dado que el terminal nos dice que es posible un ataque de MAN IN THE MIDDLE lo cual concuerda con el montaje de software usado. La instrucción para descargar estas imágenes en Ubuntu es la misma scp que para subirlas sólo que antes de poder hacerlo es necesario tener instalado en Ubuntu el Secure Shell.
Para ser más precisos ejecutamos *sudo aptitude ssh* posteriormente miramos la dirección IP (10.10.10.12 en mi caso) de la máquina virtual de Ubuntu y ejecutamos la instrucción scp.

scp –r /IMÁGENES memmaker650@10.10. 10.12:/home/memmaker650/Escritorio/

Con esto copiamos todo el directorio /IMÁGENES de Meshlium al escritorio de nuestro máquina virtual.

Compilación de los ejemplos

Para compilar nuestros ejemplos utilizando estas dos bibliotecas basta con crear un *makefile* modelo que se pueda ir modificando según el número de ficheros que contenga nuestro proyecto.

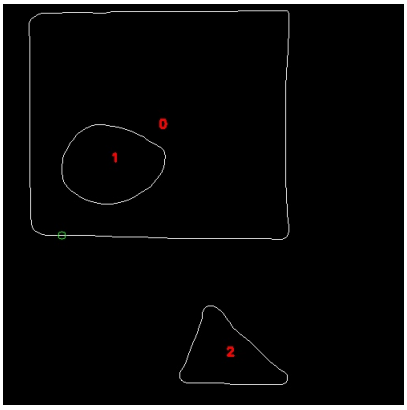
```
# La siguiente no es necesariamente requerida, se agrega para
# mostrar como funciona.

# FUNCIONA PERFECTAMENTE

CC = g++
CFLAGS = -g -Wall -O2
INCPATH = -I/usr/local/include/
LIBPATH = /usr/local/lib/libcvblob.a
OPTIONS = `pkg-config opencv --cflags --libs`
OBJ = test-NEW
DEST = est

# Reglas expl?citas

all: paso
	$(CC) $(CFLAGS) $(INCPATH) $(OPTIONS) $(OBJ).o $(LIBPATH) -o $(DEST)

paso:
	$(CC) -c $(CFLAGS) $(OPTIONS) $(OBJ).cpp

clean:
	$(RM) $(OBJ).o $(DEST)
```

Anexo

Cambios librería cvBlob

ANEXO – Cambios librería cvblob

He encontrado un error en la obtención de los puntos del contorno del blob. El funcionamiento por defecto de este método no daba como resultado la secuencia completa de puntos que conforman el contorno completo de un blob.



I

lustración 1 - Gráfico con blobs de ejemplo

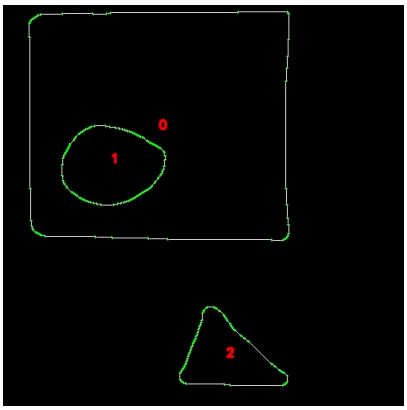
Nota: el circulo verde que se ve en la imagen es una mera comprobación del sistema de coordenadas de la librería OpenCV y de un programa de dibujo vectorial (Seashore.app) que uso para comprobar la validez de los puntos del contorno obtenidos.

El error se veía muy claramente por dos razones:

- Al visualizar el número de puntos del contorno de los blobs #0 y #1, obtenía que el blob #1 tenía el doble de puntos que el blob #1, algo imposible y que se corrobora a simple vista.
- Al plotear en una imagen todos los puntos (*CvPoint*) del contorno obtenidos con la instrucción (*cvConvertChainCodesToPolygon*) y la notoria diferencia respecto al método que dibuja los contornos (*cvRenderContourChainCode*).

Tras estas conclusiones el fallo tiene que estar en la librería *cvblob* y por lo tanto paso a detallar los cambios realizados en un método de dicha librería.

En esta imagen podemos apreciar el error, se imprimen en color verde los puntos del contorno del blob.



Código original:

```
CvContourPolygon *cvConvertChainCodesToPolygon(CvContourChainCode const *cc)
{
  CV_FUNCNAME("cvConvertChainCodesToPolygon");
  __CV_BEGIN__;
  {
    CV_ASSERT(cc!=NULL);

    CvContourPolygon *contour = new CvContourPolygon;

    unsigned int x = cc->startingPoint.x;
    unsigned int y = cc->startingPoint.y;
    contour->push_back(cvPoint(x, y));

    if (cc->chainCode.size())
    {
      CvChainCodes::const_iterator it=cc->chainCode.begin();
      CvChainCode lastCode = *it;

      x += cvChainCodeMoves[*it][0];
      y += cvChainCodeMoves[*it][1];
      ++it;

      for (; it!=cc->chainCode.end(); ++it)
      {
        if (lastCode!=*it)
        {
          contour->push_back(cvPoint(x, y));
          lastCode=*it;
        }

        x += cvChainCodeMoves[*it][0];
        y += cvChainCodeMoves[*it][1];
      }
    }

    return contour;
  }
  __CV_END__;
}
```

Código Modificado:

```
CvContourPolygon *cvConvertChainCodesToPolygon(CvContourChainCode const *cc)
{
  CV_FUNCNAME("cvConvertChainCodesToPolygon");
  __CV_BEGIN__;
  {
    CV_ASSERT(cc!=NULL);

    CvContourPolygon *contour = new CvContourPolygon;

    unsigned int x = cc->startingPoint.x;
    unsigned int y = cc->startingPoint.y;
    contour->push_back(cvPoint(x, y));


      CvChainCodes::const_iterator it=cc->chainCode.begin();
      CvChainCode lastCode = *it;

      x += cvChainCodeMoves[*it][0];
      y += cvChainCodeMoves[*it][1];

      ++it;

      for (it = cc->chainCode.begin(); it != cc->chainCode.end(); ++it)
```

```
        {
            x += cvChainCodeMoves[*it][0];
            y += cvChainCodeMoves[*it][1];

            contour->push_back(cvPoint(x, y));
        }

      return contour;
    }
    __CV_END__;
  }
```

Con la modificación de la librería los cambios son sustanciales, ya a primera vista se puede apreciar como se ha pintado por completo todo el contorno y que no hay ningún punto blanco como en la imagen de la página anterior. Esto confirma que los cambios están bien realizados.

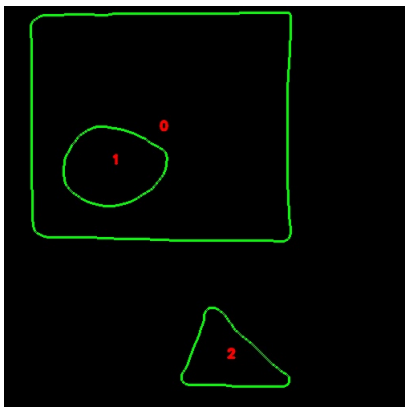


Ilustración 2 - Imagen con los puntos de contorno una vez modificada la librería.

Más cambios:

Para la realización del método llamado "rectángulo orientado" hemos tenido que realizar unos pequeños cambios en la estructura interna de la librería.

Añadidos

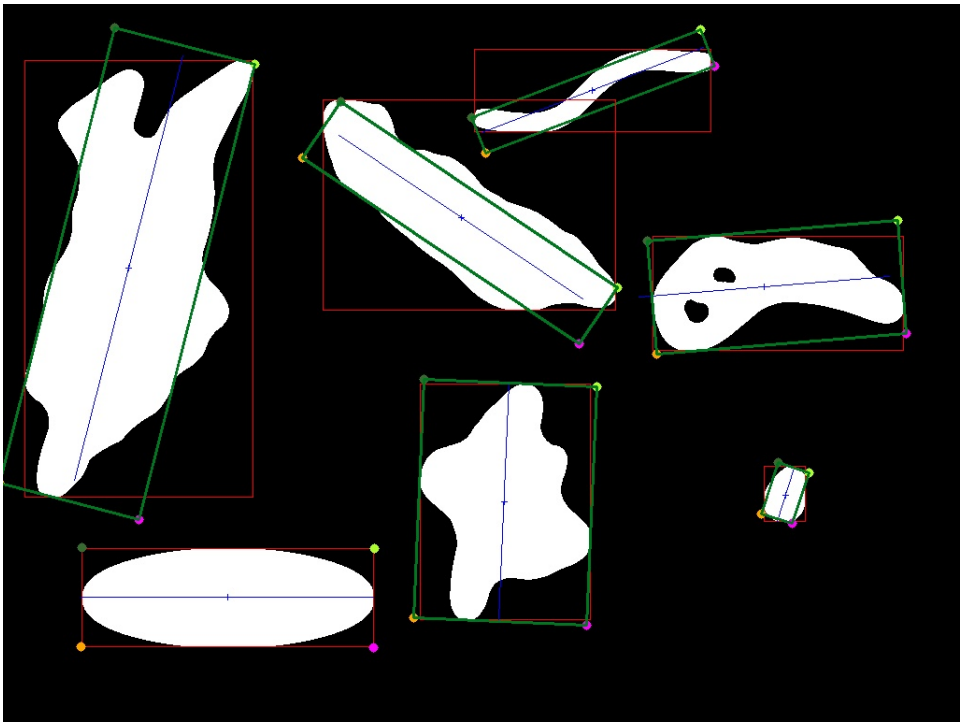
1. Puntos donde se sitúan los máximos y mínimos de ambas coordenadas
    a. *CvPoint* pminx;
    b. *CvPoint* pmaxx;
    c. *CvPoint* pminy;
    d. *CvPoint* pmaxy;
2. En el método *label* de *cvlabel.cpp*

    En la zona se calculan los valores de maxx, miny, etc. Se aprovecha para guardar también el punto donde esto sucede. Asimismo reasignar en la parte de consistencia estos puntos.
3. Por último los métodos para obtener las rectas paralelas y perpendiculares y los métodos para el dibujo de estas rectas, de momento se encuentran fuera de la librería. En un futuro habrá que integrarlos dentro de la librería.

Los métodos para el cálculo del rectángulo orientado se basan en cálculos matemáticos sencillos de pendiente de una recta, recta paralela y cálculo del punto de intersección entre dos rectas. El rectángulo orientado trata de ser más eficaz en el paso de los valores esquina al método de Homography (detección de matrículas) para obtener una buena tasa de matrículas identificadas.

Imagen de ejemplo donde podemos ver la diferencia entre el rectángulo frontal (rojo) y el rectángulo orientado (verde). El rectángulo frontal marca solamente los límites del blob en las 4 puntos cardinales. El rectángulo orientado usa esos puntos extremos para colocar rectas paralelas y perpendiculares a la recta de orientación del blob.

También se aprecia en el interior de los blobs la recta de orientación y el punto central o centro.



Código C/C++ de los métodos

```
// Estructura con la info del rectángulo orientado exterior
struct info_estructura
{
	CvBlob *b;

	CvPoint esq1;
	CvPoint esq2;
	CvPoint esq3;
	CvPoint esq4;
};
```

```
// Estructura de datos especial
struct ppendiente
{
	double pendiente;
	double n;

	double x;
	double y;

	bool error;
};
```

```
// Método para pintar el rectángulo orientado externo del blob.
void pintar_recta_orientacion(IplImage *imgFIN, double x11, double x22, double y11,
double y22)
{
	// Imprimo en rojo la linea orientada central
```

```
      cvLine(imgFIN, cvPoint(int(x11),int(y11)), cvPoint(int(x22),int(y22)),
CV_RGB(0.,0.,255.));
}

// Método para pintar el rectángulo orientado externo del blob.
void pintar_rectangulo_orientado(IplImage *imgEnd, CvBlob *b, struct info_estructura
info)
{
       // Pintar cruz del centro del blob.
      cvLine(imgEnd, cvPoint(int(b->centroid.x)-3, int(b->centroid.y)), cvPoint(int(b-
>centroid.x)+3, int(b->centroid.y)),CV_RGB(0.,0.,255.));
      cvLine(imgEnd, cvPoint(int(b->centroid.x), int(b->centroid.y)-3), cvPoint(int(b-
>centroid.x), int(b->centroid.y)+3),CV_RGB(0.,0.,255.));

      // Imprimir los 4 lados del rectángulo
      cvLine(imgEnd, info.esq1, info.esq2, cvScalar(32, 120, 0), 2);
      cvLine(imgEnd, info.esq2, info.esq4, cvScalar(32, 120, 0), 2);
      cvLine(imgEnd, info.esq3, info.esq4, cvScalar(32, 120, 0), 2);
      cvLine(imgEnd, info.esq3, info.esq1, cvScalar(32, 120, 0), 2);

      cvSaveImage("rect_orientado.jpg", imgEnd );
}
```

```
// Función para pintar los puntos extremo y qe deben coincidir dentro del rectángulo
frontal externo
void pintar_puntos_extremo(IplImage *imgEnd, CvBlob *b)
{
      // Pinto los puntos en azul
      cvCircle(imgEnd, b->pminx, 2, cvScalar(255,0,0), 5);
      cvCircle(imgEnd, b->pmaxx, 2, cvScalar(255,0,0), 5);
      cvCircle(imgEnd, b->pminy, 2, cvScalar(255,0,0), 5);
      cvCircle(imgEnd, b->pmaxy, 2, cvScalar(255,0,0), 5);

      cvSaveImage("Puntos_borde.jpg", imgEnd);
}
```

```
// Función para pintar los puntos extremo y qe deben coincidir dentro del rectángulo
frontal externo
void pintar_puntos_rectOrientado(IplImage *imgEnd, struct info_estructura inf)
{
      // Pinto los puntos de las esquinas del rectángulo orientado.
      cvCircle(imgEnd, inf.esq2, 2, cvScalar(47, 255, 173), 5); // Punto color verde
lima
      cvCircle(imgEnd, inf.esq1, 2, cvScalar(47, 107, 55), 5); // Punto color oliva
oscura
      cvCircle(imgEnd, inf.esq4, 2, cvScalar(255, 0, 255), 5); // Punto color magenta
      cvCircle(imgEnd, inf.esq3, 2, cvScalar(0, 165, 255), 5); // Punto color naranja

      cvSaveImage("Puntos_bordeOrientado.jpg", imgEnd);
}
```

```
// Función para pintar los números del label del blob
void pintar_label_blob(IplImage *imgFIN, CvBlob *b)
{
      // inicializamos la fuente de texto para escribir luego el label del blob en el
centro de este
      CvFont font;
      cvInitFont( &font, CV_FONT_VECTOR0, 0.5, 0.5, 0, 1.5, CV_AA);

      cvPutText(imgFIN, itos(b->label).c_str(), cvPoint(b->centroid.x +20, b->centroid.y
+20), &font, CV_RGB(255, 190, 44) );
}
```

```
// Transformación en punto-pendiente.
// Versión CvPoint
struct ppendiente recta2puntosApuntopendiente(CvPoint uno, CvPoint dos)
{
      double m;
      double n;
      struct ppendiente pp;

      m = (static_cast< double > (uno.y - dos.y)) / (static_cast< float > (uno.x -
dos.x));
```

```
      n = static_cast< double > (uno.y) - static_cast< double > (m * uno.x);

      if( n != (static_cast< double >(dos.y) - static_cast< double >(m * dos.x)) )
      {
        pp.error = true;
        pp.pendiente = 0;
        pp.x = 0;
        pp.y = 0;
      }

      pp.pendiente = m;
      pp.y = 0;
      pp.x = (-uno.x)/m;

      return pp;
}
```

```
// Transformación en punto-pendiente.
// Versión integer
struct ppendiente recta2puntosApuntopendiente(int uno_x, int uno_y, int dos_x, int
dos_y)
{
      double m;
      double n;
      struct ppendiente pp;

      m = (static_cast< double > (uno_y - dos_y)) / (static_cast< double > (uno_x -
dos_x));
      n = static_cast< double >(uno_y) - static_cast< double >(m * uno_x);

      if( n != (dos_y - (m * dos_x)) )
      {
        pp.error = true;
        pp.pendiente = 0;
        pp.x = 0;
        pp.y = 0;
      }

      pp.pendiente = m;
      pp.y = 0;
      pp.x = (-uno_x)/m;

      return pp;
}
```

```
// Metodo para obtener el punto de intersección entre dos puntos.
// Versión puntos como double
struct ppendiente interseccion_2rectas(double uno_x, double uno_y, double p, double
dos_x, double dos_y, double p2)
{
      struct ppendiente punto;
      punto.error = false;

      cout << "Método interseccion_2rectas v.Double" << endl;

      if( (uno_x < 0) || (uno_y < 0) || (dos_x < 0) || (dos_y < 0) )
      {
        punto.error = false;
        punto.x = 0;
        punto.y = 0;

        cout << " ¡¡¡ ERROR !!! Punto de entrada fuera del rango de visión" << endl;

        return punto;
      }

      punto.x = ((p * uno_x) - uno_y - (p2 * dos_x) + dos_y) / (p - p2);
      punto.y = (p * punto.x) - (p * uno_x) + uno_y;

      return punto;
}
```

```
// Metodo para obtener el punto de intersección entre dos puntos.
// Versión puntos como CvPoint
```

```
struct ppendiente interseccion_2rectas(CvPoint uno, double p1, CvPoint dos, double p2)
{
      struct ppendiente punto;
      punto.error = false;

      cout << "Método interseccion_2rectas v.CvPoint" << endl;

      if( (uno.x < 0) || (uno.y < 0) || (dos.x < 0) || (dos.y < 0) )
      {
        punto.error = false;
        punto.x = 0;
        punto.y = 0;

        cout << "Error" << endl;

        return punto;
      }

      punto = interseccion_2rectas( uno.x, uno.y, p1, dos.x, dos.y, p2);

      return punto;
}
```

```
// Metodo para obtener el punto de intersección entre dos puntos.
// Rectas en forma de 2puntos.
struct ppendiente interseccion_2rectas(CvPoint uno, CvPoint dos, CvPoint tres, CvPoint
cuatro)
{
      struct ppendiente punto;
      punto.error = false;

      cout << "--> Método interseccion_2rectas v.4puntos" << endl;

      if( (uno.x < 0) || (uno.y < 0) || (dos.x < 0) || (dos.y < 0) || (tres.x < 0) ||
(tres.y < 0) || (cuatro.x < 0) || (cuatro.y < 0) )
      {
        punto.x = 0;
        punto.y = 0;
        punto.error = true;

        cout << "Error" << endl;

        return punto;
      }

      double p, p2;
      struct ppendiente pp1, pp2;

      // Cálculo de la pendiente y un punto
      pp1 = recta2puntosApuntopendiente(uno, dos);
      pp2 = recta2puntosApuntopendiente(tres, cuatro);

      // Asigno las variables tras el cálculo
      p = pp1.pendiente;
      p2 = pp2.pendiente;

      // Aplico el método anterior. Así evito errores
      punto = interseccion_2rectas(uno.x, uno.y, p, tres.x, tres.y, p2);

      return punto;
}
```

```
// Metodo para obtener el punto de intersección entre dos puntos.
// Rectas en forma de 2puntos.
struct ppendiente recta_paralela(CvPoint uno, double p, IplImage *imagen)
{
      struct ppendiente pp;

      if( (uno.x < 0) || (uno.y < 0) )
      {
        pp.pendiente = 0.0;
        pp.n = 0.0;

        pp.x = 0.0;
```

```
        pp.y = 0.0;
        pp.error = true;

        return pp;
      }

      // Definición de las variables
      CvSize dimensiones = cvGetSize( imagen );
      CvPoint p1, p2;
      double y;

      // Operaciones
      y = p * (- uno.x) + uno.y;
      p1 = cvPoint( 0, (int) y);
      y = p * (dimensiones.width - uno.x) + uno.y;
      p2 = cvPoint( dimensiones.width, (int) y);

      // Pintar la línea paralela
      cvLine(imagen, p1, p2, cvScalar(0 ,255, 255), 2);

      return pp;
}
```

```
// Metodo para obtener el punto de intersección entre dos puntos.
// Rectas en forma de 2puntos.
struct ppendiente recta_paralela(CvPoint uno, struct ppendiente pp, IplImage *imagen)
{
      struct ppendiente ppend;

      if( (uno.x < 0) || (uno.y < 0) )
      {
        pp.pendiente = 0.0;
        pp.n = 0.0;

        pp.x = 0.0;
        pp.y = 0.0;
        pp.error = true;

        return ppend;
      }

      // Definición de las variables
      CvSize dimensiones = cvGetSize( imagen );
      CvPoint p1, p2;
      double y;

      y = pp.pendiente * (- uno.x) + uno.y;
      p1 = cvPoint( 0, (int) y);
      y = pp.pendiente * (dimensiones.width - uno.x) + uno.y;
      p2 = cvPoint( dimensiones.width, (int) y);

      // Pintar la línea paralela
      cvLine(imagen, p1, p2, cvScalar(0 ,255, 255), 2);

      return ppend;
}
```

```
// Método para ordenar los blobs orientados segun el eje y.
bool cmpY(const pair<CvLabel, CvBlob*>  &p1, const pair<CvLabel, CvBlob*> &p2)
{
  return p1.second->centroid.y < p2.second->centroid.y;
}
```

```
// Función para el cálculo de la Distancia Euclídea
float distancia(int x, int y, int xf, int yf)
{
      // Metodo para el cálculo de la distancia euclídea.
      float distancia;

      distancia = sqrt( pow( (float) fabs( x- xf ) , 2) + pow( (float) fabs( y - yf ),
2) );

      return distancia;
```

```
}
```

```
// Método para obtener el rectángulo orientado externo del blob.
struct info_estructura metodo_calculo_rectangulo_orientado(IplImage *imgEnd, CvBlob
*blob)
{
      struct info_estructura info;

      double angle = cvAngle(blob);
      double pend, pend2;
      double x1, y1, x2, y2;
      double lengthLine = MAX(blob->maxx-blob->minx, blob->maxy-blob->miny)/2.; // Esto
es incorrecto

      x1 = blob->centroid.x-lengthLine*cos(angle);
      y1 = blob->centroid.y-lengthLine*sin(angle);
      x2 = blob->centroid.x+lengthLine*cos(angle);
      y2 = blob->centroid.y+lengthLine*sin(angle);

      // Imprimo el rojo la linea orientada central
      // cvLine(imgEnd, cvPoint(int(x1),int(y1)), cvPoint(int(x2),int(y2)),
CV_RGB(0.,0.,255.));
      pintar_recta_orientacion(imgEnd, x1, x2, y1, y2);

    // Calculo de las pendientes.
    pend = (y2 - y1) / (x2 - x1);
    pend2 = (-1) / pend;

    // cout << "Angulo blob " << blob->label << " --> " << angle << endl;

      struct ppendiente mapa;
      double pi = 3.1415926535;

       if( ( (angle >(pi/4)) && (angle <((3*pi)/4)) ) || ( (angle <(-pi/4)) && (angle
>((-3*pi)/4)) ) )
       {
        // Pendiente de orientación con ángulo entre (pi/4) y (3*pi)/4 & (5*pi)/4
        mapa = interseccion_2rectas( blob->pminx.x, blob->pminx.y, pend, blob->pminy.x,
blob->pminy.y, pend2);
          info.esq1 = cvPoint(mapa.x, mapa.y);
          mapa = interseccion_2rectas( blob->pmaxx.x, blob->pmaxx.y, pend, blob-
>pminy.x, blob->pminy.y, pend2);
          info.esq2 = cvPoint(mapa.x, mapa.y);
          mapa = interseccion_2rectas( blob->pminx.x, blob->pminx.y, pend, blob-
>pmaxy.x, blob->pmaxy.y, pend2);
          info.esq3 = cvPoint(mapa.x, mapa.y);
          mapa = interseccion_2rectas( blob->pmaxx.x, blob->pmaxx.y, pend, blob-
>pmaxy.x, blob->pmaxy.y, pend2);
          info.esq4 = cvPoint(mapa.x, mapa.y);
      }
      else
      {
       // Pendiente de orientación con ángulo entre (7*pi)/4 & (pi/4) y (3*pi)/4 &
(5*pi)/4
       mapa = interseccion_2rectas( blob->pminy.x, blob->pminy.y, pend, blob->pminx.x,
blob->pminx.y, pend2);
          info.esq1 = cvPoint(mapa.x, mapa.y);
          mapa = interseccion_2rectas( blob->pmaxy.x, blob->pmaxy.y, pend, blob-
>pmaxx.x, blob->pmaxx.y, pend2);
          info.esq4 = cvPoint(mapa.x, mapa.y);
          mapa = interseccion_2rectas( blob->pminy.x, blob->pminy.y, pend, blob-
>pmaxx.x, blob->pmaxx.y, pend2);
          info.esq2 = cvPoint(mapa.x, mapa.y);
          mapa = interseccion_2rectas( blob->pmaxy.x, blob->pmaxy.y, pend, blob-
>pminx.x, blob->pminx.y, pend2);
          info.esq3 = cvPoint(mapa.x, mapa.y);
      }

      return info;
}
```