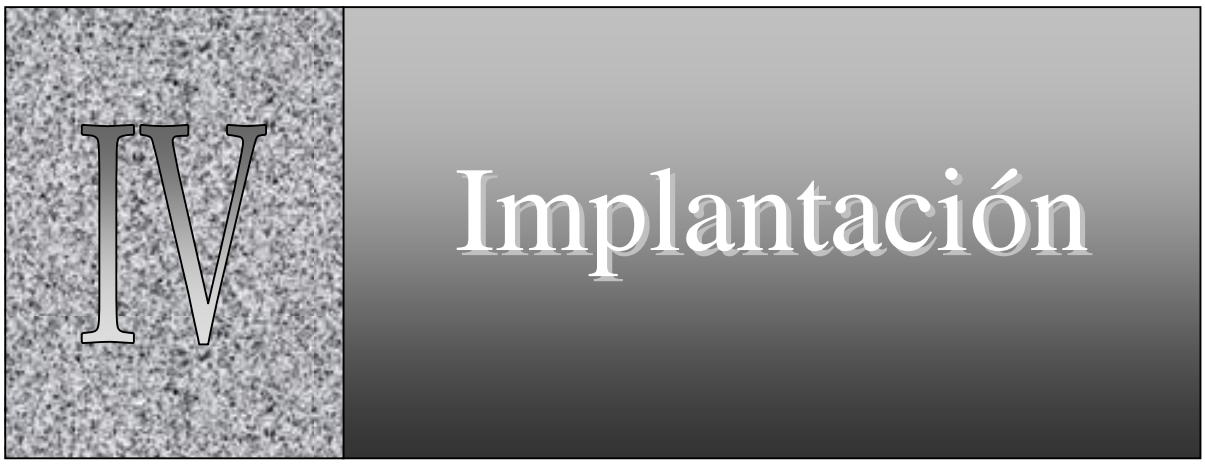


4.1 Introducción

La implantación de este proyecto se puede dividir en dos etapas:

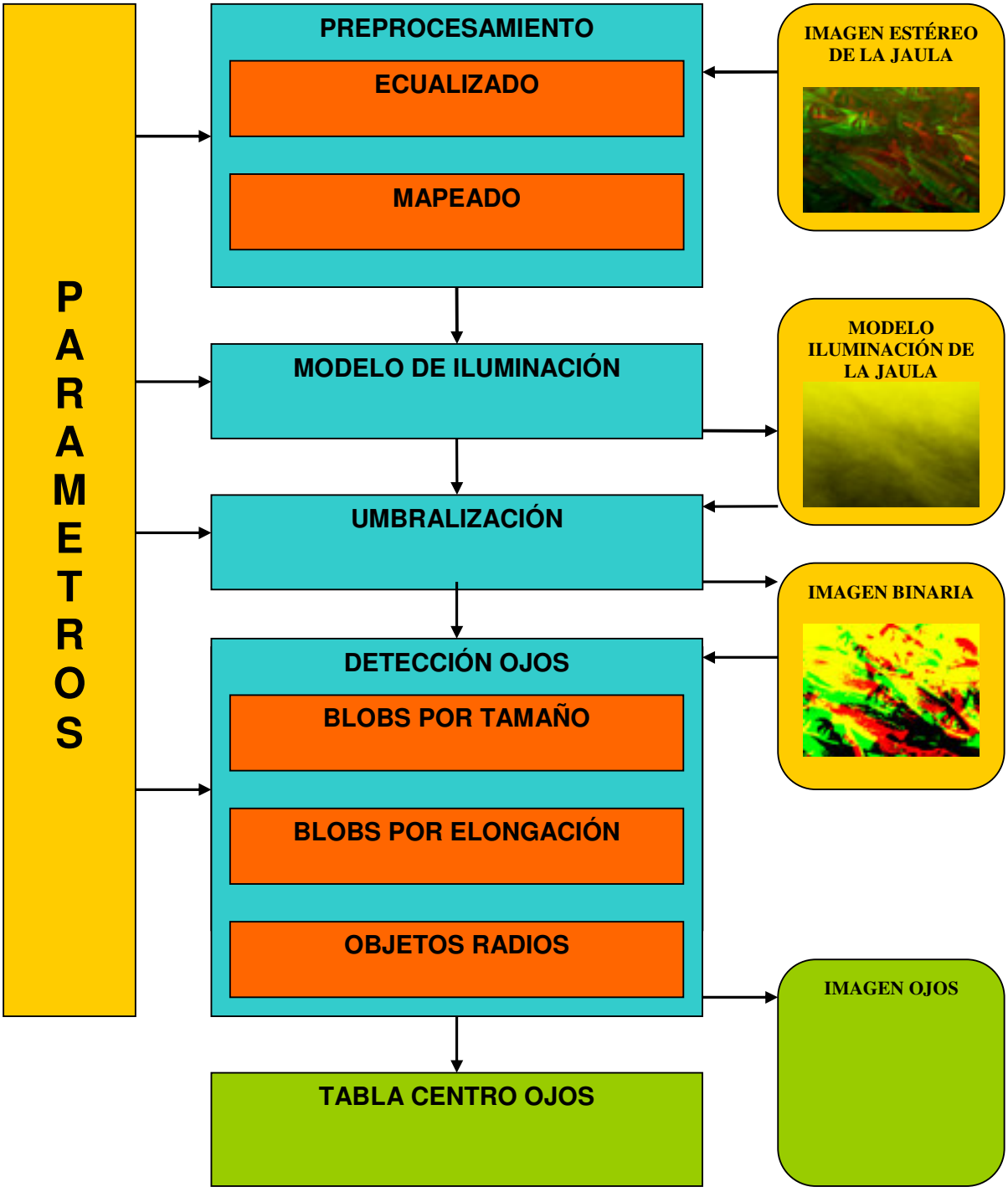
- Programación en MATLAB 7.0
- Paso a C++

En la primera etapa se ha desarrollado la totalidad de la aplicación en MATLAB 7.0. La razón de utilizar MATLAB como un primer paso en el desarrollo de la aplicación se debe a la cantidad de funciones para el procesamiento de imágenes contenida en el *Image Processing Toolbox*, lo que contribuyo a la familiarización del autor con las técnicas de percepción.

Al ser este un proyecto de desarrollo de software se ha debido emplear un lenguaje de programación para tal tarea, en concreto en el caso que nos ocupa se ha hecho uso de Visual C++ en la versión Visual Studio 6.0, ya que la aplicación existente estaba programada en este mismo lenguaje. Se destaca en este punto la potencia, así como, universalidad de esta herramienta en lo referente a programación orientada a objetos.

A continuación se estudia el modelo funcional de la aplicación, que se describe con cierto detalle para facilitar al lector una primera aproximación a la aplicación.

4.2 Arquitectura funcional de la aplicación



4.3 Parámetros principales

En este apartado se va a explicar cada uno de los parámetros variables de la aplicación, que se encuentran definidos en el archivo `constantes.h`.

4.3.1 Parámetros de imágenes

Son los parámetros que caracterizan a las imágenes de entrada de la aplicación.

Tamaño de imagen:

`SIZE_X`: Con las cámaras que se utiliza actualmente su valor es 640

`SIZE_Y`: Con las cámaras que se utiliza actualmente su valor es 480

Carpetas imágenes:

`PATH_ORIGEN`: Ruta de la carpeta donde se encuentran las imágenes originales.

`PATH_DESTINO`: Ruta de la carpeta donde se va a guardar las imágenes que genere la aplicación.

`IMG_IN`: Cadena de caracteres que se repite en todas las imágenes de la secuencia (Ej. Para `Imgxxx.bmp` sería `Img`)

`NUM_IMG`: Número de imágenes a procesar de la misma secuencia.

`NUMERO_NIVELES`: Número de niveles de grises (256)

4.3.2 Parámetros de preprocesado

Son parámetros que van a determinar si se realiza el preprocesado, y si este se realizara, determinan distintas formas.

Tenemos un primer grupo que son de tipo lógico:

`PREPROCESADO`: Determina si se realiza el preprocesado.

DESENTRELAZADO: Determina si se realiza el desentrelazado.

ECUALIZADO_OPTIMIZADO: Determina si se realiza una ecualización optimizada.

ECUALIZADO: Determina si se realiza la ecualización.

ILUMINACION: Determina si se genera el modelo de iluminación de la jaula marina.

UMBRALIZACION: Determina si se realiza la umbralización de las imágenes.

MAPEADO: Determina si se realiza un mapeado de los niveles de grises de la imagen.

Y un segundo grupo de tipo numérico:

PORCENTAJE_ILU: Porcentaje, en tanto por uno, para umbralizar en función del modelo de iluminación.

Los siguientes parámetros son para el mapeado, y ya se explicaron en [falta referencia]:

V_CONTRASTE_A

V_CONTRASTE_B

V_CONTRASTE_C

V_CONTRASTE_D

4.3.3 Parámetros de detección de ojos

Son parámetros que se van a utilizar para ajustar la aplicación en función del tipo de imágenes que tengamos. En [falta referencia] se da posibles valores para distintos experimentos.

AREA_MAX: Área máxima de blob admisible para ser candidato a ojo.

AREA_MIN: Área mínima de blob admisible para ser candidato a ojo.

ELONGACION_MAX: Elongación máxima de blob admisible para ser candidato a ojo.

ELONGACION_MIN: Elongación mínima de blob admisible para ser candidato a ojo.

`ALPHA`: ángulo de separación de radios de ojos en grados.

`PORCENTAJE_RADIOS`: porcentaje de radios correctos.

4.4 Clases

Se tiene por tanto, ya claro, el marco sobre el que realizar el trabajo y se presenta a continuación cada una de las clases que han utilizado en el PFC. El formato elegido para la presentación de estos datos es el mismo que se ha usado en proyectos anteriores [Munárriz] (identificación: modulo de programación <-> clase), esto se ha hecho así para dotar de cierta coherencia y unicidad a la aplicación en su conjunto.

A continuación se representa cual será el formato modelo para la descripción de las clases empleadas

Notación: Nombre de la clase

MÓDULO: **Módulo funcional al que pertenece la clase.**

FICHERO DE CABECERA: Nombre del fichero donde se define la clase.

FICHERO FUENTE: Nombre del fichero en el que se implementan los métodos, debe ser .cpp .

Nombre de la clase

ATRIBUTOS

Atributo1
Atributo2
.
.
.
AtributoN

FUNCIONES

Función1
Función2
.
.
.
FunciónN

DESCRIPCIÓN

Aquí se da una breve explicación de la funcionalidad de la clase.

ATRIBUTOS

Descripción de los atributos o datos miembros de la clase.

FUNCIONES:

Descripción de los métodos.

4.4.1 Clase CImagen

CLASE CImagen

MÓDULO: Procesamiento de imágenes

FICHERO DE CABECERA: L_Imagen.h

FICHERO FUENTE: L_Imagen.cpp

CImagen

ATRIBUTOS

```
ancho
alto
tabla_blobs
tabla_ojos
tabla
CabeceraBMP
```

FUNCIONES

AbrirImagen( )
AbrirImagen( )
ComprobarCabecera( )
CargarDatos( )
ReservarEspacio( )
DescargarImagen( )
InvertirImagen( )
EliminarEntrelazado( )
RecortarImagenFila( )
GuardarImagen( )
Preprocesar( )
Ecualizar( )
EcualizadoOptimizado( )
Mapeado( )
Umbralizar( )
LimpiarPixelsolitario( )
DetectarOjos( )
BlobsBySize( )
BlobsRedondeados( )
BlobsRadios( )

DESCRIPCIÓN

Esta clase soporta la totalidad de los algoritmos de preprocesamiento, corrección de iluminación, umbralización y detección de puntos característicos.

ATRIBUTOS

○ **ancho**

Anchura de las imágenes a procesar.

○ **alto**

Altura de las imágenes a procesar.

○ **tabla_blobs[2][nº blobs][2]**

Tabla en la que se almacena la posición del centro de gravedad de los blobs que se van a analizar para la detección de ojos. Almacena los blobs de los dos canales estéreo.

○ **tabla_ojos[2][nº ojos][2]**

Tabla en la que se almacena la posición del centro de gravedad de los ojos que se han detectado, como resultado de la función `DetectarOjos( )`. Almacena los blobs de los dos canales estéreo.

○ **CabeceraBMP**

Estructura que almacena la cabecera de los ficheros BMP.

FUNCIONES

□ **AbrirImagen( )**

Signatura: `AbrirImagen(CString nombre)`

Parámetro:

- nombre: Cadena de caracteres que indica el archivo BMP que se va a abrir.

Función que abre el archivo indicado por el parámetro `nombre` y carga los datos en `tabla`.

Para ello se utiliza una secuencia de funciones, mas concretamente se llama a las siguientes funciones:

- `ComprobarCabecera()`
- `ReservarEspacio()`
- `CargarDatos()`

Devuelve -1 en caso de error, y 0 en caso contrario.

□ **AbrirImagen( )**

Signatura: `AbrirImagen(int tamx, int tamy)`

Parámetros:

- tamx: anchura de la imagen.

- tamy: altura de la imagen.

Función que reserva espacio para tabla.

Para ello se llama a la función:

- `ReservarEspacio()`

Devuelve -1 en caso de error, y 0 en caso contrario.

□ **ComprobarCabecera( )**

Signatura: `ComprobarCabecera(FILE *fichero)`

Parámetro:

- fichero: Descriptor del fichero en el que se va a comprobar la cabecera.

Función que comprueba que la cabecera del fichero indicado por el descriptor de fichero sea correcta.

Devuelve -1 en caso de error, y 0 en caso contrario

□ **CargarDatos( )**

Signatura: `CargarDatos(FILE *fichero)`

Parámetro:

- fichero: Descriptor del fichero que contiene los datos que se van a cargar.

Función que carga los datos de un archivo BMP en `tabla`. Carga los datos de los dos canales útiles del estéreo.

Para ello se utiliza una secuencia de funciones, mas concretamente se llama a las siguientes funciones:

- `ComprobarCabecera()`
- `ReservarEspacio()`
- `CargarDatos()`

Devuelve -1 en caso de error, y 0 en caso contrario.

□ **ReservarEspacio( )**

Signatura: `ReservarEspacio(void)`

Función que reserva memoria dinámica para tabla[2][alto][ancho] del objeto que utilice este método.

Devuelve -1 en caso de error, y 0 en caso contrario.

□ **DescargarImagen( )**

Signatura: `DescargarImagen(void)`

Función que libera la memoria dinámica que se había reservado para tabla[2][alto][ancho]

Devuelve -1 en caso de error, y 0 en caso contrario.

□ **InvertirImagen( )**

Signatura: `InvertirImagen(CImagen original)`

Parámetro:

- original: objeto de la clase CImagen que contiene la imagen que se desea invertir.

Función que invierte los niveles de grises de la imagen contenida en el objeto `original` y la almacena en el objeto que utilice el método.

Para ello llama a la función:

- `AbrirImagen()`

Devuelve -1 en caso de error, y 0 en caso contrario.

□ **EliminarEntrelazado( )**

Signatura: `EliminarEntrelazado(CImagen org)`

Parámetro:

- org: objeto que contiene la imagen en la que se desea eliminar el entrelazado.

Función que elimina el entrelazado de la imagen contenida en `org` y la almacena sin entrelazar en el objeto que utilice el método.

Para ello llama a la función:

- `AbrirImagen()`

Devuelve -1 en caso de error, y 0 en caso contrario.

□ **RecortarImagenFila( )**

Signatura: `RecortarImagenFila(CImagen orginal,int filas)`

Parámetros:

- **original:** objeto que contiene la imagen en la que se desea recortar filas.
- **filas:** número de filas que se desea eliminar de la imagen.

Función que recorta un número de filas de la parte superior de la imagen. Se utiliza en imágenes quemadas debido a la influencia del sol en la parte superior de la imagen. Almacena la imagen recortada en el objeto que utilice el método.

Para ello llama a la función:

- `AbrirImagen()`

Devuelve -1 en caso de error, y 0 en caso contrario.

□ **GuardarImagen( )**

Signatura: `GuardarImagen(CString dest)`

Parámetro:

- **dest:** Cadena de caracteres que indica el archivo BMP en el que se va a guardar la imagen.

Función que guarda la imagen contenida en tabla en el archivo BMP indicado por la cadena de caracteres dest.

Devuelve -1 en caso de error, y 0 en caso contrario.

□ **Preprocesar( )**

Signatura: `Preprocesar()`

Función que realiza el preprocesado de las imágenes originales de la secuencia que se vaya a analizar y genera el modelo de iluminación para la posterior umbralización. Realizará distintos preprocesados, según las necesidades de cada localización, en función de ciertos parámetros de `constantes.h`.

Para ello se utiliza una secuencia de funciones, más concretamente se llama a las siguientes funciones:

- `AbrirImagen()`
- `EliminarEntrelazado()`
- `Ecualizar()`
- `EcualizadoOptimizado()`
- `Mapeado()`
- `GuardarImagen()`
- `DescargarImagen()`

Devuelve -1 en caso de error, y 0 en caso contrario

□ **Ecualizar( )**

Signatura: `Ecualizar(CImagen org)`

Parámetro:

- org: objeto que contiene la imagen que se desea ecualizar

Función que realiza el ecualizado del histograma de la imagen contenida en tabla del objeto que utilice el método.

Devuelve -1 en caso de error, y 0 en caso contrario

□ **EcualizadoOptimizado( )**

Signatura: `EcualizadoOptimizado(CImagen org)`

Parámetro:

- org: objeto que contiene la imagen que se desea ecualizar.

Función que realiza un ecualizado mejorado de la imagen contenida en el objeto que utilice el método.

Para ello se utiliza una secuencia de funciones, más concretamente se llama a las siguientes funciones:

- `AbrirImagen()`
- `InvertirImagen()`
- `Ecualizar()`

Devuelve -1 en caso de error, y 0 en caso contrario.

□ **Mapeado( )**

Signatura: `Mapeado(CImagen org)`

Parámetro:

- org: objeto que contiene la imagen que se desea mapear.

Función que aplica una transformación a la imagen aumentando el contraste en un intervalo de niveles de grises y reduciéndolo en el resto. Utiliza una serie de parámetros configurables que se encuentran en `constantes.h`.

Devuelve -1 en caso de error, y 0 en caso contrario.

□ **Umbralizar( )**

Signatura: `Umbralizar(int n,CImagen ilu)`

Parámetros:

- n: numero de secuencia de imagen.

- ilu: modelo de iluminación de la jaula marina.

Función que realiza la umbralización de la imagen contenida en el objeto que utilice el método en función del modelo de iluminación de la jaula marina contenido en el objeto `ilu`.

Para ello se utiliza una secuencia de funciones, más concretamente se llama a las siguientes funciones:

- `AbrirImagen()`
- `LimpiarPixelsolitario()`
- `DescargarImagen()`
- `GuardarImagen()`

Devuelve -1 en caso de error, y 0 en caso contrario.

□ **LimpiarPixelsolitario( )**

Signatura: `LimpiarPixelsolitario(void)`

Función que busca píxeles aislados en una imagen binaria y los elimina.

Devuelve -1 en caso de error, y 0 en caso contrario.

□ **DetectarOjos( )**

Signatura: `DetectarOjos(int n,CImagen binaria)`

Parámetros:

- n: numero de secuencia de imagen.

- binaria: imagen binaria en la que se va realizar la detección de ojos de lubinas.

Función que realiza el proceso de búsqueda de ojos de lubinas en imágenes binarias. Cuando detecta un ojo almacena la posición del centro de gravedad de dicho ojo en `tabla_ojos` para un procesado posterior.

Para ello se utiliza una secuencia de funciones, más concretamente se llama a las siguientes funciones:

- `AbrirImagen()`
- `InvertirImagen()`
- `BlobsBySize()`
- `BlobsRedondeados()`
- `BlobsRadios()`
- `GuardarImagen()`

Devuelve -1 en caso de error, y 0 en caso contrario.

□ **BlobsBySize( )**

Signatura: `BlobsBySize(int n)`

Parámetro:

- n: numero de secuencia de imagen.

Función que busca en una imagen binaria blobs cuya área esté entre los valores deseados, definidos por `AREA_MAX` y `AREA_MIN` del archivo `constantes.h`. Almacena la posición del centro de gravedad de los blobs detectados de cada canal en `tabla_blobs[2][numero_blobs][2]`.

Para ello se utiliza una secuencia de funciones de la clase `cmImageBlobAnalyser`, más concretamente se llama a las siguientes funciones:

- `Detect()`
- `ExtractFeatures()`
- `GetBlob()`

Devuelve un puntero a una tabla que contiene el número de blobs por canal que se encuentran en la zona deseada en cuanto a área se refiere.

□ **BlobsRedondeados( )**

Signatura: `BlobsBySize(int n, int *numero_blobs)`

Parámetros:

- n: numero de secuencia de imagen.

- *numero_blobs: tabla que contiene el número de blobs por canal que se va a procesar.

Función que detecta blobs con forma redondeada en una imagen binaria. No va a comprobar todos los blobs de la imagen, sólo comprueba cierto número de ellos (indicado por el parámetro `numero_blobs`), cuya posición está almacenada en `tabla_blobs`.

Para decidir qué blobs son redondeados se utiliza los parámetros `ELONGACION_MAX` y `ELONGACION_MIN` definidos en `constantes.h`.

Tras realizar la detección de blobs redondeados, se actualizará `tabla_blobs` con los blobs redondeados.

Devuelve un puntero a una tabla que contiene el número de blobs por canal que se han considerado como redondeados.

□ **BlobsRadios( )**

Signatura: `BlobsRadios(CImagen binaria, int sec, int *numero_blobs)`

Parámetros:
- binaria: imagen binaria.

- sec: numero de secuencia de imagen.

- *numero_blobs: tabla que contiene el número de blobs por canal que se va a procesar.

Función que busca en una imagen binaria blobs que cumplan los requisitos del criterio de los radios, explicado en el apartado. No se va a comprobar todos los blobs de la imagen, sólo comprueba cierto número de ellos (indicado por el parámetro `numero_blobs`), cuya posición está almacenada en `tabla_blobs`.

Almacena la posición del centro de gravedad de los blobs detectados de cada canal en `tabla_ojos[2][numero_ojos][2]`.

Devuelve un puntero a una tabla que contiene el número de ojos detectados por canal.

4.4.2 Clase cmImageBlobAnalyser

CLASE cmImageBlobAnalyser

MÓDULO: Procesamiento de blobs

FICHERO DE CABECERA: blob.h

FICHERO FUENTE: blob.cpp

cmImageBlobAnalyser

ATRIBUTOS

```
maximum_label
maximum_blob
labels_number
blobs_number
image_width
image_height
*equivalentes
*blobs_image
*blobs_data
```

FUNCIONES

cmImageBlobAnalyser ( )
~cmImageBlobAnalyser ( )
Detect ( )
***GetBlobsImage ( )**
GetBlobsNumber ( )
ExtractFeatures ( )
GetBlob ( )
LabelPixels ( )
SolveEquivalences ( )
ArrangeLabels ( )
TransferColours)
GetLabel ( )
SetLabel ( )

DESCRIPCIÓN

Esta clase soporta la totalidad de los algoritmos de procesamiento de blobs.