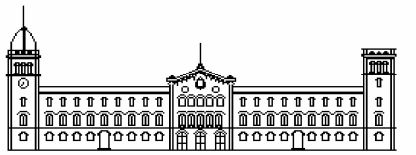


Treball fi de carrera

ENGINYERIA TÈCNICA EN INFORMÀTICA DE SISTEMES

Facultat de Matemàtiques
Universitat de Barcelona

RECONOCIMIENTO DE OBJETOS

Alberto Escudero Pardo

Director: Sergio Escalera Guerrero
Realitzat a: Departament de Matemàtica Aplicada i Anàlisi. UB

Barcelona, 18 de febrer de 2009

Resumen

En la actualidad disponemos de los medios necesarios para fabricar robots que simulen el comportamiento humano en determinadas actividades. Sin embargo, para realizar un buen uso de estos robots necesitamos que sean capaces de interactuar con su entorno. Nosotros somos capaces de realizar esta interacción a través de nuestros sentidos.

En este proyecto se realiza una aproximación al sentido de la vista a través de la visión por computador mediante la implementación de un software que permite analizar imágenes y detectar los objetos que puedan aparecer. La detección de estos objetos amplia la inteligencia artificial de los robots y su capacidad para interactuar.

Se ha obtenido un software capaz de detectar objetos en imágenes e indicar su situación dentro de ella.

Resum

En l'actualitat disposem dels medis necessaris per fabricar robots que simulin el comportament humà en determinades activitats. Tot i això, per a realitzar un bon ús d'aquests robots necessitem que siguin capaços d'interactuar amb el seu entorn. Nosaltres som capaços de realitzar aquesta interacció mitjançant els nostres sentits.

En aquest projecte es realitza una aproximació al sentit de la vista a través de la visió per computador mitjançant la implementació d'un software que permet analitzar imatges y detectar els objectes que puguin aparèixer. La detecció d'aquests objectes amplia l'intel·ligència artificial dels robots y la seva capacitat per interactuar.

S'ha obtingut un software capaç de detectar objectes en imatges i indicar la seva situació dins d'aquesta.

Abstract

Nowadays we have the resources in order to create robots that can simulate the human behaviour in specific situations. However, if we want to do a good use of those robots we need that they are able to interact with their environment. We can do that interaction through our senses.

In this project is realized an approximation to eyesight using computer vision through a software implementation that allow us to analyze images and to detect the objects that may appear. The detection of those objects increases the robots' artificial intelligence and their ability to interact.

We have obtained software that is able to detect objects inside images and to indicate their situation.

Índice

1. Introducción

1.1. Problema

En la actualidad disponemos de los medios necesarios para fabricar robots que simulen el comportamiento humano o que nos ayuden a realizar determinadas actividades de la vida cotidiana. Sin embargo, para realizar un buen uso de estos robots necesitamos que sean capaces de interactuar con su entorno. En particular las personas realizamos esta interacción a través de los sentidos. En el caso del sentido de la vista, para poder interactuar con el entorno, el sistema visual debe de ser capaz de reconocer los diferentes objetos que aparezcan en nuestro campo visual. Para ello, poseemos descripción de los objetos conocemos. Cos esta información somos capaces de localizar los diferentes objetos que aparecen en nuestro entorno. Así pues, para dar solución al problema de la visión artificial necesitamos definir dos etapas:

1) Un descriptor, que utilizaremos para describir tanto a los objetos como a la escena que los contenga.
2) Un algoritmo clasificador que, gracias a las descripciones de diferentes objetos, nos indique si otra descripción forma parte de nuestra colección de objetos.

1.2. Estado del arte

Clasificadores

Un clasificador es un método para determinar la posible clase de un objeto desconocido, o evento sobre la base de un número de casos de cada una de las clases, conocido como el conjunto de entrenamiento.

El primer paso en la clasificación es la extracción de características, que es donde cada instancia en el conjunto de entrenamiento se expresa como un vector de medidas. Cuando están siendo clasificadas las imágenes, este vector podría estar definido a partir de las intensidades de los píxeles, pero con frecuencia se aplica un paso de reducción de características. Las medidas se denominan habitualmente características, y pueden ser reales, enteros o categóricas. El espacio que abarca todas las combinaciones posibles de características se le denomina espacio de características. Para algunos problemas no todas las mediciones están disponibles y esto se conoce como datos faltantes.

Una vez las características se han extraído hay dos posibles casos, que en general se manejan de forma muy diferente. En el primer caso, la clase de cada instancia del conjunto de entrenamiento se pone a disposición del clasificador. En el segundo caso, la información no está disponible, y esto se llama clasificación sin supervisión o “clustering”.

Clasificación supervisada

Clasificación supervisada es donde la clase de cada una de las instancias en el conjunto de entrenamiento es conocida.

Los clasificadores supervisados más utilizados son los clasificadores binarios, que distinguen entre dos tipos de objetos o eventos. La función de clasificación se conoce como discriminante. En el caso de los clasificadores multiclase estos se generan frecuentemente a partir de un número de clasificadores binarios combinados.

Rendimiento de un clasificador supervisado

La salida de un clasificador es una estimación de la clase más probable dado un espacio de características. La medida obvia del rendimiento es comparar la estimación de la clase contra la otra clase. El error del clasificador suele venir dado por el número total de casos mal clasificados.

El rendimiento real de un clasificador viene dado por las clasificaciones realizadas sobre datos que nunca ha visto antes. Mejorar el rendimiento del conjunto de entrenamiento puede tener el aparentemente efecto paradójico de empeorar la clasificación en otros datos (un efecto que se conoce como sobreentrenamiento). Por esta razón, al evaluar el rendimiento de un clasificador, todos los datos disponibles se dividen en tres conjuntos distintos. El primero es el conjunto de entrenamiento, el segundo es un conjunto de validación que se utiliza para dar una medida intermedia del rendimiento del clasificador que tiene un conjunto de los parámetros que hay que ajustar. El conjunto final es un conjunto de prueba, que se utiliza para evaluar el rendimiento del clasificador ajustado.

Tipos de clasificadores supervisados

1. Discriminante lineal de Fisher [1]
2. Vecinos más cercanos [2]
3. Máquinas de Soporte de Vectores [3]
4. Adaboost [4]

Clasificación sin supervisión

Un algoritmo de agrupamiento (en inglés, *clustering*) es un procedimiento de agrupación de una serie de vectores según criterios habitualmente de distancia; se trata de disponer los vectores de entrada de forma que estén más cercanos aquellos que tengan características comunes [5].

Un clasificador de clustering representa un problema ya que los datos pueden revelar clusters de diferentes formas y tamaños. Además, el número de clusters en los datos dependen de la resolución con que vemos los datos [6].

Localización de objetos en imágenes

Cuando queremos localizar un objeto en una imagen, una técnica utilizada consiste en recorrer toda la imagen aplicando múltiples ventanas detectoras de diferentes tamaños. Para cada ventana se aplica algún clasificador que nos indica si existe el objeto que intentamos. Al ser un algoritmo que se basa en diversos recorridos de la imagen, el desplazamiento de la ventana a través de ella es un factor muy importante. Con la disminución del desplazamiento de la ventana, el tiempo de proceso crece exponencialmente. Por otro lado, un desplazamiento demasiado grande podría comportar la no detección de objetos por no quedar localizados dentro de su ventana.

1.3. Propuesta

En este proyecto proponemos una solución de clasificación binaria basada en el trabajo de Viola y Jones [7].

Para el proceso de extracción de características de imágenes se utilizará el descriptor circular implementado por el proyectista Mario Guitart Matamoros, basado en el descriptor BSM.

Para el entrenamiento del clasificador se utilizara Adaboost, que es un algoritmo de aprendizaje formulado por Yoav Freund y Robert Schapier. Este algoritmo es adaptable, en el sentido que la creación de los siguientes clasificadores se ajusta a favor de las instancias clasificadas erróneamente por los anteriores clasificadores. Es sensible al ruido en los datos y a los valores atípicos. Sin embargo, es menos susceptible al sobreentrenamiento que la mayoría de algoritmos.

Para la detección de objetos se usará un recorrido de la imagen mediante windowing. A cada ventana se le aplicará una cascada de clasificadores entrenados mediante Adaboost. Mediante este algoritmo seremos capaces de describir y localizar los objetos contenidos en imágenes.

1.4. Estructura de la memoria

La memoria está estructurada en los siguientes apartados:

- Metodología. En esta sección se describirán los algoritmos utilizados para elaborar el software, así como su diseño y las herramientas utilizadas para realizarlo.
- Resultados. En este apartado se expondrán los experimentos realizados para evaluar el software y los resultados obtenidos.
- Conclusiones y discusiones. En esta sección se presentarán las conclusiones a las que se ha llegado tras realizar el proyecto, así como posibles mejoras o cambios que se pudiesen realizar.
- Bibliografía. Contiene todas las referencias utilizadas para realizar la memoria del proyecto.
- Apéndices. Se explican los contenidos del CD adjunto, así como otros documentos relacionados que complementen los estudios realizados en este proyecto.

2. Metodología

En esta apartado se describirán los algoritmos utilizados para elaborar el software, así como su diseño y las herramientas utilizadas para realizarlo.

2.1. Análisis

En la sección de análisis se detalla la distribución de tareas a lo largo del proyecto, los costes relacionados y los casos de uso de la aplicación.

2.1.1. Diagramas de Gantt del proyecto y Costes

El proyecto se programó para que la distribución de las tareas fuese como se observa en la imagen 1. Sin embargo, a causa de problemas en la fase de integración del descriptor en el software, se tuvieron que alargar los plazos tal y como se puede ver en la imagen 2.

Numero	Tarea	Inicio	Fin	Duracion	T4 - 2008			T1 - 2009		
					Octubre	Noviembre	Diciembre	Enero	Febero	Marzo
1	Introducción OpenCV y Diseño	10/5/2008	10/15/2008	7						
2	Programación del Detector	10/14/2008	12/17/2008	46						
3	Integración Descriptor	10/21/2008	12/17/2008	41						
4	Entrenamiento Clasificador	10/28/2008	1/8/2009	52						
5	Testeo	10/28/2008	1/8/2009	52						
6	Creación Interfaz	12/17/2008	1/8/2009	16						
7	Redacción Memoria	1/8/2009	2/11/2009	24						

Imagen 1.Diagrama Gantt Original

Numero	Tarea	Inicio	Fin	Duracion	T4 - 2008			T1 - 2009		
					Octubre	Noviembre	Diciembre	Enero	Febero	Marzo
1	Introducción OpenCV y Diseño	10/5/2008	10/15/2008	7						
2	Programación del Detector	10/14/2008	1/9/2009	63						
3	Integración Descriptor	10/21/2008	1/21/2009	65						
4	Entrenamiento Clasificador	10/28/2008	1/21/2009	60						
5	Testeo	10/28/2008	2/1/2009	69						
6	Creación Interfaz	12/17/2008	2/1/2009	33						
7	Redacción Memoria	1/8/2009	2/11/2009	24						

Imagen 2.Diagrama Gantt Final

El análisis de costes del proyecto es tal y como se observa en la tabla 1. Usaremos como entorno de desarrollo Microsoft Visual Studio. La licencia de Matlab es necesaria para realizar el entrenamiento del clasificador.

Concepto	**Cantidad**	**Precio (€)**	**Precio final (€)**
Licencia Matlab Student Version	1	69.60	69.60
Licencia Microsoft Visual Studio 6.0	1	1.171,37	1.171,37
Horas de trabajo	270	20	5400
Total			6640,97

Tabla 1.Relación de gastos del proyecto

2.1.2. Casos de uso

Se distingue cuatro casos de uso, mostrados en la imagen 3, y explicados a continuación.

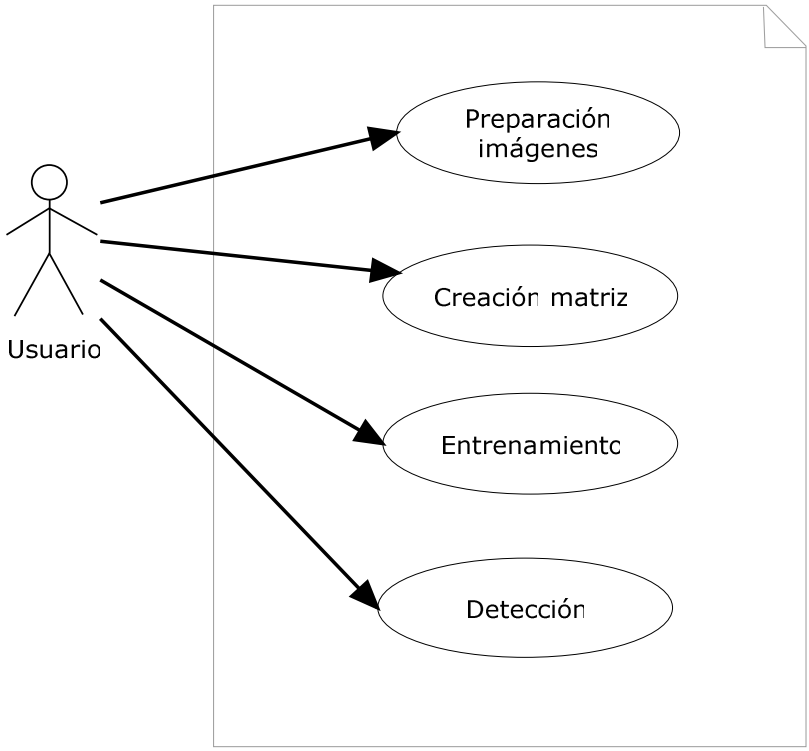


Imagen 3. Diagrama de casos de uso

Caso de uso 1

1. Preparación de las imágenes del clasificador

1.1. Descripción

En este caso de uso se preparan las imágenes para que estén en el formato adecuado y ser utilizadas por el clasificador.

2. Flujo de Eventos

2.1. Flujo Básico

1. Introducir en una carpeta todas las imágenes que se utilizarán como positivos para el entrenamiento del clasificador, y en otra carpeta las imágenes que se usarán como negativos.
2. Aplicar el programa PreparaImagen.exe, que realiza la operación Canny a cada una de las imágenes. Una vez finalizado, modificar las imágenes para que tengan 24 bits de profundidad.

Caso de uso 2

1. Creación de la matriz de positivos

1.1. Descripción

En este caso de uso se crea una matriz a partir de las imágenes positivas, que será utilizada para el entrenamiento del clasificador.

2. Flujo de Eventos

2.1. Flujo Básico

1. Introducir en el código de DemotMatlabEngine.cpp la dirección de la carpeta que contiene los positivos. Indicar el número de círculos y sectores que usará el descriptor, así como el nombre con el que se desea que se guarden los datos.
2. Ejecutar el código.

3. Precondiciones

3.1. Haber preparado las imágenes mediante el caso de uso 1.

3.2. Incluir en el path de matlab la carpeta que contiene las imágenes positivas.

3.3. Disponer de la librería Descriptor_Mex.dll.

Caso de uso 3

1. Entrenamiento de la cascada

1.1. Descripción

En este caso de uso se obtiene un clasificador a partir de la matriz con la información sobre los positivos.

2. Flujo de Eventos

2.1. Flujo Básico

1. Abrir Matlab y cargar la matriz.
2. Ejecutar el entrenador.
3. Guardar el resultado del entrenador en un archivo .xml.

3. Precondiciones

3.1. Haber preparado las imágenes mediante el caso de uso 2.

3.2. Disponer de la librería Descriptor_Mex.dll.

Caso de uso 4

1. Detección

1.1. Descripción

En este caso de uso se detectan los objetos que pueden aparecer en una imagen a partir del clasificador entrenado en el caso de uso 3.

2. Flujo de Eventos

2.1. Flujo Básico

1. Introducir los valores y archivos que se utilizarán para la detección.
2. Carga de la imagen a analizar.
3. Carga de la cascada en memoria.
4. Recorrido de la imagen en busca de los objetos.
5. Los resultados se filtran para eliminar posibles falsos positivos.
6. El resultado del filtrado del paso 5 se guarda en un archivo y es mostrado.

2.2 Flujo alternativo

2.2.1. En el paso 2.

La imagen indicada no existe, el programa finaliza.

2.2.2. En el paso 3.

El archivo con la cascada no existe, el programa finaliza.

3. Precondiciones

3.1. Disponer de la cascada entrenada en el caso de uso 3.

2.2. Diseño e implementación

2.2.1. Algoritmos

El software esta definido por los siguientes pseudocódigos. Los algoritmos de detección y entrenamiento están detallados a continuación.

Pseudocódigo 1: software de detección

Parte principal del código. Se encarga de gestionar el recorrido de la imagen y aplicar el método de detección.

```
1 Carga de la imagen a analizar y de la imagen en la que se guardan los datos.
2 Carga del archivo del clasificador.
3 Procesar la imagen mediante proyección para reducir el tamaño si así se
4 desea.
5 Preparación de la imagen para ser analizada por el detector.
6 Desde el tamaño mínimo de la ventana de detección hasta el máximo,
7 incrementándose en el factor indicado.
8       Desde el punto inicial hasta el tamaño de la imagen menos el de la
9       ventana en el eje y, incrementándose en el factor indicado.
10              Desde el punto inicial hasta el tamaño de la imagen menos el de
11              la ventana en el eje x, incrementándose en el factor indicado.
12              Aplicar el detector a la ventana.
13                      Si la ventana es un positivo, marcarla en la imagen de
14                      salida y guardar los valores en la lista de positivos.
15              Finalizar el recorrido en x.
16      Finalizar el recorrido en y.
17 Finalizar el incremento del tamaño de la ventana.
18 Calcular intersecciones de positivos.
19 Mostrar resultado.
```

Pseudocódigo 2: Carga del clasificador (línea 2 del pseudocódigo 1)

El pseudocódigo de carga del clasificador detalla el proceso del método que realiza la carga del archivo que contiene el clasificador en la estructura para su posterior utilización.

```
1 Apertura del archivo.
2 Mientras el archivo tenga líneas.
3       Si la línea indica inicio de nivel.
4               Cargar valores de nivel
5               Mientras la línea no indique final de nivel.
6                       Cargar datos de características.
7       Fin de nivel.
8 Fin del archivo.
```

Pseudocódigo 3: Proyección (línea 3 del pseudocódigo 1)

El pseudocódigo de carga del clasificador detalla el proceso del método que utiliza el algoritmo de proyección para reducir el tamaño de la imagen y acelerar el tiempo de proceso de la aplicación.

```
1 Desde el inicio de la imagen hasta el final en el eje x.
2       Desde el inicio de la imagen hasta el final en el eje y.
3               Si el píxel no es blanco, se incrementa la suma en 1.
4       Si la suma es mayor de 1.
5               Si el punto menos el margen no esta fuera de la imagen,
6               marcamos como margen lateral izquierdo el punto menos el
7               margen.
8               Salimos del bucle.
9 Desde el final de la imagen hasta el inicio en el eje x.
10      Desde el inicio de la imagen hasta el final en el eje y.
11              Si el píxel no es blanco, se incrementa la suma en 1.
12      Si la suma es mayor de 1.
13              Si el punto más el margen no esta fuera de la imagen, marcamos
14              como margen lateral derecho el punto más el margen.
15              Salimos del bucle.
16 Desde el inicio de la imagen hasta el final en el eje y.
```

```
17      Desde el inicio de la imagen hasta el final en el eje x.
18          Si el píxel no es blanco, se incrementa la suma en 1.
19      Si la suma es mayor de 1.
20          Si el punto menos el margen no esta fuera de la imagen,
21          marcamos como margen superior el punto menos el margen.
22          Salimos del bucle.
23 Desde el final de la imagen hasta el inicio en el eje y.
24      Desde el inicio de la imagen hasta el final en el eje x.
25          Si el píxel no es blanco, se incrementa la suma en 1.
26      Si la suma es mayor de 1.
27          Si el punto más el margen no esta fuera de la imagen, marcamos
28          como margen lateral inferior el punto más el margen.
29          Salimos del bucle.
30 Si algún margen no es el original.
31      Copiamos la región indicada en una imagen nueva.
32      Retornamos la imagen nueva.
33 Retornamos la imagen original.
```

Pseudocódigo 4: Detector (línea 12 del pseudocódigo 1)

El pseudocódigo del detecto explica el algoritmo de detección utilizado para detectar los posibles objetos dentro de las imágenes.

```
1 Obtener región a analizar de la imagen total.
2 Describir la región.
3 Mientras queden niveles del clasificador.
4       Mientras queden características del nivel.
5           Si la polaridad es -1, y el valor de la región indicada por la
6           característica es mayor que el threshold de la característica, se
7           incrementa la suma en el alpha de la característica.
8           Si la polaridad es 1, y el valor de la región indicada por la
9           característica es menor que el threshold de la característica, se
10          incrementa la suma en el alpha de la característica.
11      Fin de las características.
12      Si la suma es menor que el threshold del nivel.
```

```
13          Salir del detector indicando no detectado.
14 Fin de los niveles.
15 Salir del detector indicando detectado.
```

Pseudocódigo 5: Cálculo de intersecciones (línea 18 del pseudocódigo 1)

El pseudocódigo de cálculo de intersecciones detalla el proceso del método que reduce filtra la lista de positivos para eliminar posibles falsos positivos.

```
1 Seleccionamos el primer positivo.
2 Mientras el positivo actual no sea el último.
3       Mientras queden positivos en la lista de positivos.
4           Si no ha sido utilizado y tiene el área mínima indicada.
5               Incrementamos el contador de intersecciones.
6               Guardamos la acumulación de los puntos de las esquinas.
7               Marcamos el positivo como utilizado.
8       Fin de la lista de positivos.
9       Si hay el número mínimo de intersecciones.
10          Marcamos el área delimitada por la media de las esquinas en la
11          imagen.
12      Reiniciamos los contadores
13      Seleccionamos el siguiente positivo.
14 Fin de los positivos.
15 Retornamos la imagen.
```

Algoritmo de aprendizaje del clasificador

El clasificador está basado en una cascada de clasificadores binarios entrenados mediante el algoritmo de Adaboost descrito a continuación.

- Dadas las imágenes de ejemplo $(x_1, y_1), \ldots, (x_n, y_n)$ donde $y_i = 0, 1$ para los ejemplos negativos y positivos respectivamente.
- Inicializamos los pesos $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ para $y_i = 0, 1$ respectivamente, Donde *m* y *l* son el número de negativos y positivos respectivamente.
- Para *t=1,…,T*:
  1. Normalizar los pesos,

$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$

  De manera que w_t es una distribución de probabilidad.
  2. Para cada característica*, j,* entrenar un clasificador h_j que esta restringido a usar una sola característica. El error es evaluado con respecto a $w_t, \epsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$.
  3. Escoger el clasificador, h_j, con el menor error ϵ_t.
  4. Actualizar los pesos:

$$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$

  Donde $e_i = 0$ si el ejemplo *x*, se ha clasificado correctamente, $e_i = 1$ en caso contrario, y $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- El clasificador fuerte final es:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{en caso contrario} \end{cases}$$

  donde $\alpha_t = \log \frac{1}{\beta_t}$

Una vez obtenemos un clasificador, entrenamos los siguientes teniendo en cuenta los casos negativos no descartados por los clasificadores anteriores.

2.2.2. Herramientas utilizadas y detalles de implementación

El código del detector ha sido realizado en C++, usando como entorno de desarrollo Microsoft Visual C++ 6.0. Se ha utilizado una versión adaptada, con el fin de reducir el tiempo de procesado, del descriptor del proyectista Mario Guitart Matamoros utilizado para la descripción de las regiones de la ventana del detector.

Para el entrenamiento de la cascada de clasificadores se ha utilizado un clasificador basado en el algoritmo Adaboost implementando en Matlab. Se ha modificado el código para que utilizase como descriptor el mismo que el usado por el detector. Para ello se creó un MEX-file conteniendo el código. El resultado del entrenamiento se guarda en un archivo .xml para permitir el acceso de los datos al código elaborado en C++.

2.2.3. Estructuras

Se han definido las siguientes estructuras para cargar los datos del clasificador y guardar los positivos localizados.

Feature

```
typedef struct feature_t *featurePtr;
typedef struct feature_t{
    int numero;
    double thrf;
    int polarity;
    double alpha;
    featurePtr siguiente;
}feature;
```

Esta estructura contiene la información relativa a una característica definida por el clasificador. Tiene los siguientes valores:

- numero: número de la característica que se debe comparar.
- thrf: threshold con el que se compara la característica.
- polarity: indica el tipo de comparación ha realizar.
- alpha: cantidad que se suma si se supera la comparación.
- siguiente: puntero a la siguiente característica del nivel.

Level

```
typedef struct level_t *levelPtr;
typedef struct level_t{
    double thr;
    featurePtr primerf;
    levelPtr siguiente;
}level;
```

Esta estructura contiene la información relativa a un nivel del clasificador. Tiene los siguientes valores:

- thr: threshold del nivel.
- primerf: puntero a la primera característica del nivel.
- siguiente: puntero al siguiente nivel.

Positivo

```
typedef struct positivo_t *positivoPtr;
typedef struct positivo_t{
    int x;
    int y;
    int k;
    int usado;
    positivoPtr siguiente;
}positivo;
```

Esta estructura contiene la información relativa a los positivos localizados. Consta de los siguientes valores:

- x: posición en el eje X del vértice izquierdo de la imagen.
- y: posición en el eje y del vértice superior de la imagen.
- k: tamaño de la ventana de detección.
- usado: parámetro usado para indicar si el positivo ha sido utilizado en una intersección.
- siguiente: puntero al siguiente positivo.

2.2.4. Secuencias

Los diagramas de secuencia que se hallan a continuación, muestra el funcionamiento interno del software.

La imagen 4 muestra el diagrama de secuencia del código principal. En ella se puede observar el recorrido en los ejes que realiza el software sobre la imagen para los diversos tamaños de la ventana del descriptor. Este diagrama describe el escenario en el caso de que no se produzca ningún error.

La imagen 5 muestra la secuencia del método de la proyección, cuyo fin es reducir el tamaño de la imagen para acelerar el proceso de cálculo. En el

diagrama se detalla al flujo para calcular el margen izquierdo y derecho de la imagen. El cálculo del margen superior e inferior se realiza de manera análoga.

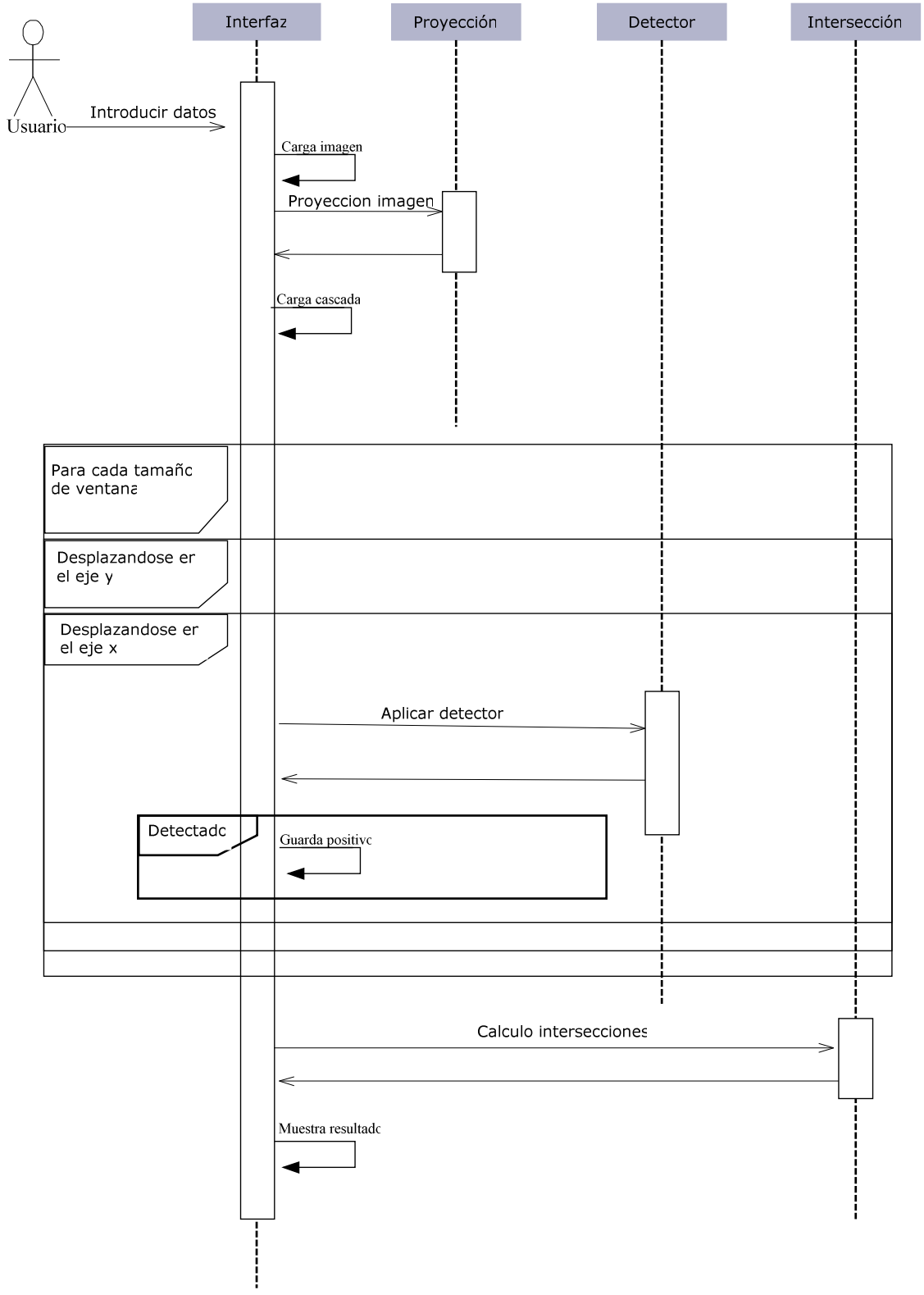


Imagen 4. Secuencia principal

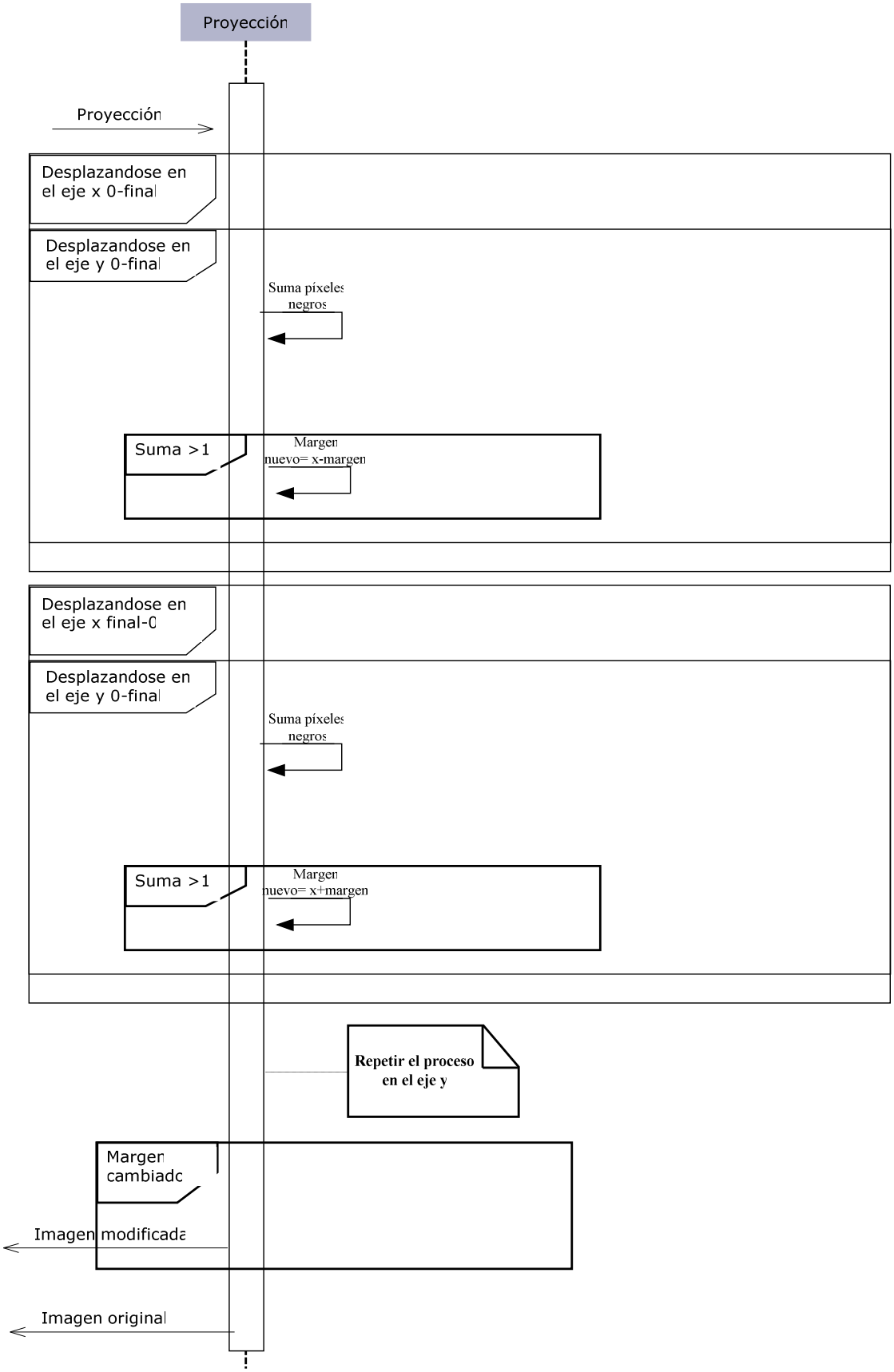


Imagen 5. Diagrama de secuencia del método de proyección

La imagen 6 muestra el funcionamiento interno del detector. El diagrama muestra los dos posibles retornos del método. Si en algún nivel no se supera el threshold, el método retorna no detectado. Si se superan todos los niveles el método retorna detectado.

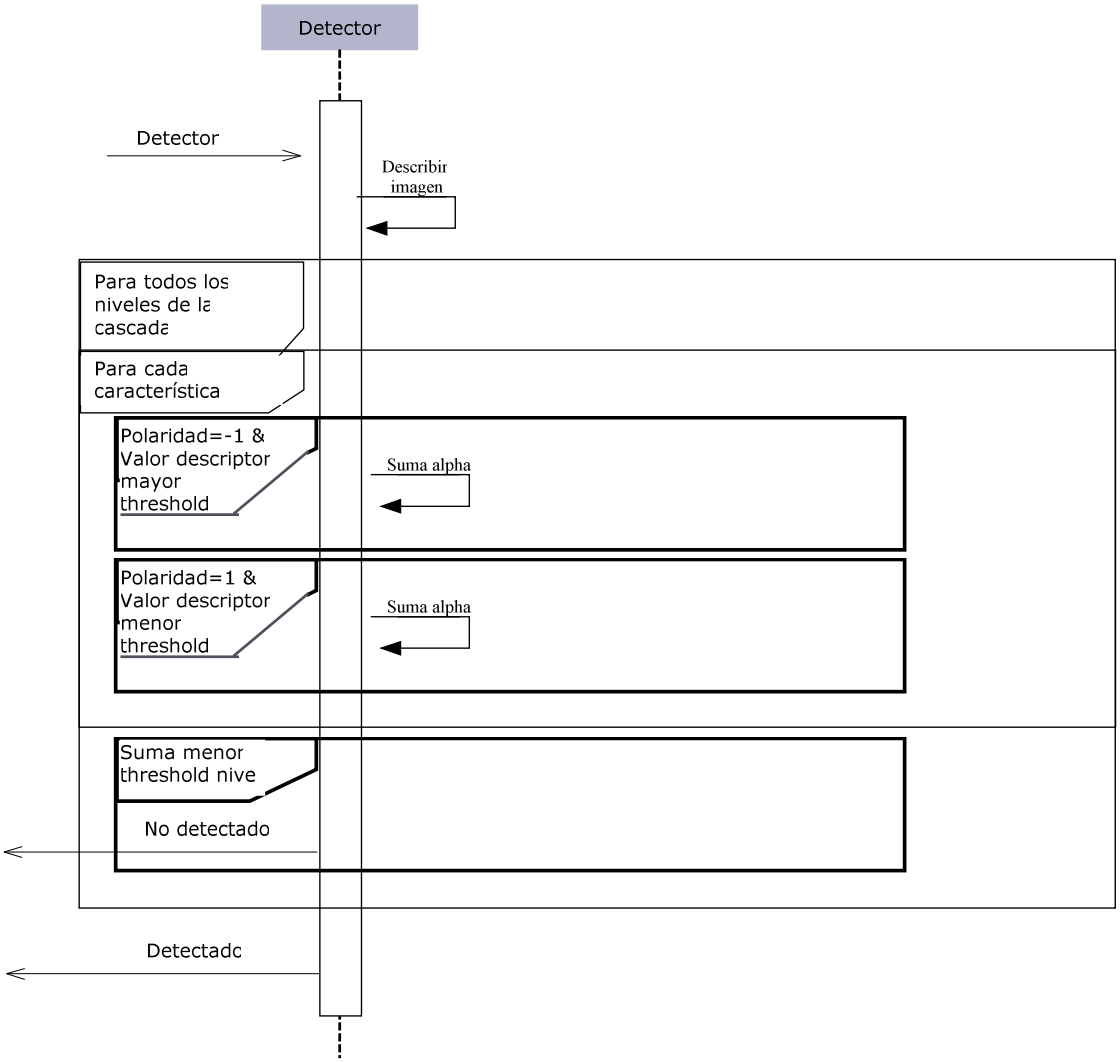


Imagen 6. Diagrama de secuencia del detector

La imagen 7 muestra como se realiza el cálculo de las intersecciones, para eliminar posibles falsos positivos y mejorar la visualización de los resultados. El diagrama muestra el escenario en el caso de que halla como mínimo dos positivos, ya que si no resulta imposible calcular la intersección.

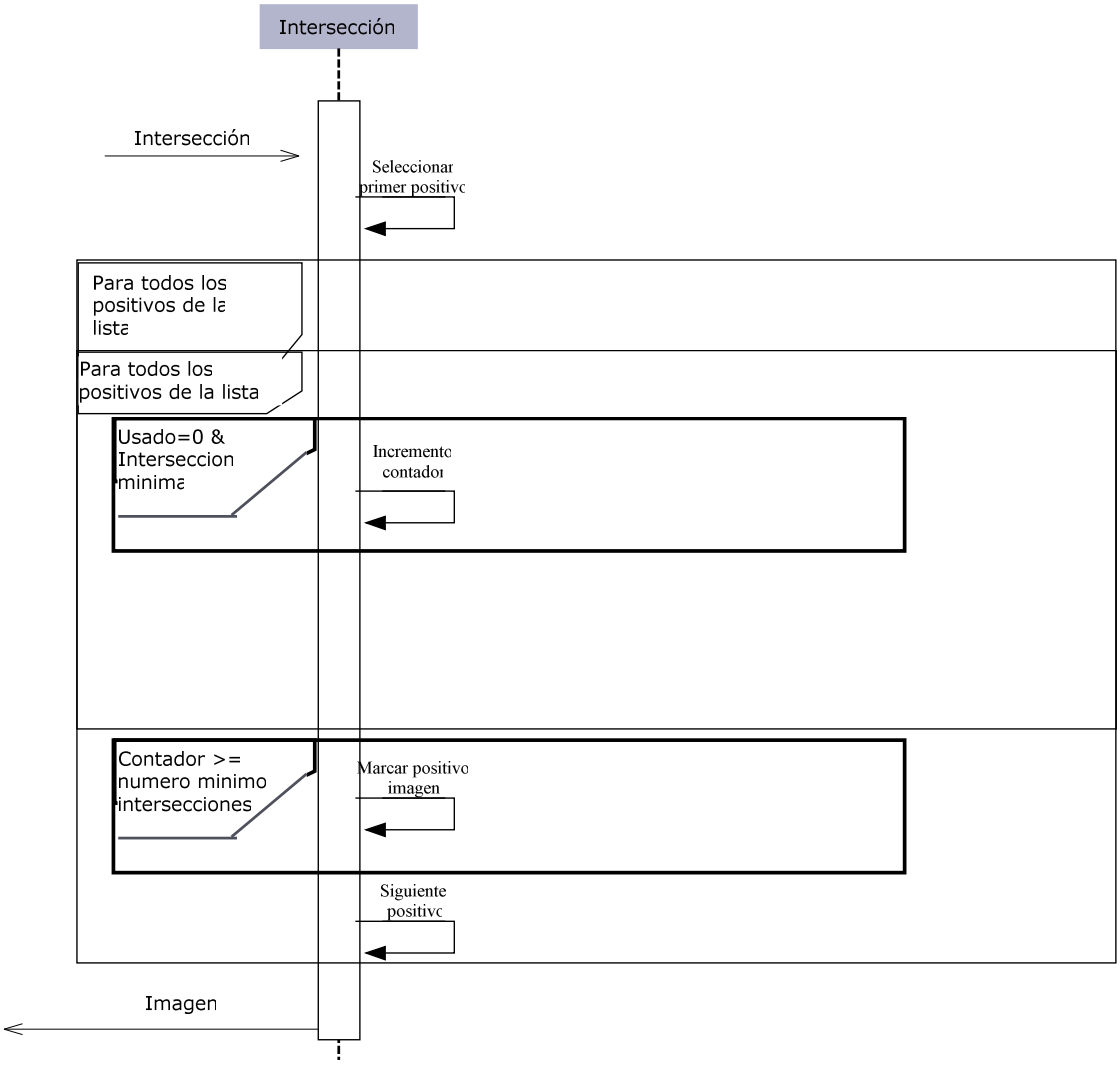


Imagen 7. Diagrama de secuencia del calculo de Intersecciones

3. Resultados

En este apartado se expondrán los datos de los métodos, los experimentos a realizar para evaluar el software, y los resultados obtenidos.

3.1. Datos

En esta sección se describen los datos que son necesarios para poder realizar los procesos de entrenamiento y detección.

3.1.1. Entrenamiento

Para el entrenamiento del clasificador en Matlab necesitaremos los siguientes datos:

- Una matriz que contiene en cada fila una imagen positiva a la cual se le ha aplicado la función Canny y ha sido descrita por el descriptor BSM de Mario Guitart Matamoros.
- Número de negativos que se quieren utilizar para entrenar cada nivel de la cascada.
- Número de niveles que se desean entrenar.
- Porcentaje de acierto mínimo para pasar al siguiente nivel. El clasificador de ese nivel debe clasificar como positivo el porcentaje indicado.
- Porcentaje de error máximo para pasar al siguiente nivel. El porcentaje máximo de negativos que el clasificador puede clasificar como positivos en un nivel.
- Una carpeta que contiene las imágenes negativas, a las cuales se les ha aplicado la función Canny. Es necesario que las imágenes negativas no contengan ninguna de las imágenes positivas.
- Formato en el que se encuentran las imágenes negativas.
- Tamaño mínimo en el eje y de los negativos a utilizar. El entrenador del clasificador selecciona regiones al azar de las imágenes negativas para usarlas. Esta región debe tener como mínimo el tamaño indicado, si no se descartará y se seleccionara otra.

- Tamaño mínimo en el eje x de los negativos a utilizar. El mismo caso que en el eje Y.
- Número de iteraciones máximas que se usaran por nivel. El entrenador intentara cumplir las condiciones de porcentaje de acierto mínimo y de error máximo en un número concreto de iteraciones.
- Número de círculos que utilizara el descriptor. Debe ser el mismo que se uso para describir a los positivos de la matriz.
- Número de sectores que utilizara el descriptor. Debe ser el mismo que se usó para describir a los positivos de la matriz.

3.1.2. Detección

Para la detección de objetos en imágenes tendremos que tener los siguientes datos:

- Una imagen de entrada. Ésta es la imagen que se desea analizar y debe de estar en formato jpg.
- Un nombre para la imagen de salida. La imagen debe de tener alguno de los formatos aceptados por la librería OpenCV. En nuestro caso para mostrarlos por la interfaz utilizaremos el formato bmp.
- Un archivo con la cascada de clasificadores entrenada. Debemos saber que número de círculos y sectores se ha utilizado en el descriptor a la hora de entrenarla.
- Un número máximo de niveles a cargar del clasificador. Dejamos que se pueda seleccionar este parámetro para poder detectar casos de sobreentrenamiento en el clasificador y corregirlos sin necesidad de entrenar otra cascada.
- Un tamaño mínimo de la ventana detectora. Este tamaño debe ser aproximadamente el cuadrado más pequeño que pueda contener el objeto a detectar.
- Un tamaño máximo de la ventana detectora. Éste será mayor que el mínimo, y nos indicará hasta que tamaño debemos buscar objetos.
- Como se irá incrementando el tamaño de la ventana detectora, y el tamaño en el cual se incrementa. Puede ser sumando píxeles al tamaño anterior o multiplicándolo por un factor.

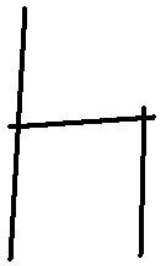
- Como se desplazará la ventana detectora y el factor de desplazamiento. Como en el caso del incremento, el desplazamiento por cada tamaño de la ventana puede ser siempre el mismo, aumentar multiplicándolo por un factor, o que mantenga siempre la misma relación de tamaño respecto a la primera ventana detectora.
- Número de intersecciones y tamaño de éstas sobre un objeto para considerarlo positivo. Nos indica el número de detecciones que ha de darse sobre una zona para considerar que contiene una imagen positiva.
- La operación de preproceso que se desea aplicar a la imagen. Podemos elegir no realizar ningún tipo de preproceso o realizar proyección, en cuyo caso deberemos indicar un margen que se añadirá a la imagen una vez se haya realizado la operación.

3.2 Experimentos

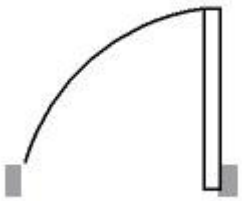
En este apartado se muestran los experimentos realizados y los resultados obtenidos.

3.2.1 Entrenamiento de clasificadores

Para el testeo del software se entrenaron dos cascadas de clasificadores diferentes. La primera preparada para detectar sillas vistas desde el lateral, como la que se muestra de ejemplo en la imagen 8. La otra se ha entrenado para detectar puertas en planos, como la de la imagen 9.



8. Silla utilizada para entrenar el clasificador de sillas



9. Puerta utilizada para entrenar el clasificador de puertas

3.2.1.1. Clasificador de sillas

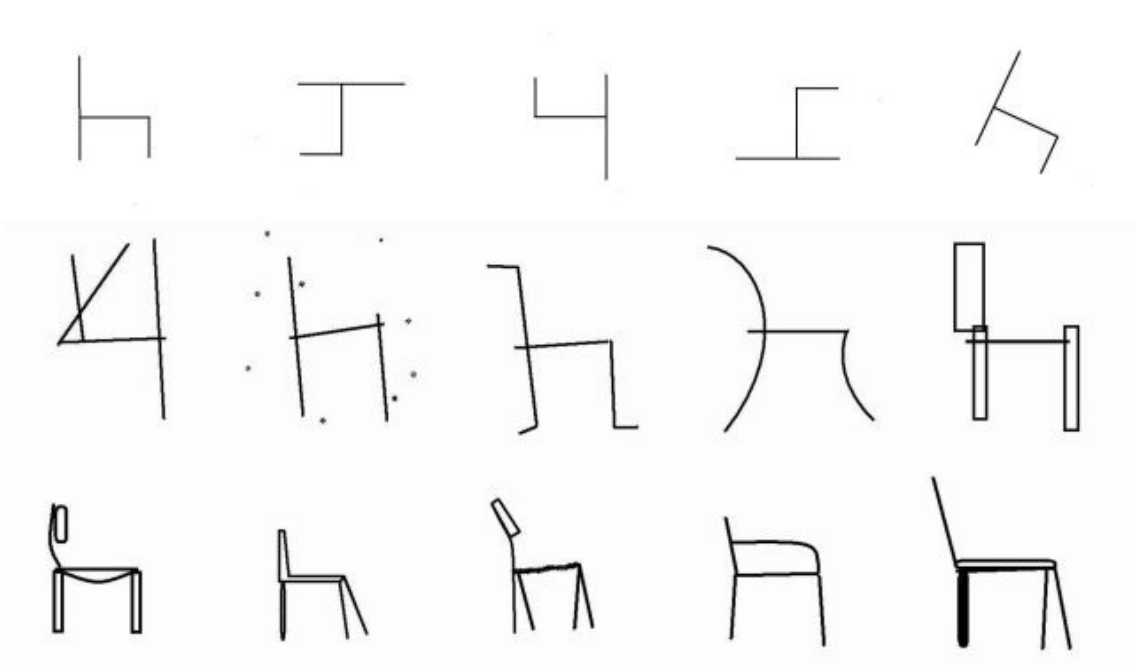
Para el entrenamiento del clasificador de sillas se han utilizado los siguientes datos:

- 40 imágenes positivas con la operación Canny aplicada, en formato jpg de 24 bits de profundidad. Las imágenes contienen diferentes tipos de

sillas en diferentes posiciones como se observa en la imagen 10. Estas imágenes ya han sido descritas por el descriptor, utilizando 6 círculos y 6 sectores.

- 100 negativos por nivel de cascada.
- Un máximo de 10 niveles.
- El porcentaje de acierto mínimo es del 99% y el de error máximo del 20%.
- Unas 5000 imágenes negativas, preparadas de la misma manera que las positivas. Ninguna de estas imágenes contiene alguna positiva.
- Como tamaño mínimo y el eje X e Y 20 píxeles.
- El número de iteraciones máximo es de 250.
- Se utilizan 6 círculos y 6 sectores, coincidiendo con los valores usados para describir los positivos.

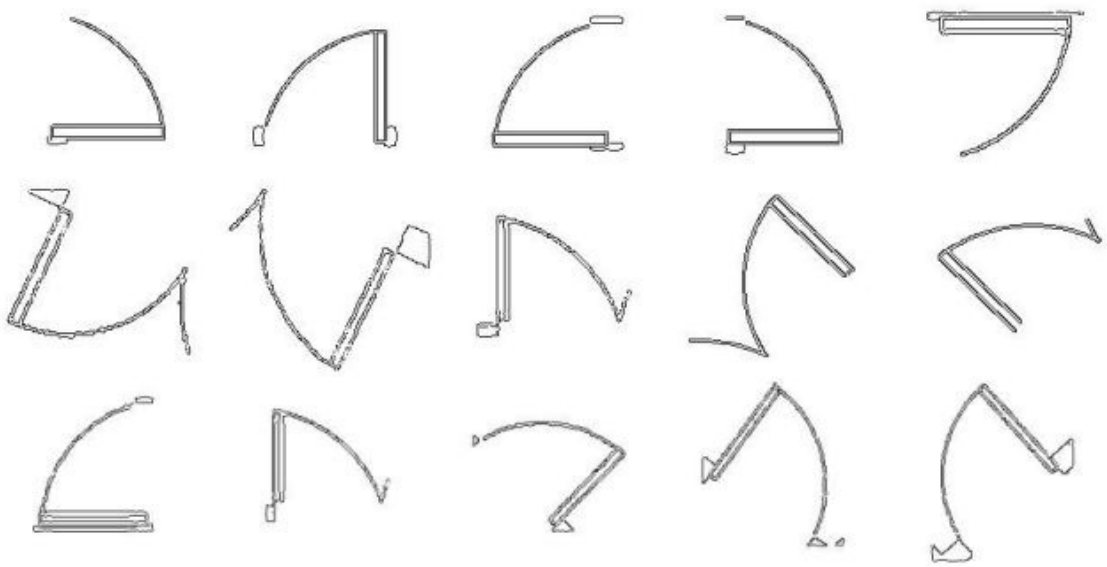
Utilizando estos valores hemos obtenido una cascada de clasificadores de 5 niveles, que utilizaremos posteriormente en el proceso de detección.



10. Modelos de sillas utilizadas para entrenar el clasificador

3.2.2.2. Clasificador de puertas

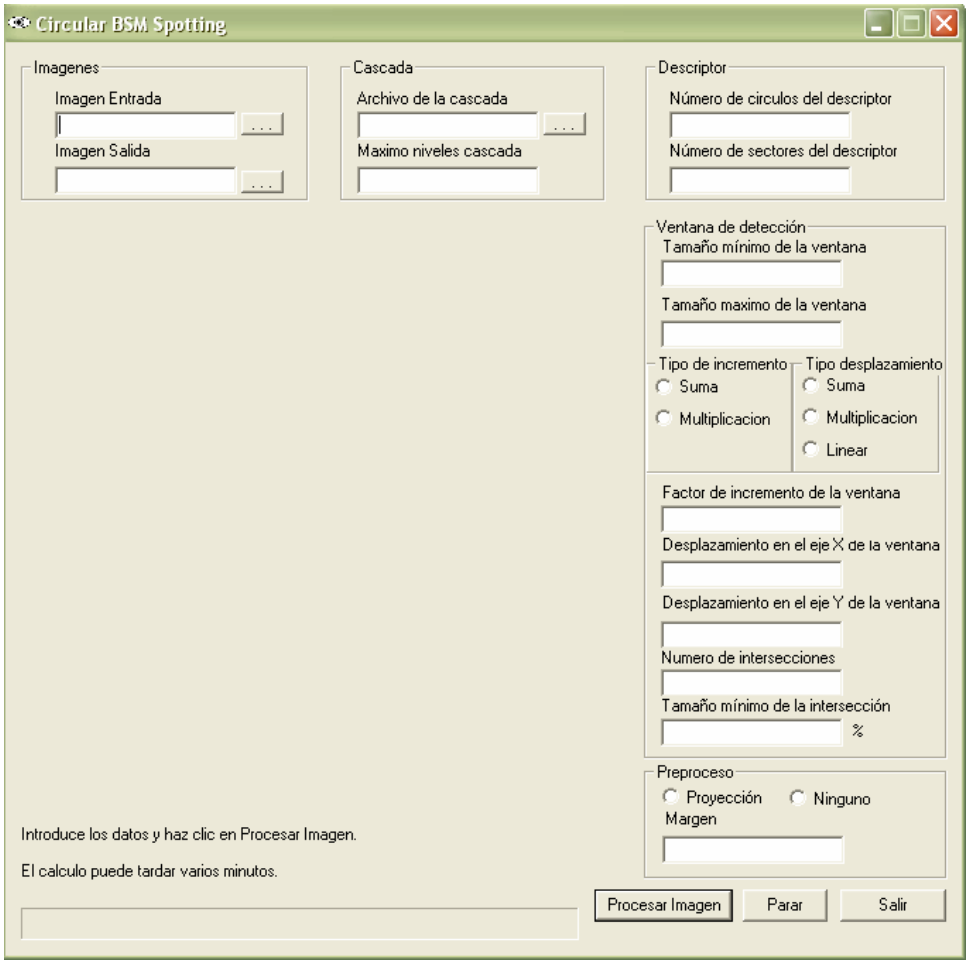
Para entrenar el clasificador de puertas hemos usado los mismos valores que en el clasificador de sillas, exceptuando los positivos. En este caso los positivos han sido 41 puertas. Ya que sólo se disponen de un par de símbolos diferentes para indicar los modelos de puertas, se han incluido en el conjunto de entrenamiento diferentes orientaciones y tamaños del mismo modelo (Imagen 11). Se ha obtenido una cascada de clasificadores de 5 niveles.



11. Modelos de puertas, con el método Canny aplicado, utilizadas para entrenar el clasificador

3.2.2 Detección

Para realizar estos experimentos se han utilizado las cascadas entrenadas para la detección de los diferentes objetos, y la Interfaz para la introducción de los datos de detección (Imagen 12). En los experimentos aquí mostrados se ha utilizado el tipo de incremento y de desplazamiento suma, aunque son los métodos que mas tiempo requieren, también son los que nos ofrecen una mejor detección.



12. Interfaz programada para la introducción de los datos necesarios para el funcionamiento del detector

3.2.2.1 Detección de sillas

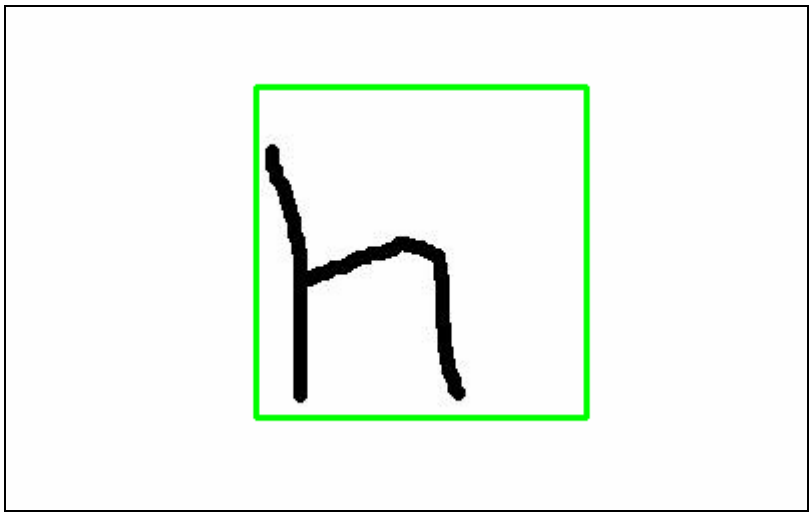
Se han realizado pruebas con diferentes situaciones (detección de una única silla en la imagen, varias sillas, una silla y otro objeto que no es una silla) aplicando diferentes valores de detección (modificando la aplicación de preproceso y el tipo de desplazamiento). En todos los experimentos se ha utilizado la cascada detectora de sillas, los parámetros de 6 círculos y 6 sectores para el descriptor y todos los niveles de cascada disponibles, en nuestro caso 5.

Estos son algunos de los resultados obtenidos.

Detección de una silla

En los casos de detección de un único objeto, el margen a la hora de fijar los parámetros es mayor, ya que sabemos que hay un único objeto en la imagen, y lo único que nos interesa es la posición. Para la detección obtenida en la Imagen 13 se han utilizado los siguientes valores:

- Un tamaño mínimo de ventana de 140 y máximo de 180.
- El tipo de incremento y de desplazamiento ha sido en los dos casos de suma.
- Un incremento de 10 píxeles en el tamaño de la ventana y un desplazamiento también de 10 en el eje x e y de esta.
- Dado que sólo hay un objeto no tenemos que preocuparnos de falsos positivos, por lo que sólo pedimos 1 una intersección del 40% para considerar la detección como positiva.
- No se ha utilizado ningún tipo de preproceso.



13. Detección de una única silla

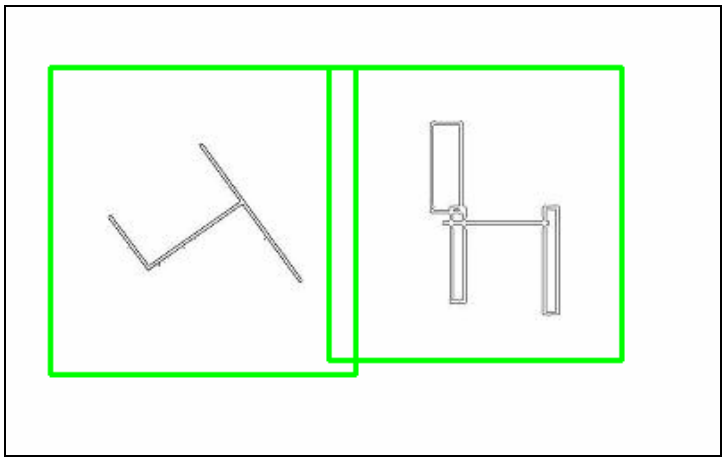
Detección de varias sillas sin preproceso en la imagen

El caso de de varias imágenes es similar al anterior, pero se ha de tener en cuenta que el tamaño de la ventana de detección ha de comprender las dimensiones de los diferentes objetos. Se han aplicado los siguientes valores:

- Un tamaño mínimo de ventana de 160 y máximo de 180.
- El tipo de incremento y de desplazamiento han sido en los dos casos de suma.

- Un incremento de 10 píxeles en el tamaño de la ventana y un desplazamiento también de 10 en el eje x e y de ésta.
- Fijamos un mínimo de 5 intersecciones del 70%.
- No se ha utilizado ningún tipo de preproceso.

  Con estos datos hemos obtenido la Imagen 14.



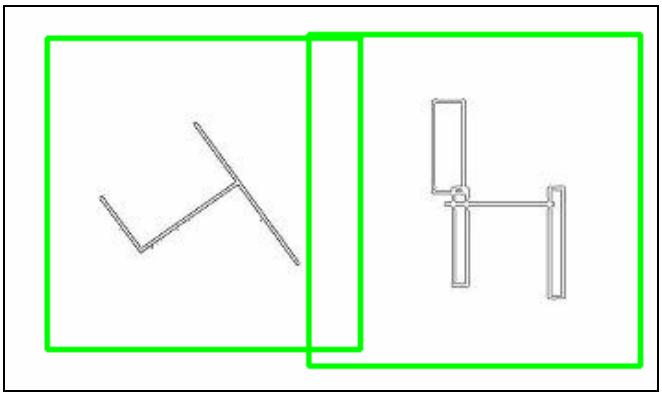
14. Detección de varias sillas en una misma imagen

Detección de varias sillas con preproceso en la imagen

Hemos utilizado la misma imagen, pero en este caso hemos seleccionado que se aplique el proceso de proyección con un margen de 50 píxeles. En el caso de aplicar preproceso, se ha de tener muy en cuenta el margen mínimo, ya que podríamos eliminar posibles detecciones al reducirlo. Los valores que hemos cambiado respecto al experimento anterior han sido:

- Se ha incrementado el tamaño máximo de la ventana de detección hasta 190.
- Se ha reducido el número de intersecciones necesarias hasta 2 y el tamaño de estas al 60%.

  El resultado obtenido ha sido la Imagen 15.

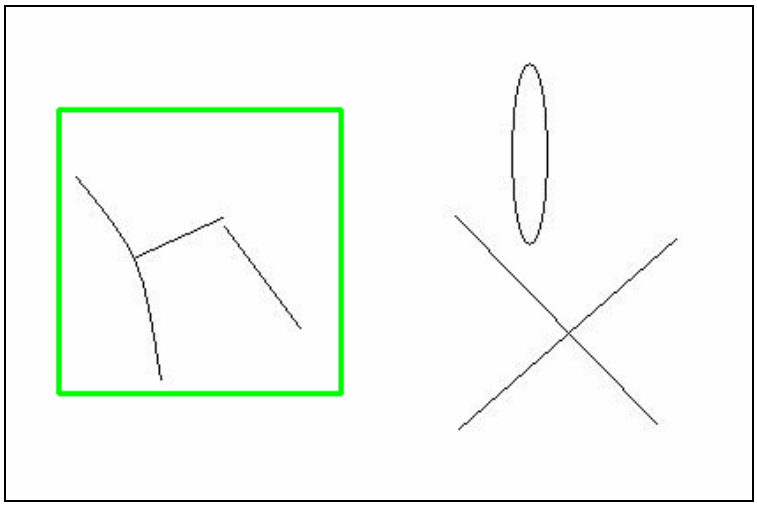


15. Detección de varias sillas en una misma imagen aplicando el preproceso de proyección

Detección de una silla en una silla con otros objetos en la imagen

Se ha probado la capacidad del detector para la detección de sillas cuando hay presentes otros objetos, y su capacidad para distinguirlos como objetos que no son silla. Se han utilizado los siguientes valores para obtener el resultado de la Imagen 16:

- Tamaño mínimo del descriptor de 150 y máximo de 180.
- El tipo de incremento y de desplazamiento ha sido en los dos casos de suma.
- Un incremento de 10 píxeles en el tamaño de la ventana y un desplazamiento también de 10 en el eje x e y de sta.
- Fijamos un mínimo de 3 intersecciones del 40%.
- No se ha utilizado ningún tipo de preproceso.



16. Detección de una silla con otro objeto en la imagen. El objeto tiene cierta similitud con una silla y fue necesario incrementar el número de intersecciones para eliminar el falso positivo

Finalmente, de un total de 10 imágenes de test analizadas, siendo estas tanto imágenes positivas utilizadas en el entrenamiento, como creadas para realizar los tests, se ha observado un 100% de detecciones. Los falsos positivos encontrados en estos tests han sido prácticamente nulos, y eliminados fácilmente mediante el incremento del número de intersecciones.

Detección de puertas en planos

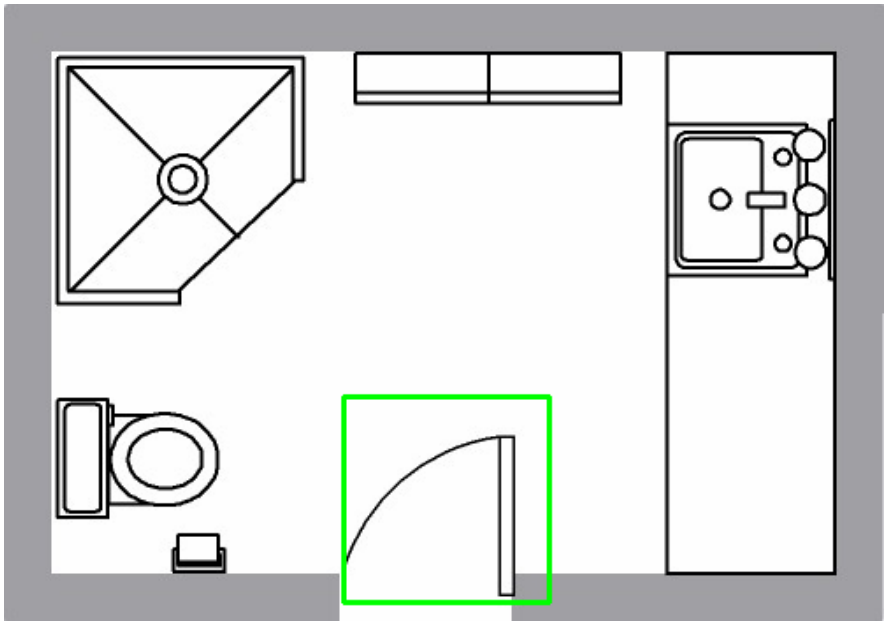
Los experimentos de detección de puertas en planos entraña la dificultad de que la puerta siempre esta unida a otro elemento del plano y de que los símbolos para cada sentido de apertura de una puerta son completamente opuestos. Con lo cual el clasificador obtenido, aunque detecta todas las puertas, también detecta una pequeña cantidad de falsos positivos.

En todos los experimentos se ha utilizado la cascada detectora de puertas, los parámetros de 6 círculos y 6 sectores para el descriptor y todos los niveles de cascada disponibles. El desplazamiento ha sido de 10 píxeles en ambos ejes, así como el incremento del tamaño de la ventana.

Algunos de los resultados obtenidos son las imágenes mostradas a continuación.

En la imagen 17 observamos la detección de una única puerta en el plano. Los valores específicos utilizados en esta detección han sido:

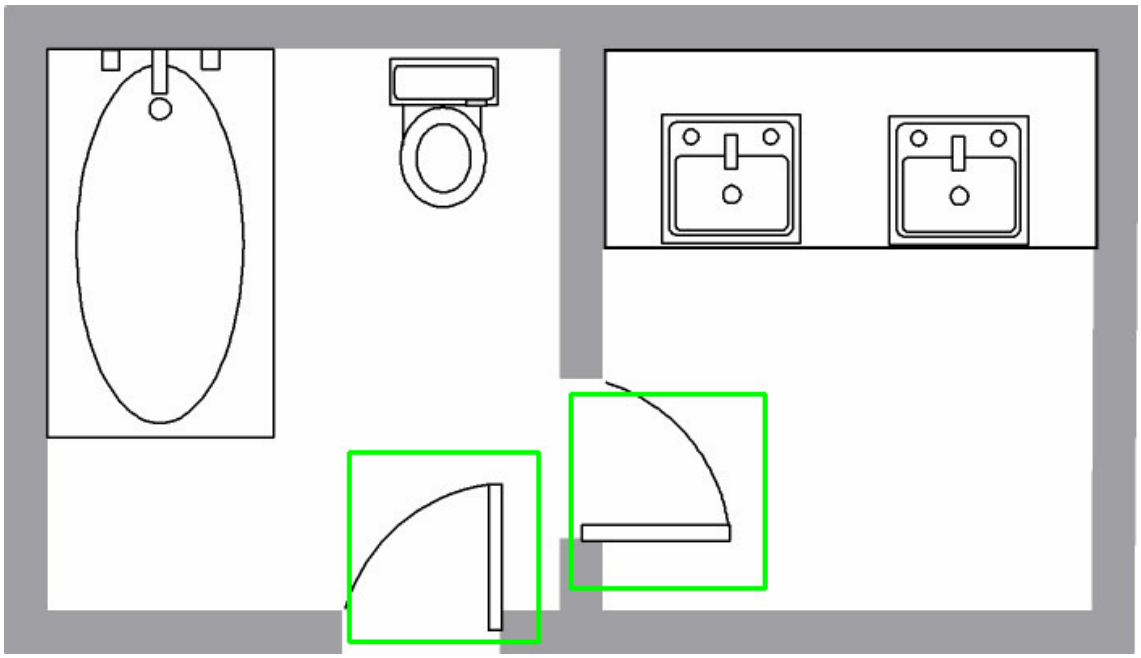
- Un mínimo de 4 intersecciones del 70%.
- El tamaño mínimo de la ventana de detección ha sido de 110 y el máximo de 130.



17. Detección de una puerta en un plano

En la imagen 18 observamos la detección de 2 puertas con diferente orientación. Esto es posible gracias al descriptor utilizado, el cual es invariante a la rotación de las imágenes. Los valores usados fueron:

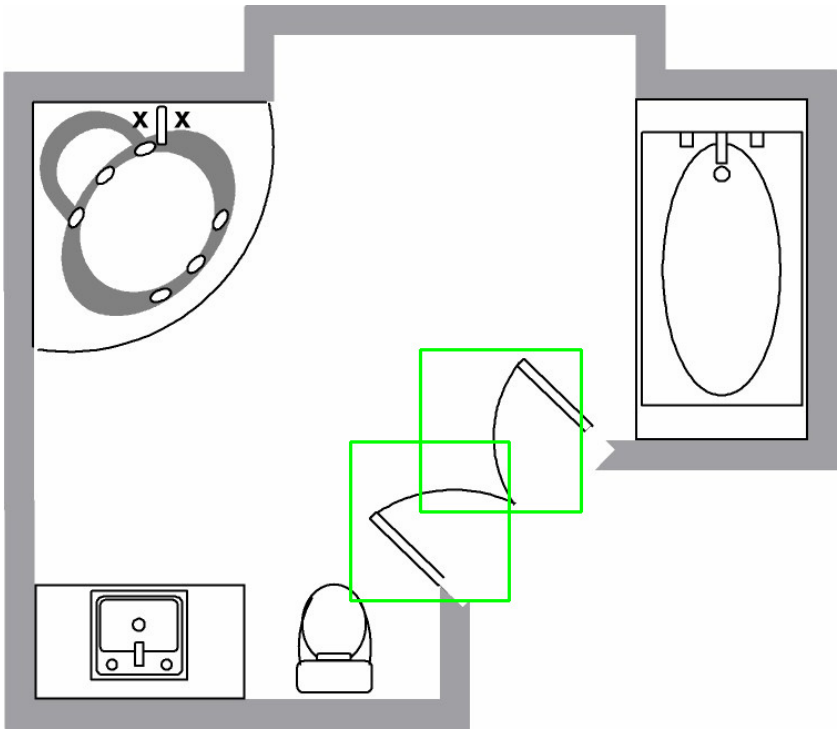
- Un mínimo de 3 intersecciones del 70%.
- El tamaño mínimo de la ventana de detección ha sido de 110 y el máximo de 130.



18. Detección de varias puertas con diferente orientación

En la imagen 19 se observa una detección parecida a la del caso anterior, pero en ésta no solo cambia la orientación de la puerta, sino también el sentido de apertura de ésta, y con esto el símbolo utilizado. Además los dos símbolos son colindantes. Los valores que obtuvieron esta detección fueron:

- Un mínimo de 2 intersecciones del 40%.
- El tamaño mínimo de la ventana de detección ha sido de 140 y el máximo de 160.



19. Detección de varias puertas con diferente sentido de apertura

En caso de los experimentos sobre planos, estos han sido realizados sobre un total de 15 imágenes. Se ha obtenido un porcentaje de localización de aproximadamente el 80%. Sin embargo es este caso el número de falsos positivos ha sido mayor, dado el gran número de elementos que componen un plano.

4. Conclusiones

En este proyecto se ha tratado el problema de la detección de objetos en imágenes mediante técnicas de Aprendizaje Estadístico y Visión por Computador.

Hemos obtenido una aplicación capaz de detectar objetos en imágenes, con una gran personalización de los valores de detección. Esta personalización permite que sea el usuario el que elija la precisión con la que desea efectuar la detección, o si por el contrario prefiere que el proceso sea más rápido. Esto se ha conseguido mediante diversas estrategias.

La primera es la capacidad de entrenar diversos clasificadores e indicarle a la aplicación cual utilizar, lo cual le permite cambiar el tipo de objeto a detectar de manera sencilla. Además, somos capaces de indicar los niveles del clasificador que se desean cargar, con lo que los casos de sobreentrenamiento se pueden subsanar sin necesidad de crear otra cascada de clasificadores.

La segunda estrategia es la inclusión de un filtro para detectar posibles falsos positivos. Este filtro, implementado mediante el cálculo de las intersecciones de los positivos detectados, permite eliminar detecciones erróneas que guarden un gran parecido con el objeto a detectar.

La última es la selección de los parámetros de desplazamiento y tamaños de la ventana detectora, con lo cual el usuario puede acotar las zonas a detectar.

Se ha incluido una opción de preproceso para acelerar el tiempo de proceso de la aplicación basada en la proyección. Gracias a esto es posible eliminar márgenes que no contienen ningún tipo de objeto, pero esta misma característica hace que el tipo de imágenes al que se puede aplicar sea limitado.

La aplicación permite la detección de objetos en imágenes de forma rápida y con necesidad de poca precisión en los parámetros de detección en los casos de imágenes con objetos aislados. En los casos con diversos objetos en la imagen los parámetros necesarios para una correcta detección han de ser más precisos. En definitiva, la precisión del detector depende en gran medida

de los parámetros de detección y del hecho de que la cascada de clasificadores haya sido entrenada correctamente.

El tiempo de cálculo de la aplicación crece de manera exponencial al disminuir el desplazamiento de la ventana o los márgenes de tamaño de ésta, así como por el tamaño original de la imagen. Esto podría solucionarse aplicando preprocesos a la imagen con el fin de eliminar zonas no deseadas o para reducir la escala de la imagen.

Para conseguir una mejora de tiempo mediante el código habría que conseguir una mejora de rendimiento del descriptor de imágenes, ya que el mayor tiempo de proceso resulta de la operación de obtención de las características de la imagen mediante el descriptor, siendo el tiempo de aplicación del método de detección prácticamente nulo.

En resumen, se ha obtenido una buena aplicación detectora, con un gran margen para mejorar en el tiempo de procesado pero con un porcentaje de detección muy elevado.

5. Bibliografía

[1] *computervision.wikia.com/wiki/Fisher%27s_linear_discriminant*

[2] *en.wikipedia.org/wiki/Nearest_neighbor_search*

[3] *en.wikipedia.org/wiki/Support_vector_machine*

[4] *en.wikipedia.org/wiki/AdaBoost*

[5].*computervision.wikia.com/wiki/Classifiers_and_Discriminants*

[6] *www.mts.jhu.edu/~priebe/COURSES/FALL2003/550.730/jdm00.pdf*

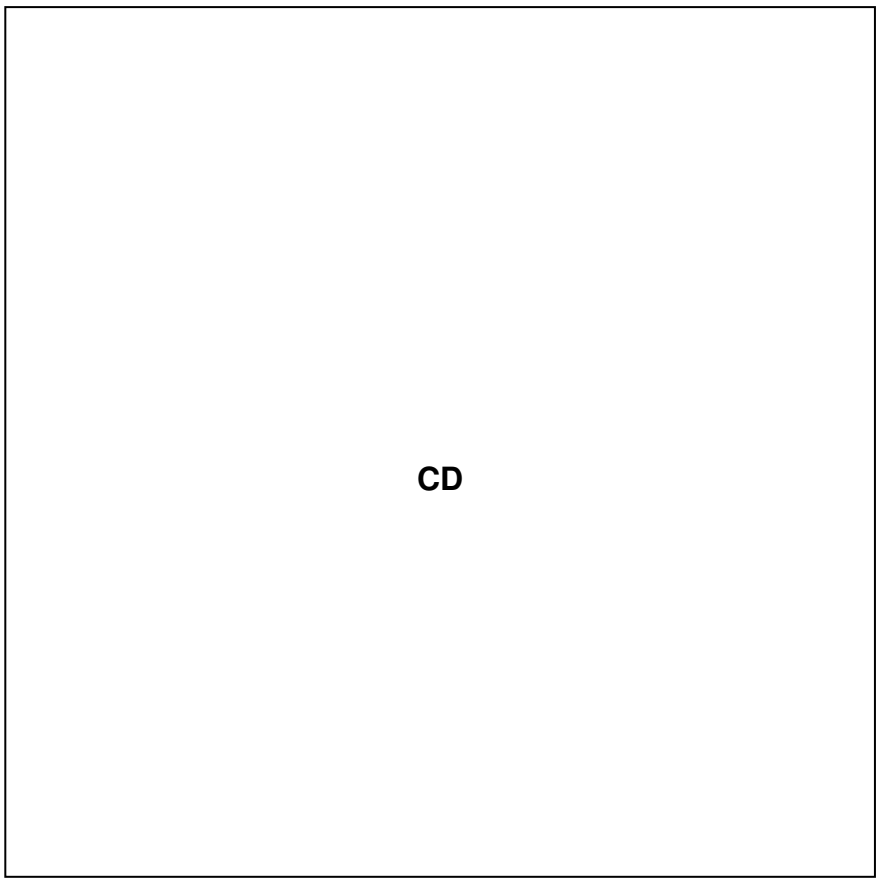
[7]Paul Viola & Michael Jones, "Robust Real-time Object Detection" en, *Second International Workshop On Statistical And Computational Theories Of Vision – Modeling, Learning, Computing, And Sampling, 2001.*

6. Anexos

6.1 CD

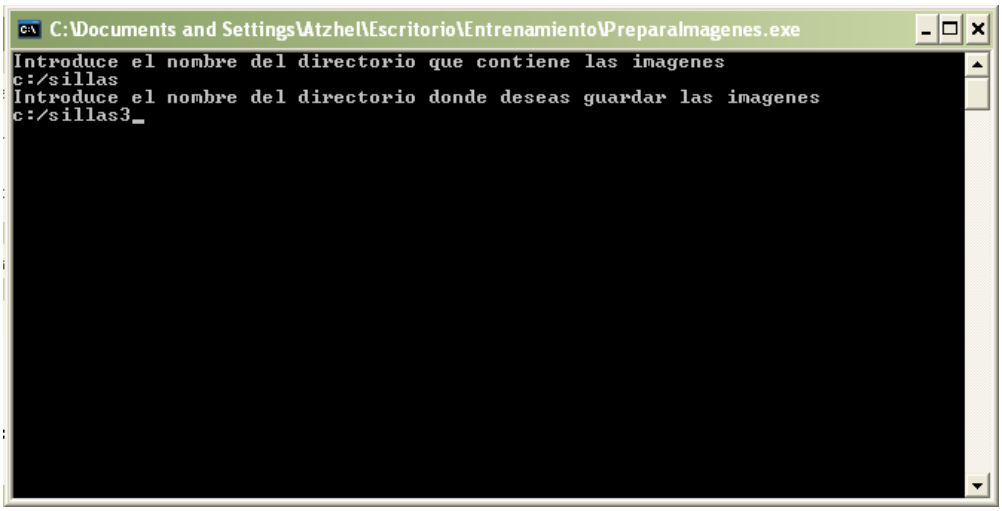
El CD adjunto contiene las siguientes carpetas:

- Código fuente. Códigos del programa.
    - Interfaz. Código de la Interfaz.
    - Principal. Código que se ha usado para generar la librería usada por la interfaz.
- Interfaz. Ejecutables y cascadas de clasificadores para probar la aplicación.
- Entrenamiento. Ejecutables y archivos necesarios para entrenar cascadas de clasificadores.
- Documentos. Manuales de usuario y memoria del proyecto.

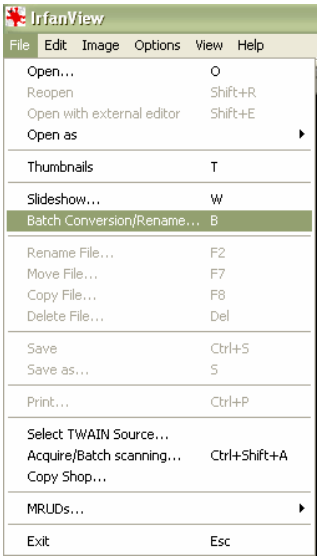


6.2. Manual de usuario del proceso de entrenamiento

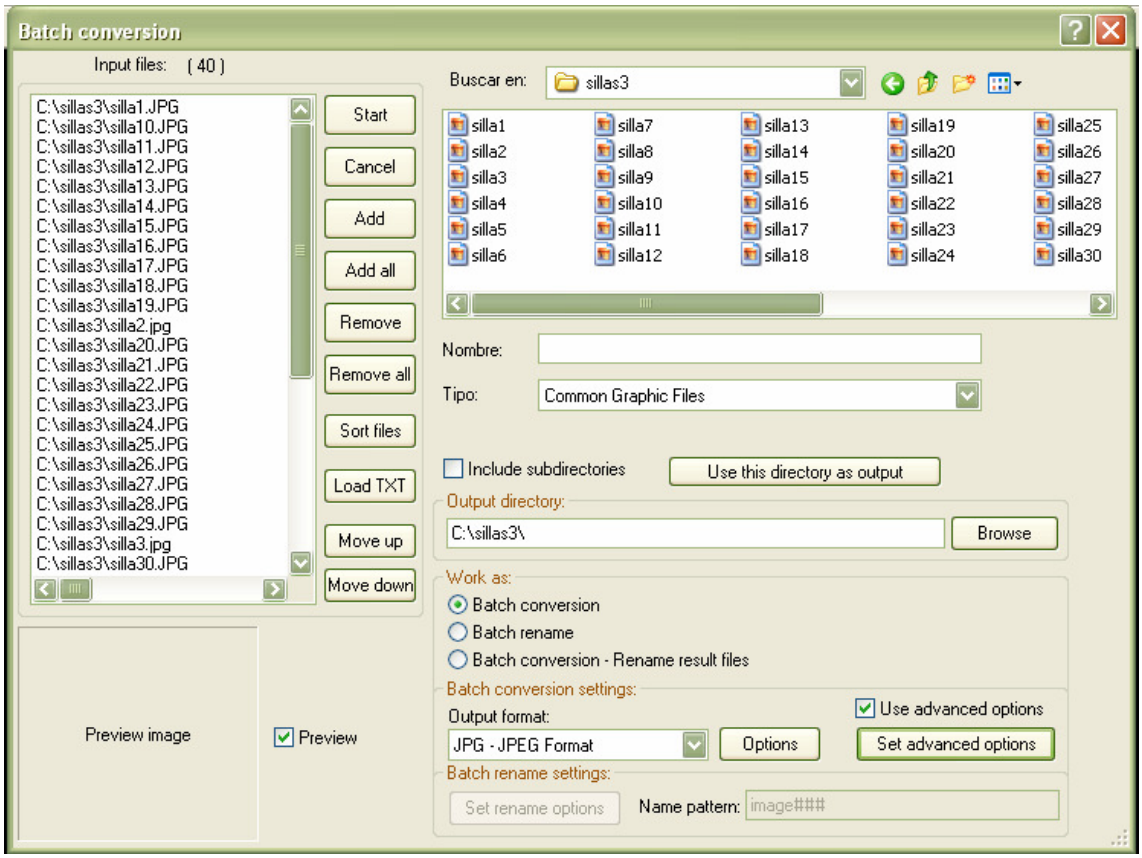
Para empezar, colocamos todas las imágenes que vamos a usar como positivos en una única carpeta. Todos los positivos deben tener el mismo nombre y un número para diferenciarlos, por ejemplo silla1.jpg, silla2.jpg, etc. Ejecutamos el programa PreparaImagenes.exe, y le indicamos la ruta de la carpeta que contiene los positivos y la ruta donde deseamos que se guarden los resultados. A causa de limitaciones en el código, el nombre de la ruta no debe contener espacios. Como precaución, seleccionaremos como destino una carpeta diferente de la original.



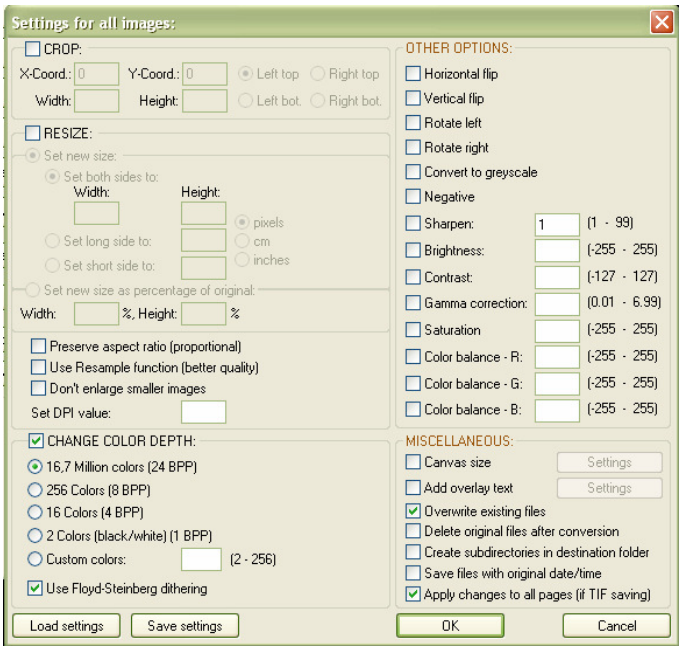
A continuación, necesitamos que las imágenes tengan una profundidad de 24 bits, sin embargo el paso anterior nos da como resultado imágenes de 8. Aplicaremos un conversor a todas las imágenes, como por ejemplo IrfanView. Seleccionamos la opción Batch Conversion.



Seleccionamos la carpeta que contiene las imágenes del paso anterior, y añadimos todos los archivos para ser transformados. Seleccionamos como directorio de salida la misma carpeta. Como tipo de conversión seleccionamos Batch conversion, formato de salida JPG, y usar las opciones avanzadas.

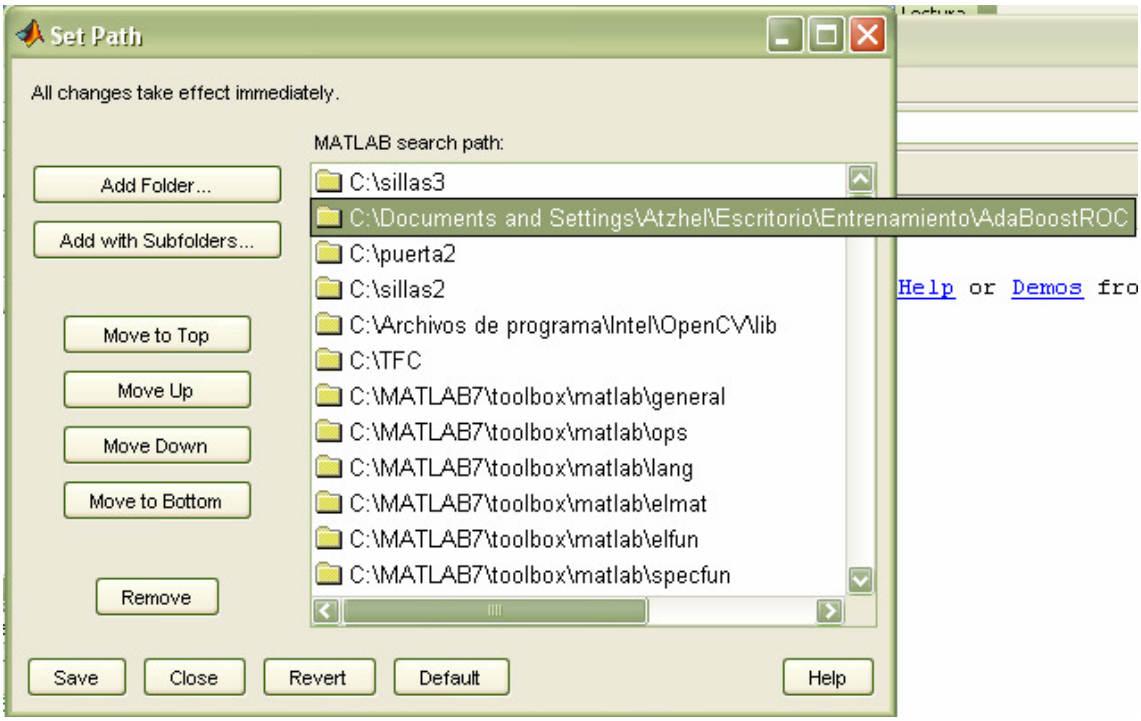


Dentro de opciones avanzadas seleccionamos cambiar la profundidad de color a 24 BPP, y en miscelánea Sobrescribir los archivos existentes



Una vez introducido todos los parámetros le damos a Start y ya tenemos las imágenes listas para iniciar el entrenamiento.

Incluimos en el path de Matlab el directorio que contiene las imágenes y la carpeta AdaBoostROC, y lo guardamos. A continuación, podemos cerrar Matlab si así lo deseamos.



Abrimos el Workspace que se encuentra dentro de DemoMatlabEngine con Visual Studio, abrimos el archivo DemoMatlabEngine.cpp y modificamos los siguientes valores.

En la línea 19:

char WorkDir[256]="**C:\\TFC**";

char directorio[256]="**C:\\sillas3**";

En WorkDir indicamos la carpeta donde deseamos que se guarden los resultados, y en directorio la carpeta que contiene las imágenes positivas. Las "\" de separación deben indicarse como "\\".

En la línea 54:

strncat(comando,",**6**,**6**,0);",**8**);

El primer número son los círculos que se desean usar para el descriptor, el segundo los sectores. El último número es la dimensión de la cadena que los contiene. En el caso de que fuese mayor se habría de incrementar, por ejemplo:

strncat(comando,",**10**,**10**,0);",**10**);

En la línea 70:

m_MatlabEngine->Execute("save **descriptor_silla**");

Indicamos el nombre con el que deseamos que se guarden la matriz de datos.

Nos situamos en la carpeta donde guardamos los resultados en el paso anterior y los cargamos con la instrucción “load *descriptor_silla*”. El nombre variara dependiendo de cómo lo hallamos guardado en el paso anterior.

Una vez tenemos los datos cargados ejecutamos:

train_cascade(silla,100,10,0.99,0.2,'C:/negatius','silla_06_06','jpg',20,20,0.5,250,6,6)

Donde los valores son, por orden:

- Nombre de la matriz que contiene los positivos.
- Número de negativos utilizados por nivel.
- Número de niveles a entrenar.
- Porcentaje de acierto mínimo para pasar al siguiente nivel.
- Porcentaje de error máximo para pasar al siguiente nivel.
- Ruta de la carpeta que contiene los negativos.
- Nombre de la carpeta en que se guardara el resultado.
- Formato de las imágenes.
- Tamaño mínimo en el eje y de los negativos a utilizar.
- Tamaño mínimo en el eje x de los negativos a utilizar.
- Sigma gradiente.
- Número de iteraciones máximas que se usaran por nivel.
- Número de círculos del descriptor.
- Número de sectores del descriptor.

Se puede parar la ejecución en cualquier momento, el resultado se guardara hasta el último nivel entrenado.

Entramos en la carpeta donde guardamos los resultados en el paso anterior, en nuestro caso 'silla_06_06'. Cargamos el resultado del entrenamiento con la instrucción "load estado_cascada". Esta instrucción será siempre la misma.

Para finalizar, ejecutamos:

PARSER_CASCADE(clasificador,**'silla_06_06.xml'**)

Donde el texto en negrita es el nombre con el que se desea guardar la cascada para poder ser utilizada. La extensión debe de ser .xml.

Con esto, ya tenemos una cascada entrenada y lista para ser utilizada por nuestro software detector.

6.3. Manual de usuario del detector

La interfaz del programa necesita los siguientes datos para ejecutarse:

- Imagen Entrada. La imagen que se desea analizar. Debe de estar en formato ".jpg". Puede seleccionarse usando el botón que hay al lado del cuadro de texto.
- Imagen Salida. Nombre de la imagen en la que se guardara el resultado. Por defecto el formato es ".bmp" para ser mostrada por la aplicación. Puede seleccionar la ubicación donde se guardará y el nombre mediante el botón. Si lo desea, puede introducir directamente el nombre, con extensión ".bmp" y se guardará en la carpeta del ejecutable. Si se selecciona el nombre de una imagen ya existente, esta se cargara en el espacio destinado de la interfaz.
- Archivo de la cascada. Archivo que contiene la cascada del clasificador. Se puede seleccionar el archivo con el botón correspondiente. Los números al final del nombre del archivo indican el número de círculos y sectores que se han usado para crearlo.
- Máximo niveles cascada. Número máximo de niveles que se cargarán del clasificador. En el caso de que se indiquen más niveles de los que posee el clasificador, se cargan todos de los que se disponga.
- Número de círculos del descriptor y número de sectores del descriptor son los parámetros que se han usado para crear la cascada.
- Tamaño mínimo de la ventana. Este mínimo variará según el tamaño de los objetos que contiene la imagen a analizar.
- Tamaño máximo de la ventana. Debe ser igual o mayor que el tamaño mínimo de la ventana.
- Tipo de incremento. Indica como se irá incrementando la ventana de detección, si sumando el factor de incremento o multiplicándolo.
- Tipo desplazamiento. Indica como se moverá la ventana de desplazamiento. Si se utiliza el método de multiplicación se recomiendo seleccionar el mismo método como tipo de incremento.
- Factor de incremento de la ventana. En el caso de seleccionar el incremento en forma de suma serán píxeles. Si se ha elegido multiplicar,

el factor será el porcentaje respecto a la ventana anterior. Por ejemplo, un incremento de 10% significará que en cada incremento se multiplicara el tamaño de la ventana anterior por 1.1.

- Desplazamiento en el eje X e Y de la ventana. Indica el desplazamiento por la imagen en número de píxeles en el caso de haber seleccionado como tipo de incremento suma. Si se ha seleccionado multiplicación, indicara el desplazamiento inicial, y se usara como factor el incremento introducido en la casilla Factor de incremento de la ventana. Si se ha seleccionado lineal, también indicaran el desplazamiento inicial, y el factor de incremento será la relación entre el tamaño inicial de la ventana y el desplazamiento.
- Número de intersecciones. Número mínimo de positivos en la misma zona que han de darse para que se considere como válido.
- Tamaño mínimo de la intersección. Tamaño de la intersección que ha de haber entre dos positivos para considerar que detectan el mismo objeto. Es un porcentaje de la media del tamaño mínimo y máximo de la ventana del descriptor.
- Preproceso que se desea aplicar a la imagen. El proceso de Proyección eliminará del borde de la imagen las zonas blancas. En el caso de no desear ningún preproceso se seleccionará Ninguno o nada.
- Margen, en píxeles, que se añadirá a la imagen después del proceso de proyección. En el caso de que después de aplicar la proyección el tamaño sea mayor que el de la imagen original, se dejarán las dimensiones de esta. Si no se indica ningún margen, este será de 0.

Una vez introducidos los datos, se apretará el botón Procesar Imagen. La barra de progreso indicará el progreso del procesado de la imagen. Durante el procesado puede pulsar en cualquier momento el botón Parar, el cual interrumpirá el procesado de la imagen. Una vez finalizado, si no se ha cargado la imagen, minimice la interfaz y luego vuelva a abrirla.

Ejemplo

Seleccionamos como imagen de entrada la imagen Prueba7 que viene en el directorio de la interfaz, y como nombre de la imagen de Salida, Salida7.

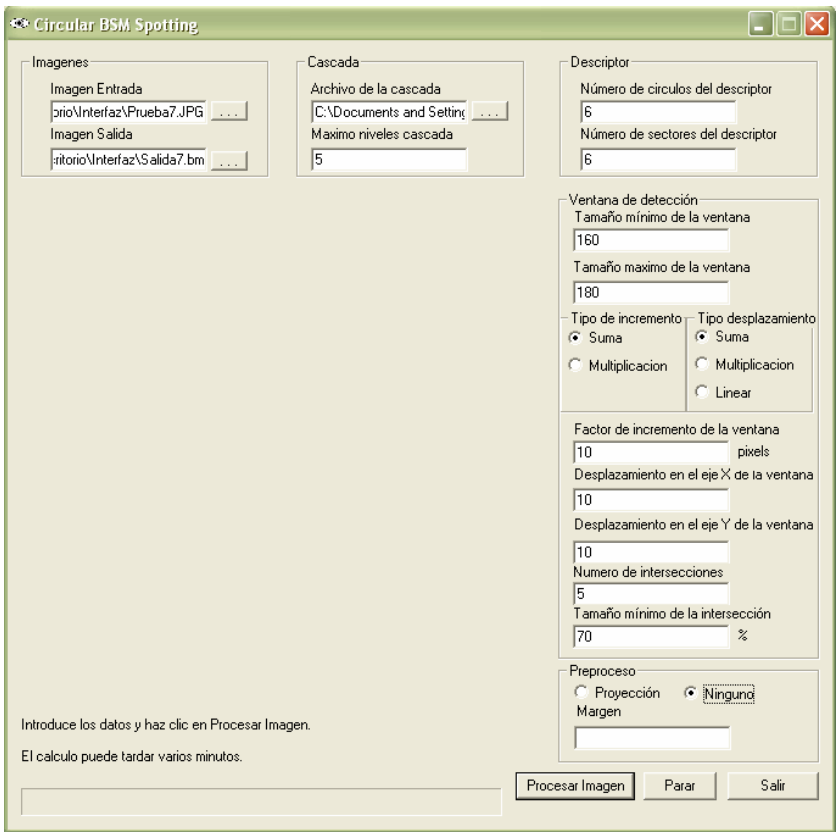
Elegimos como cascada el archivo sillas_06_06, e indicamos que queremos cargar 5 niveles de cascada. Tal y como indica el nombre del archivo, ha sido creado con un descriptor de 6 círculos y 6 sectores, por lo tanto introducimos esos valores en el lugar correspondiente.

Si observamos la imagen, vemos que la silla más pequeña se encuentra dentro de un cuadrado de aproximadamente 160 píxeles de lado. Introducimos este valor como tamaño mínimo de la ventana. Como máximo introduciremos 180.

Seleccionamos como tipo de incremento y de desplazamiento suma y como incremento de la ventana 10 píxeles. También elegimos como desplazamiento en el eje X e Y de la ventana 10 píxeles.

Indicamos que queremos que el número mínimo de intersecciones sea de 5, y que el tamaño de estas sea del 70%. Por último, no deseamos ningún tipo de preproceso, y así se lo indicamos.

Debemos tener la ventana de la interfaz de la siguiente forma.



Para iniciar el proceso le damos a Procesar Imagen. Una vez finalizado obtenemos la siguiente imagen.