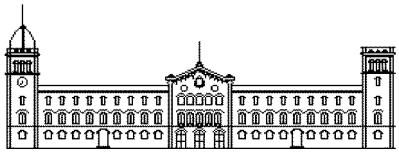


Trabajo fin de carrera

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Facultad de Matemáticas
Universidad de Barcelona

DESCRIPTOR DE OBJETOS BASADO EN FORMA

Mario Guitart Matamoros

Director: Sergio Escalera Guerrero
Realizado en: Departamento de Matemática Aplicada y Análisis. UB

Barcelona, 18 de febrero de 2009

1.Resumen

El descriptor circular BSM es un software desarrollado en lenguaje de programación visual c++ para resolver, de forma eficaz, el problema de la descripción de objetos en imágenes. La finalidad de esta aplicación es su uso en visión artificial, tanto en la tarea de aprendizaje como en la de detección de objetos. El software contiene una parte de código en visual C++ y un fichero MEX que permite realizar el aprendizaje de objetos en Matlab. El descriptor implementado, basado en la técnica BSM de Sergio Escalera, nos permite conseguir robustez frente a cambios de iluminación, cambios en el punto de vista, deformaciones rígidas o elásticas, y sobretodo invariancia a rotación. La técnica se basa en la descripción de la forma o estructura visual de los objetos, y la invariancia a rotación se consigue gracias a la posibilidad de poder rotar la descripción en función de la colocación del objeto en la imagen.
Los resultados obtenidos a la hora de detectar objetos confirman la robustez de los métodos implementados en este proyecto.

1.Resum

El descriptor circular BSM és un software desarrollat en llenguatge de programació visual C++ per resoldre, de manera eficaç, el problema a l'hora de descriure objectes trobats en imatges. La finalitat d'aquesta aplicació és l'ús en visió artificial, tant en la tarea d'aprenentatge com en la detecció d'objectes. El software conté una part de codi en visual C++ i un arxiu MEX que permet realitzar l'aprenentatge d'objectes en Matlab. El descriptor implementat, basat en la tècnica BSM de Sergio Escalera, ens permet aconseguir robustesa enfront de canvis d'iluminació, canvis en el punt de vista, deformacions rígides o elàstiques, i sobretot invariància a rotació.
La tècnica es basa en la descripció de la forma o l'estructura visual dels objectes, i l'invariància a rotació s'aconsegueix gràcies a la possibilitat de poder rotar la descripció en funció de la col.locació de l'objecte en l'imatge.
Els resultats obtinguts a l'hora de detectar objectes confirmen la robustesa dels mètodes implementats en aquest projecte.

1.Abstract

The circular Blurred Shape Model descriptor BSM is a software developed in C++ to robustly solve the problem of object description in images. The purpose of this application is its use in machine vision, both in the learning task and the detection of objects. The software contains a code in visual C++ and a MEX file that enables learning objects in Matlab. The implemented descriptor, based on the technique of BSM Sergio Escalera, can achieve high robustness against lighting changes, changes in point of view, rigid or elastic deformations, and especially to rotation. The technique is based on the description of the shape or structure of visual objects, and the rotation invariance is achieved because of the possibility of rotating the descriptior correlogram in terms of the diagonal of major density. The results in the detection of objects confirm the robustness of the methods implemented in this project.

Índice

2. Introducción

En este apartado estudiamos el problema que nos plantean, explicamos los descriptores que nos han servido de inspiración y ofrecemos una propuesta que cumpla los requisitos.

2.1 Problema

El problema planteado en este proyecto se engloba dentro del estudio de la Visión artificial, también conocida como Visión por Computador (del inglés Computer Vision) o Visión técnica, que es un subcampo de la inteligencia artificial.
La visión artificial es una técnica basada en la adquisición de imágenes y procesamiento en algún tipo de CPU (computador, microcontrolador, DSP, etc), con la finalidad de extraer determinadas propiedades de la imagen.
Su uso es aplicable a procesos científicos, militares y está introducido en algunas tareas industriales con el fin de automatizar procesos reservados anteriormente a la inspección humana. Este hecho conlleva a que algunas empresas lo adopten para la realización de trabajos visuales altamente repetitivos que sean fatigosos o difíciles de realizar para un operario.

Un sistema de Visión artificial se compone básicamente de los siguientes componentes:

- *Captador de Imagen*: Es el encargado de recoger las características del objeto bajo estudio.

- *Adquisición de imágenes*: Es la interfaz entre el sensor y la computadora o módulo de proceso que permite al mismo disponer de la información capturada por el sensor de imagen.

- *Descriptor o Algoritmos de análisis de imagen*: Es la parte inteligente del sistema. Su misión consiste en aplicar las transformaciones necesarias y extracciones de información de las imágenes capturadas, con el fin de obtener los resultados para los que haya sido diseñado.

- *Computadora o módulo de proceso*: Es el sistema que analiza las imágenes recibidas por el sensor para extraer la información de interés en cada uno de los casos implementados y ejecutar los algoritmos diseñados para la obtención de los objetivos.

- *Proceso automatización*: Una vez obtenidos y procesados los resultados en el computador, esta parte ejecuta las acciones asociadas a los resultados.

En este proyecto nos centramos en el **Descriptor o Algoritmo de análisis de imagen,** que se basa en intentar crear una descripción de los componentes u objetos que se encuentran en la imagen. Los descriptores son el primer paso para poder encontrar la conexión entre los píxeles contenidos en una imagen digital y aquello que los humanos recordamos después de haber observado durante unos minutos una imagen o un conjunto de las mismas. Para obtener tal descriptor necesitamos crear un software que sea capaz de analizar objetos dentro de una imagen y un algoritmo que describa dicho objeto de forma genérica.

Este software tendrá una doble funcionalidad dentro de la Visión artificial, ya que su uso formará parte del proceso de aprendizaje, necesario para tener almacenadas las descripciones de todos los objetos capaces de ser detectados por la visión humana. Su uso también estará presente en la parte de detección de objetos.
El algoritmo utilizado funcionará sobre una imagen con objetos sobre fondo negro ó blanco y realizará un barrido de dicha imagen hasta encontrar todos los puntos que pertenecen al objeto. Usaremos un vector que será nuestro descriptor de objeto y donde almacenaremos valores en función de la ubicación de los puntos que pertenecen al objeto en cuestión.

La descripción de los objetos en la fase de aprendizaje se basará en la introducción de imágenes en las que únicamente aparezca el objeto del cual queremos aprender. Tanto la ubicación, como el ángulo o rotación del objeto en la imagen no sería un problema ya que la descripción debe ser genérica.

Este software también formará parte de un proceso de detección donde su descripción será capaz de detectar los objetos aprendidos en cualquier imagen que le queramos introducir, y este proceso estará descrito en la memoria del alumno *Alberto Escudero*.

2.2 Estado del Arte

• En 1999, David Lowe propuso una técnica de descripción de objetos invariante a escala y rotaciones, el Scale-Invariant Feature Transform (SIFT) [1].
Hasta ese momento todos los detectores de objetos creados consideraban un único nivel de detalle (escala) y no tenían en cuenta que los objetos de interés pueden tener diferentes tamaños o estar a diferentes distancias de la cámara. La técnica se dividía en 2 partes diferenciadas:

1) El **detector SIFT** que utilizaba la combinación de varias técnicas para detectar objetos:

- La *representación espacio-escala* de una imagen: Esta técnica consiste en crear una familia de imágenes suavizadas a diferentes niveles de detalle (definido por un parámetro de escala t). El detector SIFT creaba el conjunto de imágenes en un rango entre t y $2t$ llamado octava.

- *Pirámides guassianas*: Formadas por la familia de imágenes suavizadas donde la base contiene la más alta resolución. En el caso del detector SIFT, a partir de la escala $2t$ se divide a la mitad de resolución y se usa esta resolución para la siguiente octava.

- *Filtro Log*: Teniendo en cuenta que el espacio escala representaría la familia de imágenes y la pirámide gaussiana es el núcleo de suavizado de la imagen que ha de ser Gaussiana, suponemos que los cruces por cero de estas ecuaciones representarían los bordes del objeto, y los extremos permiten detectar manchas. En el caso del detector SIFT sustraen dos escalas consecutivas. Se compara cada píxel con todos sus vecinos y sólo se seleccionan los extremos de *espacio-escala* que sean mayores o menores que sus vecinos.

2) **Descriptor SIFT** basado en Histogramas de orientación del gradiente: Este descriptor es invariante a rotación ya que se realiza una normalización en rotación que permite comparar puntos en varias orientaciones, donde la orientación de un punto es la orientación dominante del gradiente en su vecindad.

El gradiente es un vector bidimensional cuyos componentes están dados por las primeras derivadas de las direcciones verticales y horizontales. Podemos definir el gradiente para cada punto de la imagen, como el vector que apunta en dirección del incremento máximo posible de intensidad, y la magnitud del gradiente del vector corresponde a la cantidad de cambio de intensidad en esa dirección.

Los pasos realizados para obtener el descriptor SIFT son:

- Calcular la magnitud y la orientación del gradiente en la vecindad del punto utilizando la primera imagen suavizada.

- Una vez tenemos la orientación de todos sus vecinos, creamos un histograma, donde su máximo determinará la orientación del punto. De aquí podremos obtener los ejes locales para cada punto de interés.

- En el caso de existir en el histograma picos superiores al 80% del máximo se genera un nuevo punto para cada pico con su orientación correspondiente.

- Se segmenta la vecindad en regiones de *4x4* píxeles.

- Se genera un histograma de orientación de gradiente para cada región usando una ponderación Guassiana de ancho 4 píxeles.

- Hay que tener en cuenta que un pequeño desplazamiento espacial provoca que la contribución de un píxel pase de un segmento a otro y cambie la descripción. En el caso de una pequeña rotación la contribución podría pasar de una orientación a otra.

- Para evitar este problema un píxel contribuye a todos sus vecinos, multiplicando la contribución por un peso 1-*d*, donde *d* es la distancia al centro del segmento.

- Por último se obtiene un histograma tridimensional de *4x4x8* = 128 casillas considerando 8 direcciones principales y formando un vector con el valor de todas las casillas. Este vector se genera para cada punto de interés.

El descriptor SIFT creado por David Lowe es parcialmente robusto a cambios de iluminación, puntos de vista, se calcula rápido y es muy distintivo; pero también tiene inconvenientes ya que no es robusto a deformaciones rígidas o elásticas de objetos. Esto es debido a que a la hora de describir se basa en la creación del vector descriptor a partir de las normales de los puntos de interés del objeto y eso provocaría que pequeños

cambios de curvatura varíen radicalmente el descriptor.

• En 2004 apareció una modificación a SIFT propuesta por Ke y SuKthankar [2] cuya idea era obtener un descriptor que sea tan distintivo y robusto como SIFT, pero con un vector de menos componentes. Para reducir la dimensionalidad usaron la técnica de Análisis de Componentes Principales (PCA). El cálculo de la orientación de los puntos de interés se realiza igual que en SIFT, pero se trabaja con ventanas de *39x39*, girando la ventana según la orientación dominante y formando un vector concatenando los mapas de gradientes horizontales y verticales.

En este caso se obtiene un vector de *2x39x39* = 3042 elementos, pero no toda la información es significativa. Para reducir el número de casillas se recopilan muchos ejemplos de puntos de interés y utilizando sus vectores calculamos la matriz de covarianza, obteniendo así el promedio. Después diagonalizamos la matriz y la organizamos por eigenvalores decrecientes, donde los primeros serán los más altos.

A partir de aquí, para cada nuevo punto de interés se obtiene el vector inicial y se convierte al sistema de ejes de los eigenvectores. Finalmente el descriptor estará constituido por las 20 primeras componentes, los 20 ejes más distintivos.

De esta manera se consiguió crear una versión simplificada del descriptor SIFT, pero aun así continuaba presentando carencias respecto a deformaciones rígidas o elásticas de objetos.

• Otro descriptor creado posteriormente al de David Lowe en 1999 fue el denominado SURF [3]. Este descriptor utiliza la siguiente técnica:
Para determinar los ejes {u,v} que representen un eje de referencia local se hace un histograma de las orientaciones del gradiente dentro de la región. El histograma contiene sólo 6 celdas. Para calcular el descriptor SURF, se consideran los gradientes d_X paralelos al eje u, y los gradientes d_Y paralelos al eje v. Se usan *4x4* subregiones. Una vez obtenidos los gradientes, se genera un vector *4x4x4* = 64 casillas considerando 4 direcciones principales para cada región. Para rellenar el vector se utiliza un algoritmo que tiene en cuenta la magnitud del gradiente y su ángulo de inclinación. La creación de los descriptores en esta técnica se basa en sumas bidimensionales de los vectores obtenidos de los puntos de ínteres, a diferencia de la técnica SIFT que realizaba sumas tridimensionales.

El sistema SURF es robusto a cambios de iluminación, puntos de vista y es más rápido de calcular que el SIFT original, pero además de no ser invariante a deformaciones rígidas o elásticas, tampoco lo es a rotación.

2.3 Propuesta

Teniendo en cuenta las ventajas e inconvenientes de los descriptores explicados en el apartado anterior, y con el objetivo de solucionar la falta de variancia frente a deformaciones rígidas o elásticas, proponemos crear un descriptor que se focalice en la "forma" o estructura del objeto.

Para crear este software nos hemos basado en la técnica BSM [4], (Blurred Shape Model) creada por Sergio Escalera, que esta inspirado en el proceso del cerebro humano y en su capacidad de ver e interpretar cualquier imagen (reconocimiento de formas, estructuras y dimensiones) para crear modelos de descripción y clasificación de símbolos escritos a mano.

El descriptor BSM a diferencia de otros descriptores detecta las variaciones, las deformaciones elásticas y las distorsiones no uniformes que se pueden producir en la reproducción manual de cualquier símbolo. Esto es posible dividiendo las regiones que componen cada imagen en subregiones, formando una rejilla de cuadros o cuadrícula, y guardando información de cada forma, distinguiendo pequeñas diferencias entre símbolos.

Aunque este método es eficaz, tiene una pequeña carencia; no es invariante a rotación. En nuestro caso concretamente necesitamos un descriptor que sea invariante a deformaciones rígidas o elásticas e invariante a rotación; ya que tiene que ser capaz de describir de igual manera cualquier tipo de objeto que pertenezca a la misma familia, independientemente de la ubicación, tamaño o inclinación en la imagen. Analizando estos factores y basándonos en la técnica BSM comentada anteriormente, nuestra propuesta consiste en crear un descriptor circular.

Para definir esta técnica necesitamos dividir la imagen en sectores circulares, a diferencia del BSM que eran cuadrados. Esto se consigue dividiendo la altura/anchura de la imagen en una escala lineal de círculos concéntricos y seccionando la imagen. Una vez seccionada la imagen debemos tener bien claro que sectores serán vecinos entre sí, no como en otros descriptores en los que los puntos de interés influenciaban sobre toda la imagen. En nuestro caso, los vecinos únicamente serán aquellos que se encuentren en contacto directo con el sector estudiado. Un apunte a tener en cuenta es que para describir o detectar cualquier imagen, trabajaremos con su contorno.

Después de seccionar la imagen entramos en el proceso de descripción del objeto. Para esta parte debemos recorrer la imagen y para cada píxel realizar los siguientes pasos:

- Obtener el sector al que pertenece.
- Calcular su valor, ya que únicamente nos interesan aquellos píxeles negros, porque representan el borde del objeto.
- Para cada píxel negro, utilizar el algoritmo en el que los valores se obtienen calculando la distancia del píxel a los centros de los sectores vecinos.
- Con los valores obtenidos de cada píxel o punto de interés crearemos un vector que será nuestro descriptor, y cuya longitud será variable, ya que dependerá del número de sectores que queramos utilizar.

Una vez creado el descriptor en función de los puntos de interés tendríamos solucionado el tema de las deformaciones rígidas o elásticas, pero no la rotación. Para conseguir que nuestro sistema sea invariante a rotación necesitamos utilizar otro algoritmo, basado en el cálculo de los sectores que forman las diagonales dominantes en la imagen. Esta es una manera de saber cual sería la diagonal principal del objeto aprendido, y poder rotar

el vector descriptor para generalizar la descripción de todos los objetos que pertenecen a la misma familia independientemente de su grado de inclinación en la imagen.

Una vez rotado el vector en función de la diagonal principal tenemos en cuenta si se ha colocado a la inversa, para ello necesitaremos un método que calcule la mayor densidad entre la parte superior e inferior del descriptor y rote en consecuencia. Finalmente limitaremos los valores del vector descriptor en un rango entre 0 y 1.
De esta manera conseguiremos que la técnica creada sea invariante a rotación y comparable a diferentes escalas.

Este software estará bien validado antes de su uso en el proceso de aprendizaje y estará incluido también en el framework de detección creado por el alumno Alberto Escudero

2.4 Estructura Memoria

A partir de este apartado estructuramos la memoria de la siguiente forma:

- *Metodología:* Apartado donde mostramos los pasos utilizados en el proceso de análisis del problema y las técnicas de diseño necesarias antes de empezar con la implementación del código. Se divide en 2 partes fundamentales:

    - *Análisis:* Parte de la memoria dedicada a la creación de un Plan de Costes, descripción de los algoritmos utilizados y especificación de los Casos de Uso que tendrá la aplicación.

    - *Diseño:* Este apartado explica y muestra al detalle las técnicas UML utilizadas para la creación del software. Estas técnicas son el diagrama de Clases, los diagramas de Secuencia y las herramientas empleadas durante el ciclo de vida del proyecto.

- *Resultados:* Este apartado muestra los datos, métodos y parámetros utilizados en el proceso de pruebas de las diferentes utilidades de la aplicación, así como los experimentos realizados y su cuantificación.

- *Conclusiones:* A partir de los resultados obtenidos, mostraremos en esta parte de la memoria las conclusiones sacadas y posibles mejoras.

- *Bibliografía:* Apartado necesario para mostrar las fuentes utilizadas para aprender y desarrollar nuestro proyecto software.

- *Apéndices:* En este apartado mostraremos el contenido del CD y todos aquellos puntos de interés útiles para el uso y comprensión del software.

3. Metodología

En este apartado vamos a mostrar el proceso de Análisis de requisitos y la parte de diseño UML necesaria para el desarrollo del software.

3.1 Análisis

Nuestra parte de Análisis se centra en la creación de un cronograma y plan de costes donde podemos comprobar el ciclo de vida del proyecto software y estudiar su viabilidad. También analizamos los requisitos funcionales, utilizando el modelo de "Casos de Uso".

3.1.1 Cronograma y Plan de Costes

La planificación temporal para el proyecto Descriptor BSM circular la representamos utilizando un diagrama de Gantt y lo mostramos en la tabla (3.1).

Descriptor BSM Project

Número	Tarea	Especificación	Inicio	Final	Duración	2008			2009	
						Octubre	Noviembre	Diciembre	Enero	Febrero
1	Análisis Requisitos	No Funcionales , Casos de Uso, Viabiliadad, Costes	1/10/2008	8/10/2008	5					
2	Adquisición Licencias Software		8/10/2008	11/10/2008	3					
3	Modelado	Diagrama Clases, Secuencia...	11/10/2008	16/10/2008	3					
4	Implementación: Módulo Estructura	Seccionar Circularmente Imagen	16/10/2008	23/10/2008	5					
5	Implementación : Módulo Sectores	Rellenar Información Sectores	23/10/2008	6/11/2008	10					
6	Implementación : Módulo Descriptor	Rellenar Vector Descriptor, Generizar Vector	6/11/2008	27/11/2008	15					
7	Implementación : Módulo Matlab	Creación MEX	27/11/2008	4/12/2008	5					
8	Pruebas : Módulo Estructura		4/12/2008	6/12/2008	2					
9	Pruebas: Módulo Descriptor		6/12/2008	11/12/2008	3					
10	Proceso Integración	Proyecto Detector	11/12/2008	24/12/2008	9					
11	Pruebas Aprendizaje y Detección		14/1/2009	18/2/2009	25					
12	Implementación : Interficie Gráfica		14/1/2009	28/1/2009	10					
13	Memoria Proyecto		19/1/2009	18/2/2009	22					

Table 3.1: Cronograma Descriptor BSM

Como podemos comprobar el proyecto se dividirá en 5 procesos diferenciados :

- La parte de Análisis de los requisitos necesarios para satisfacer el problema.
- El proceso de modelado, donde incluimos las estructuras UML útiles para la posterior implementación.
- La parte de Implementación dividida en 5 módulos para conseguir escalabilidad e independencia entre ellos.
- El proceso de pruebas necesario para comprobar que los módulos funcionan correctamente de manera individual y en conjunto.
- Y la parte de Memoria donde recopilaremos todos los procesos creados hasta el momento y las técnicas de trabajo.

En el cronograma podemos observar que durante las primeras 10 tareas, los requisitos se deberán cumplir de manera secuencial. Sin embargo las tareas de pruebas, Interfície Gráfica y Memoria se podrán realizar de manera concurrente entre ellas.

Plan Costes

En el proceso de estudio y desarrollo del proyecto, son necesarios al menos dos perfiles: El de Gestor de proyecto y el de Analista/Programador.
En la actualidad dichos perfiles cobran sueldos del orden de los que se muestran en la siguiente tabla (3.2), donde se suponen una media de 240 días laborables por año y un coste de seguridad social a cargo de la empresa de un 33%.

Roles	Sueldo Bruto Anual	Coste Anual	Coste/día
Gestor Proyectos	*42000 €*	*55860 €*	*232,75 €*
Analista/Programador	*25000 €*	*33250 €*	*138,55 €*

Table 3.2: Sueldos Estipulados

El proyecto durará, contando la fase inicial de preparación 85 días laborables.
Supondremos que el jefe de proyecto dedicará una media de 3 horas semanales a la dirección de este proyecto, de modo que dedicará un total de 51 horas al proyecto.
El Analista/Programador dedicará una media de 2 horas diarias, lo que suma un total de 170 horas. El coste de personal lo representamos en la tabla (3.3).

Roles	Horas	Días	Coste/Día	Total
Gestor Proyectos	*51*	*6,4*	*232,75€*	*1.489,6 €*
Analista/Programador	*170*	*21,25*	*138,55€*	*2.944,2 €*
TOTAL:				***4.433,78 €***

Table 3.3: Coste Personal

A parte de los costes por personal que trabajan en el proyecto, debemos contabilizar los gastos materiales de los recursos usados.
Se puede realizar el proyecto en el siguiente sistema:

• Ordenador Portátil MacBook Pro Intel Core 2 Duo 2,4 GHz, 2 Gb DDR2 SDRAM y disco duro de 200 Gb, valorado en 1800 €.

• Conexión ADSL a Internet, valorada en 50 € mensuales.

• Licencia Microsoft Visual Basic 6.0: 448,829 €.

• Licencia Matlab Individual: 6000€

Al portátil se le estima una vida de 5 años, mientras que a ambos software de 4 años, hasta nuevas versiones.
De modo que se obtienen los siguientes costes imputables al material usado (tabla 3.4), sabiendo que la duración del proyecto a sido de 4,25 meses teniendo en cuenta un total de 20 días laborables al mes.

Recurso	Coste	Plazo Amortización	% Imputable	Coste imput.
Portátil	*1800 €*	*60 meses*	*(4,25/60)*100= 7%*	*126 €*
Internet	*50 €/mes*	*-*	*-*	*212,5 €*
Visual Basic	*448,829 €*	*48 meses*	*(4,25/48)*100= 8,8%*	*39,49 €*
Matlab	*6000 €*	*48 meses*	*(4,25/48)*100= 8,8%*	*528 €*
TOTAL:				***905,99 €***

Table 3.4: Coste Material

Sumando los valores anteriores se obtiene el coste de desarrollo total y lo mostramos en la tabla (3.5).

Concepto	Coste
Personal	*4.433,78 €*
Material	*905,99 €*
TOTAL	***5.339,77 €***

Table 3.5: Coste Proyecto

Estos valores obtenidos son una estimación detallada del presupuesto que se debería gastar una empresa, en la creación del proyecto software **Descriptor BSM Circular**.

3.1.2 Casos de Uso

Los casos de Uso fundamentales de la aplicación son "Aprender Objeto" y "Describir Objeto", como se ve en el diagrama (3.6).

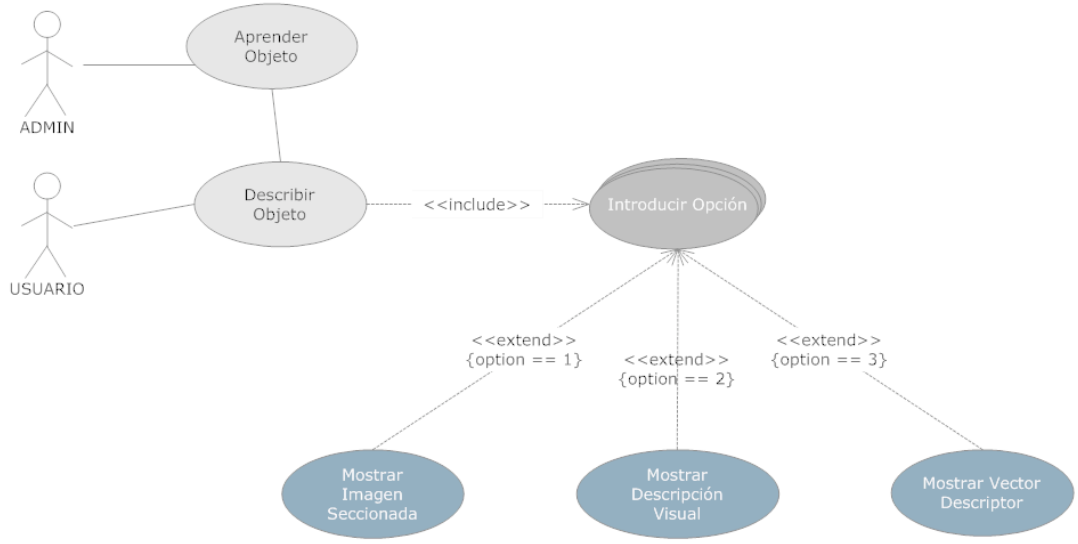


Diagrama 3.6: Casos de Uso

Como vamos a crear 2 aplicaciones diferentes para que los usuarios puedan describir objetos por paso de parámetros a la función o a través de un menú, necesitamos crear 2 casos de uso diferentes para "Describir Objeto".
En el primer caso tabla (3.7) definimos el caso de uso "Describir Objeto" que representaría la aplicación que contiene un menú de usuario. Mientras que en la tabla (3.8) definimos el segundo caso de uso "Describir Objeto" que necesita el paso de todos los parámetros en la ejecución.

CASO USO:	DESCRIBIR OBJETO		
Versión	1.0	**Fecha**	Octubre 2008
Autor	Mario Guitart		
Descripción	Una vez terminado el proceso de instalación del material necesario en la computadora, los usuarios podrán describir objetos de las imágenes que deseen, utilizando la aplicación Descriptor_Objetos_Consola.exe.		
Actores	Usuario		
Precondición	Antes de describir el objeto, el usuario debe tener las librerías OpenCV.		
Flujo principal	El caso de uso empieza cuando el usuario ejecuta el archivo Descriptor_Objetos_Consola pasándole como parámetro la imagen que desea describir. El sistema muestra al usuario el menú con las 4 actividades posibles: "mostrar secciones", "mostrar descriptor visual", "mostrar vector descriptor", "salir".		
Subflujos	*Mostrar Secciones* 1. El usuario selecciona la opción "mostrar secciones". 2. El sistema pide el número de círculos. 3. El usuario introduce el número de círculos. 4. El sistema comprueba que el valor introducido es un número. Si es un carácter, muestra un mensaje de error y vuelve a pedir el número de círculos. 5. El sistema pide el número de sectores. 6. El usuario introduce el número de sectores. 7. El sistema comprueba que el valor introducido es un número. Si es un carácter, muestra un mensaje de error y vuelve a pedir el número de sectores. 8. El sistema muestra una imagen con los sectores dibujados *Mostrar Descriptor Visual* 1. El usuario selecciona la opción "mostrar descriptor visual". 2. Los pasos 2-3-4-5-6-7 son iguales que en el subflujo Mostrar Secciones. 3. El sistema almacena la estructura de sectores. 4. El sistema analiza la imagen a describir. 5. El sistema crea un vector descriptor y lo generaliza. 6. El sistema muestra una nueva imagen, mostrando en escala de grises los valores del vector descriptor. *Mostrar Vector Descriptor* 1. El usuario selecciona la opción "mostrar vector descriptor". 2. Los pasos 2-3-4-5-6-7 son iguales que en el subflujo Mostrar Secciones. 3. El sistema almacena la estructura de sectores. 4. El sistema analiza la imagen a describir. 5. El sistema genera un vector descriptor y lo generaliza. 6. El sistema muestra por pantalla los valores del vector Descriptor. *Salir* 1. El usuario selecciona la opción "Salir". 2. El sistema muestra el mensaje de despedida y cierra la aplicación.		

Flujos Alternativos	*Imagen incorrecta* Si en el flujo principal, introduces una imagen de manera incorrecta o la imagen no existe, el sistema rechaza la petición mostrándote por pantalla los pasos a seguir para ejecutar bien la aplicación. *Parámetros Incorrectos* Si en el flujo principal no introduces ningún parámetro, el sistema rechaza la petición mostrándote por pantalla los pasos a seguir para ejecutar bien la aplicación.
Poscondición	
Requisitos no funcionales	
Prioridad	Prioritario
Comentarios	

Table 3.7: Descriptor Objetos Consola

CASO USO:	**DESCRIBIR OBJETO**		
Versión	1.0	**Fecha**	Octubre 2008
Autor	Mario Guitart		
Descripción	Una vez terminado el proceso de instalación del material necesario en la computadora, los usuarios podrán describir objetos de las imágenes que deseen, utilizando la aplicación Descriptor_Objetos_Param.exe		
Actores	Usuario		
Precondición	Antes de describir el objeto, el usuario debe tener las librerías OpenCV.		
Flujo principal	El caso de uso empieza cuando el usuario ejecuta el archivo Descriptor_Objetos_Param pasándole como parámetros la imagen que desea describir, el número de círculos, el número de sectores y la opción a elegir. El sistema comprueba que los datos sean correctos y en función de la opción escogida realiza la tarea: “mostrar secciones”, “mostrar descriptor visual” o ”mostrar vector descriptor”.		
Subflujos	*Mostrar Secciones* 1. El usuario ha introducido la opción 1. 2. El sistema estructura la imagen en función de los valores introducidos por el usuario. 3. El sistema muestra una imagen con los sectores dibujados. *Mostrar Descriptor Visual* 1. El usuario ha introducido la opción 2. 2. El sistema almacena la estructura de sectores. 3. El sistema analiza la imagen a describir. 4. El sistema genera un vector descriptor y lo generaliza. 5. El sistema muestra una nueva imagen, mostrando en escala de grises los valores del vector descriptor. *Mostrar Vector Descriptor* 1. El usuario ha introducido la opción 3. 2. El sistema almacena la estructura de sectores. 3. El sistema analiza la imagen a describir. 4. El sistema genera un vector descriptor y lo generaliza. 5. El sistema muestra por pantalla los valores del vector Descriptor.		

Flujos Alternativos	*Imagen incorrecta* Si en el flujo principal, introduces una imagen de manera incorrecta o la imagen no existe, el sistema rechaza la petición mostrándote por pantalla los pasos a seguir para ejecutar bien la aplicación. *Parámetros Incorrectos* Si en el flujo principal no introduces ningún parámetro o el número de parámetros es superior a 4, el sistema rechaza la petición mostrándote por pantalla los pasos a seguir para ejecutar bien la aplicación. *Introducción Carácteres* Si en alguno de los 3 parámetros correspondientes, al número círculos, número sectores u opción, introduces un carácter en lugar de un valor, el sistema indica el fallo cometido mostrándote los pasos correctos para ejecutar la aplicación.
Poscondición	
Requisitos no funcionales	
Prioridad	Prioritario

Table 3.8: Descriptor Objetos Parámetros

Por último representamos el caso de uso "Aprender Objeto" en la tabla (3.9).

En este caso de uso tendrá importancia la utilización de Matlab, ya que en el aprendizaje se cargan las imágenes en Matlab y se llama al método BSM descriptor para obtener la descripción de cada objeto contenido en la imagen. Para el uso de la aplicación con Matlab deberemos modificar la interficie del proyecto, creando un Mex-File.

CASO USO:	**Aprender Objetos**		
Versión	1.0	Fecha	Octubre 2008
Autor	Mario Guitart		
Descripción	Una vez terminado el software en visual c++, creamos un fichero MEX-File necesario para poder utilizar la aplicación desde Matlab. Su uso formará parte del proceso de aprendizaje porque necesitamos la descripción de cada uno de los objetos para guardar valores en la cascada.		
Actores	Admin.		
Precondición	Antes de aprender el objeto, el administrador debe tener instalado Matlab y utilizar imágenes donde sólo aparezca un objeto, así su aprendizaje será correcto.		
Flujo principal	El caso de uso empieza cuando el administrador llama a la función Mex desde matlab, pasándole los parámetros siguientes: "imagen cargada", "número círculos", "número sectores" y "descriptor logarítmico". En el caso de introducir mal los parámetros, la aplicación indica el error y terminaría el caso de uso.		

Subflujos	*Describir objeto* 1. El administrador ha llamado a la función Mex. 2. El sistema convierte la imagen Matlab en IplImage. 3. El sistema comprueba que los parámetros sean correctos. 4. El sistema analiza la imagen para rellenar un vector con la descripción. 5. El sistema devuelve el vector descriptor. *Guardar Descripción* 1. El archivo Mex devuelve el vector con la descripción. 2. El administrador llama al método crearCascada pasándole el vector. 3. El sistema utiliza el algoritmo necesario para crear la cascada.
Flujos Alternativos	*Parámetros Incorrectos* Si en el flujo principal no introduces ningún parámetro o el número de parámetros es superior a 4, el sistema rechaza la petición mostrándote un error y termina el caso de uso.
Postcondición	
Requisitos no funcionales	
Prioridad	Prioritario
Comentarios	

Table 3.9: Aprender Objetos

3.1.3 Algoritmos

En el proceso de descripción de un objeto plasmado en una imagen se realizan 4 tareas fundamentales:

A) *Introducción de Parámetros:*

1. Cargamos la imagen jpg en la variable **img Iplimage** utilizando el método **cvLoadImage**.

2. Llamamos al método **calculoVector(img, n_circulo, n_sectores, log)** pasándole los siguientes valores:
   - **2.1.** La imagen cargada correctamente.
   - **2.2.** El número de círculos necesarios para estructurar la imagen en diferentes alturas.
   - **2.3.** El número de sectores para dividir las alturas en secciones.
   - **2.4.** El tipo de descriptor (lineal o logarítmico).

B) *Seccionar Imagen*:

1. Dentro del método **calculoVector** definimos un puntero **sector** a un **struct de sectores**, un puntero **altura** de tipo **int** y un puntero **sectors** de tipo **double** para guardar los ángulos de los sectores.

2. En función de la variable **log**, llamaremos al método **calcularCirculosLineal(img,**

n_circulo, altura) pasándole la imagen, el número de círculos, y el puntero de alturas o llamaremos al método **calcularCirculosLog** con los mismos parámetros.

3. Una vez seccionada la imagen por alturas llamamos al método **calcularSecciones(img, n_sectores, sectors)** para guardar los ángulos.
4. Con los datos almacenados en los punteros **altura** y **sectors** llamamos al método **calcularSectores(sector, altura, sectors, secciones, n_sectores, puntoCA, n_circulo).**
   - **4.1.** Dentro del método **calcularSectores()** asignamos a cada **struct sector** sus 2 alturas y 2 ángulos que definen su ubicación en la imagen.
   - **4.2.** Llamamos al método **puntoMedio(sector, num, puntoCA)** pasándole el puntero sector, el número de secciones, y el punto medio de la imagen, para almacenar los puntos medios de cada sector.
     - **4.2.1.** Dentro del método **puntoMedio()**, para cada sector llamamos al método **coordenadasPunto(sector, cont, ang, dist, puntoCA)** pasándole el ángulo y la distancia respecto del centro de la imagen, necesarios para obtener las coordenadas reales en la imagen.
   - **4.3.** Después llamamos al método **calculoVecinos(sector, num, n_sectores, altur, n_circulo)** pasándole los sectores, el número de secciones y el puntero de alturas.
     - **4.3.1.** Dentro del método **calculoVecinos()** para cada sector, en función del grupo al que pertenece llamamos a los siguientes métodos:
       - **4.3.1.1.rellenarVecinosG1(sector, cont, n_vecinos, num, n_sectores)** para los sectores centrales, pasándole la variable cont que identifica el sector. Introducimos los vecinos en su struct comprobando antes si ya lo hemos introducido, con el método **esVecino(sector,cont, n_vecinos,c).**
       - **4.3.1.2.rellenarVecinosM2(sector,cont,n_vecinos,num,n_circulo)** para los sectores intermedios, teniendo en cuenta que el número de vecinos varía.
       - **4.3.1.3.rellenarVecinosF3(sector,cont,n_vecinos,num,n_circulo)** para los sectores más periféricos.

C)Describir Objetos:

1. Una vez tenemos toda la información necesaria de cada sector llamamos al método **calculoDescriptor(img, sector, vector, secciones)** introduciéndole la imagen, el puntero **sector**, un puntero **vector** donde guardaremos la descripción numérica y el número de secciones.
   - **1.1.** Dentro del método **calculoDescriptor()** definimos un puntero sector_vector donde guardaremos los valores temporales de cada sector.
   - **1.2.** Creamos un bucle donde para cada píxel de la imagen, llamamos al método **obtenerSector(img, sector, pixel, secciones)** que nos devuelve el identificador del sector donde se encuentra y llamamos al método **cvGet2D(img, y, x)** para extraer el valor numérico entre 0 y 255 del píxel.
     - **1.2.1.** Dentro del método **obtenerSector()** utilizamos el método **calculaAngulo(img, puntoM, pixel)** para obtener el ángulo respecto el eje central de la imagen.
     - **1.2.2.** Llamamos al método **distanciaPuntos(pixel, puntoM)** para calcular la distancia del píxel respecto el centro de la imagen.

1.2.3. Hacemos un bucle para cada sector llamando a los métodos **compararAngulo(sector[i], ang) y compararAltura(sector[i], dist)** para comprobar si pertenece a algún sector.

1.3. En el caso de ser un punto negro en la imagen llamamos al método **rellenarVector(sector, sec, pixel, vector, sector_vector)** pasándole la variable **sec** que es el identificador del sector.

1.3.1. Dentro del método **rellenarVector()** calculamos para cada sector vecino la distancia de su punto medio con respeto el píxel utilizando el método **distanciaPuntos().**

1.3.2. Guardamos los valores en el puntero **sector_vector** y realizamos la suma de todos los valores.

1.3.3. Después dividimos cada valor almacenado por la suma y volvemos a sumar todos los valores, para posteriormente volver a dividir cada valor.

1.3.4. Los valores almacenados en **sector_vector** se los sumamos a los que tenemos en el puntero **vector** que será nuestra descripción.

2. Cuando ya tenemos el vector descriptor, llamamos al método **escalaBinaria(vector, secciones)** para transformar esos valores en un rango entre 0 y 1.

D)*Generalizar Vector :*

1. Comprobamos si la imagen es blanca con el método **imagenBlanca(vector, secciones),** ya que en este caso no modificamos la descripción.

2. Si la imagen contiene un objeto, llamamos al método **generizarVector(sector, vector, secciones, n_circulo)** para describir de la misma manera al mismo objeto, independientemente de su colocación y grado de inclinación en la imagen.

2.1. Dentro del método **generizarVector(),** comprobamos la diagonal de la imagen que posee más valor.

2.2. Rotamos el vector para que la diagonal principal se encuentre en el sector 0. Para esto utilizamos el método **rotarVector(vector, d, sector, secciones, n_circulo)** pasándole la variable **d** con la diagonal principal.

2.3. Una vez rotado el vector buscamos la dirección de giro, llamando al método **calcularDireccionGiro(sector, vector, secciones, n_circulo)** que nos devuelve la mitad de la imagen con más valor numérico.

2.4. En función del resultado, si la imagen se a centrado a la inversa, volveremos a rotar el vector.

Una vez realizados estos pasos , ya tendríamos el vector que describe nuestro objeto.

3.2 Diseño e implementación

En la parte de diseño del proyecto hemos definido las herramientas necesarias para su implementación, así como los diagramas UML útiles para la comprensión de la solución adoptada.

3.2.1 Herramientas Utilizadas y detalles Implementación

Para la creación del proyecto Descriptor BSM circular necesitamos utilizar las siguientes herramientas:

• Ordenador Portátil MacBook Pro Intel Core 2 Duo 2,4 GHz, 2 Gb DDR2 SDRAM y disco duro de 200 Gb.

• Microsoft Windows XP Professional Versión 2002 Service Pack 2: Sistema operativo sobre el que funcionará la aplicación.

• Microsoft Visual Basic 6.0: Editor y compilador, que utilizaremos para crear el código del proyecto en c++.

• Matlab Versión 6.5: Programa a través del cual realizaremos el proceso de aprendizaje.

• Librerías Ipl25 y Opencv 1.0 necesarias para la creación y el análisis de las imágenes.

• SmartDraw 2009: Software de diseño para la creación de cronogramas y estructuras UML.

• Conexión ADSL de telefónica para la búsqueda de información a través de la red.

Alguno de los detalles importantes a destacar en el proceso de implementación serían los siguientes:
El código principal del software estará escrito en lenguaje C++ y su finalidad principal será la descripción de objetos encontrados en imágenes en formato jpg.
Necesitaremos clases bien diferenciadas para crear por separado los diferentes módulos de implementación comentados en la tabla (3.1).
Crearemos diferentes archivos .h para tener las declaraciones de los métodos separados de la parte del código.
Las imágenes introducidas serán convertidas en IplImage para poder facilitar su manejo durante el proceso de la aplicación.
Como vamos a tener 2 ejecutables diferentes, necesitamos crear 2 clases main, una para cada ejecutable. El resto del código será común para ambos.

Para el proceso de aprendizaje necesitamos crear una interficie independiente en lenguaje C++ que pueda ser utilizada en Matlab. Esta interficie ha de contener un método llamado Mex-Function que recibirá los parámetros enviados desde Matlab y devolverá el vector descriptor. El método utilizará el código del Descriptor BSM circular para poder describir el objeto. Con este código crearemos una dll para que pueda ser utilizada desde Matlab.

Además tenemos que crear una librería independiente de los ejecutables y la dll para Matlab. Esta librería estará preparada para su uso desde cualquier aplicación en c++.
La funcionalidad será la misma que el archivo ejecutable Descriptor_Objetos_Param.exe. Podremos llamar al método calcularVector() pasándole como parámetros la imagen, número de círculos, número de sectores y la opción.

3.2.2 Diagrama de Clases

Las clases utilizadas durante el proceso de implementación del proyecto las mostramos en la imagen (3.10).

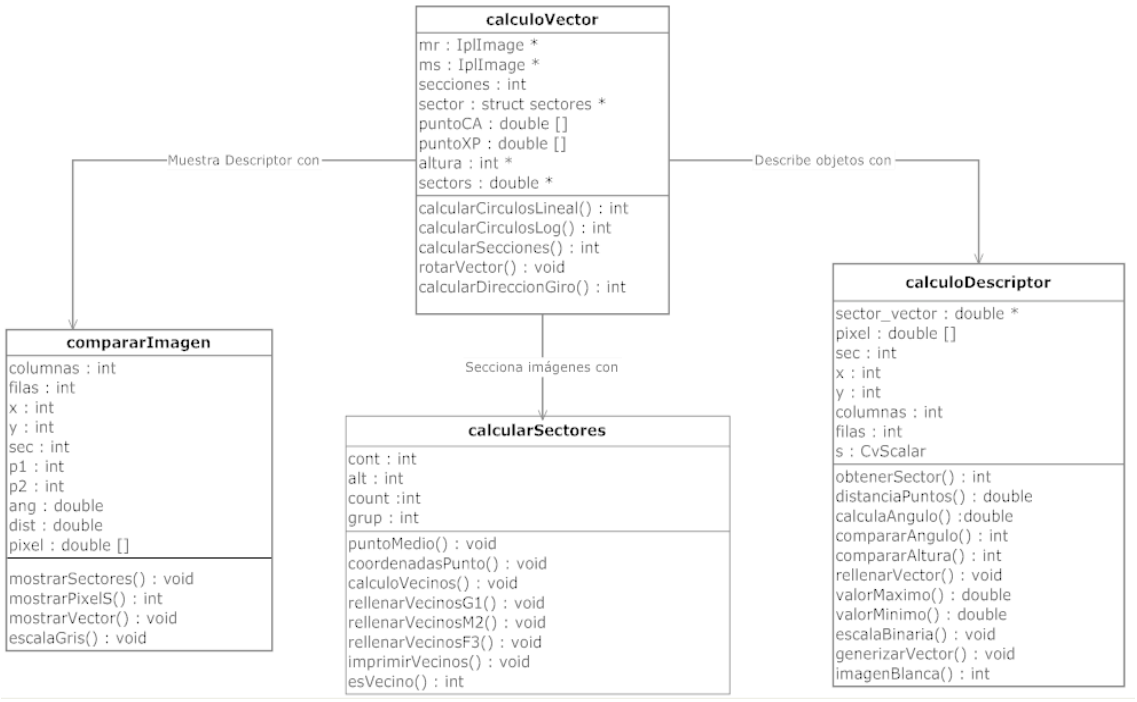


Imagen 3.10: Diagrama Clases BSM circular

Como podemos observar en el diagrama, las 4 clases son módulos independientes, donde cada una de ellas realiza una tarea especifica.

La clase calculoVector la podemos considerar la clase principal, ya que es la encargada de realizar llamadas a los métodos de las diferentes clases para obtener finalmente el vector descriptor.

En el diagrama vemos como desde la clase calculoVector, seccionamos imágenes llamando a métodos de calcularSectores, describimos objetos llamando a métodos de calculoDescriptor y mostramos el descriptor llamando a métodos de compararImagen.

En la clase calculoVector tenemos los punteros necesarios para trabajar durante todo el proceso:

- Struct sectores * sector –> Apunta al conjunto de estructuras de sectores creados después de llamar a métodos de la clase calcularSectores.
- Double * vector –> Apunta a las posiciones de memoria que contendrán la descripción del objeto, una vez utilizados los métodos de la clase calculoDescriptor.

- IplImage * mr, ms –> Son las imágenes creadas internamente para mostrar, las secciones en las que se divide la imagen o el vector descriptor visual. Estas imágenes se obtendrán llamando a métodos de la clase compararImagen.

Estas 4 clases muestran el código del proyecto necesario para describir objetos, que será secuencial.

3.2.3 Diagramas de Secuencia

En el diagrama (3.11) mostramos la secuencia de pasos necesarios para aprender un objeto. Como hemos comentado en el capitulo anterior, necesitamos crear una dll mex-file para poder ejecutar el proyecto desde Matlab. En este caso la funcionalidad de la aplicación será recibir los parámetros desde matlab y devolver el vector descriptor.
Esta tarea únicamente la realizará el Administrador de la aplicación.

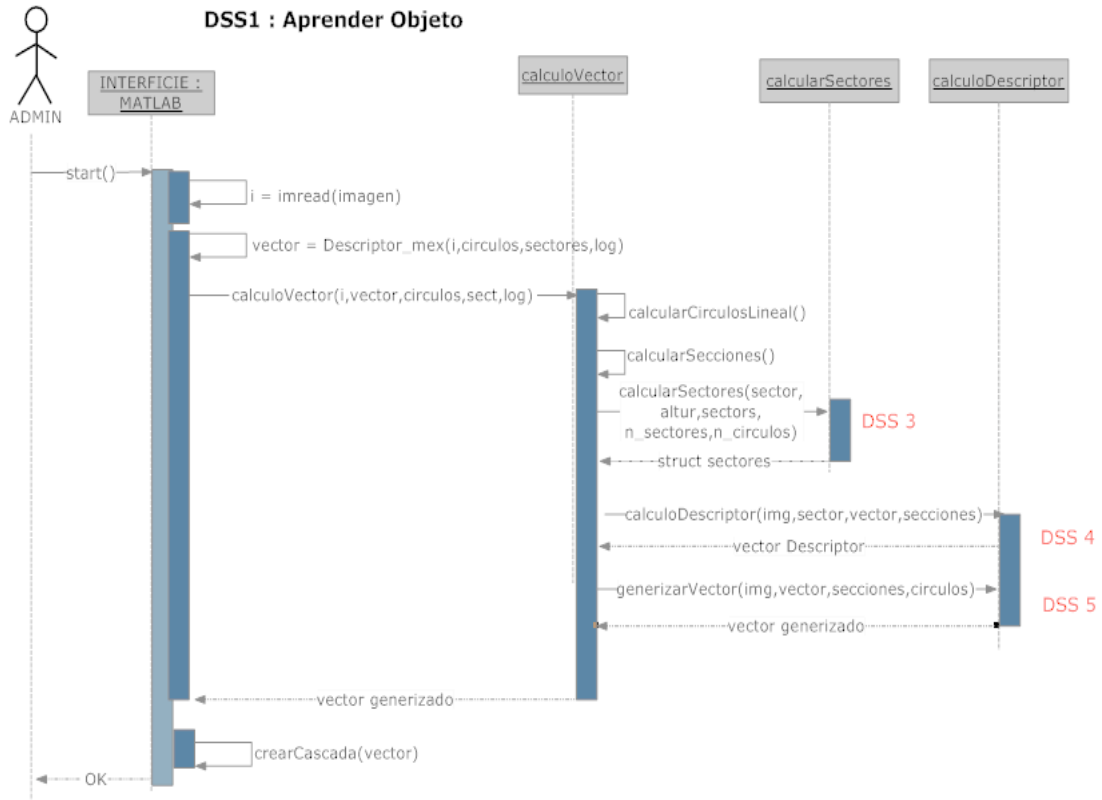


Imagen 3.11: DSS1

En el diagrama (3.12) mostramos los pasos que se producen en el sistema cuando ejecutamos el Descriptor BSM circular. Como he comentado anteriormente, hemos creado 2 ejecutables:

- Descriptor_Objetos_Consola: Contiene un menú para utilizar la aplicación. La imagen es el único parámetro que se introduce al ejecutar.

- Descriptor_Objetos_Param: Es una aplicación más directa, en la que se pasan todos los parámetros por consola en el momento de la ejecución.

Hay que tener en cuenta que la única diferencia entre ambos se encontraría en el main, lo que permite compartir el resto de clases y que el diagrama (3.12) sea común.
Si observas la imagen aparecen 3 notas en rojo que indican el diagrama de secuencia en el que se especifica el método que la contiene.

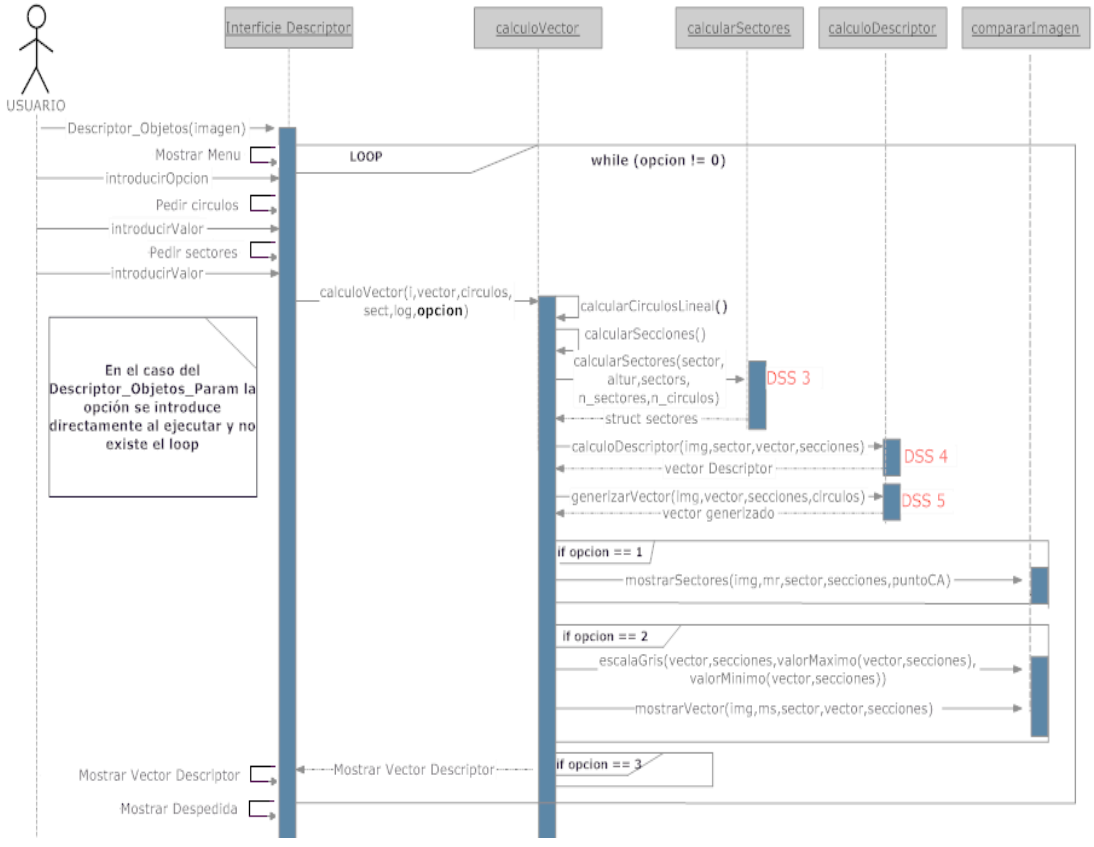


Imagen 3.12: DSS2 Descriptor Objetos

En el diagrama (3.13) nos centramos en la parte de seccionar la imagen, en función de los parámetros introducidos. Esta tarea será común, en el proceso de aprendizaje y en el de descripción, ya que la descripción es una de las partes del proceso de aprendizaje.

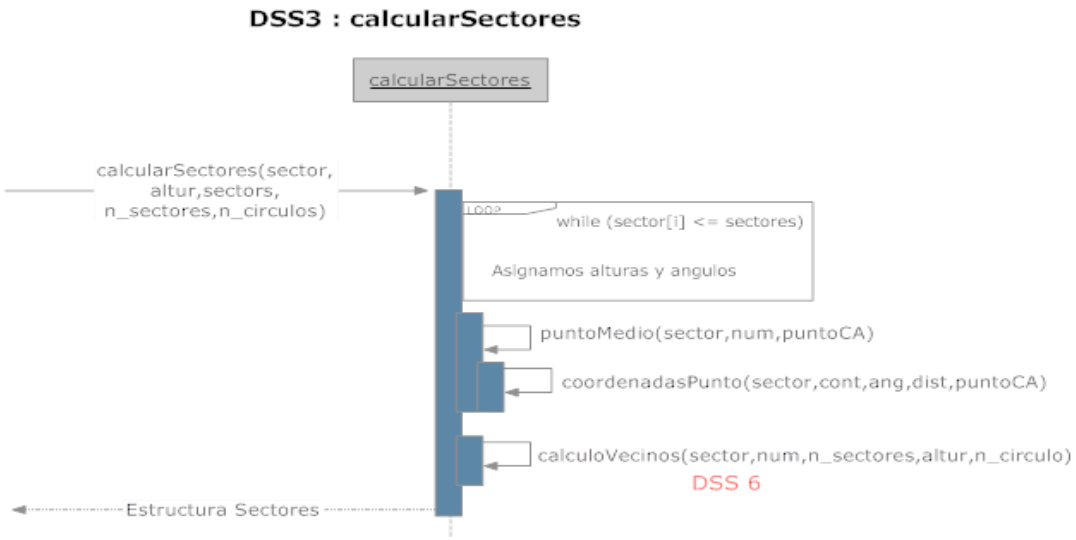


Imagen 3.13: DSS3

En el diagrama (3.14) nos centramos en la parte de descripción de objeto. Como se puede observar el sistema recorre cada píxel buscando aquellos que pertenezcan al contorno del objeto y obteniendo su sector. En el caso de ser cierto, llamamos al método rellenarVector(), de esta manera conseguimos que todos los píxeles del objeto contribuyan en el resultado final del vector.

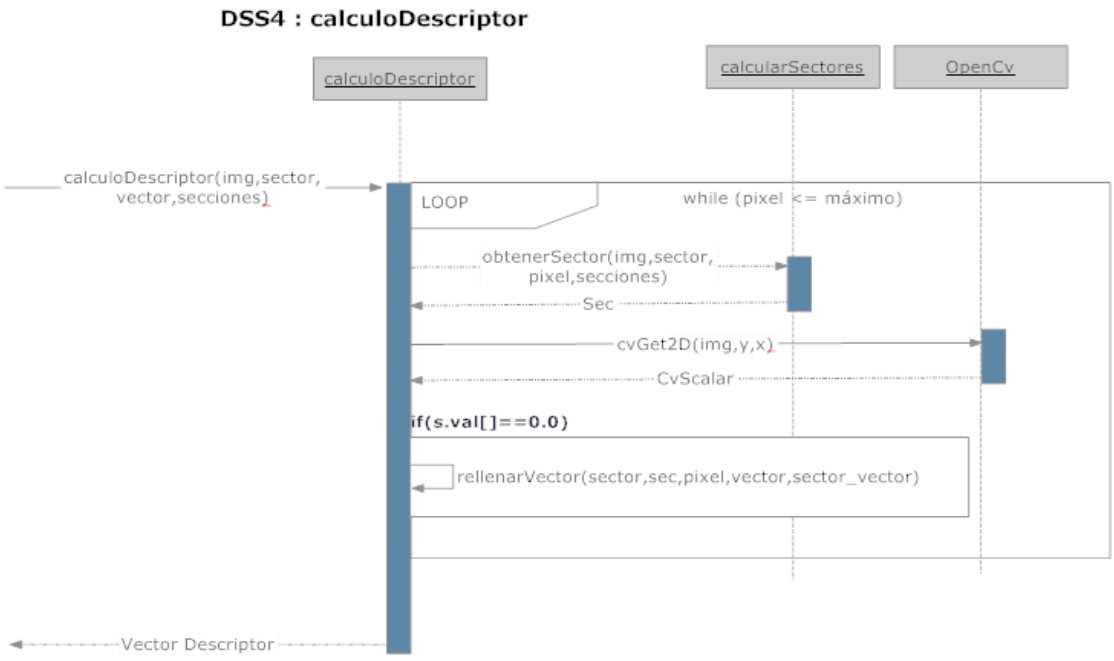


Imagen 3.14: DSS4

En el diagrama (3.15) podemos observar que el método esta relacionado con la generalización del vector. Este método nos permitirá invariancia frente a rotación a la hora de describir el objeto. Se basa en el cálculo de la diagonal principal del objeto y la rotación respecto a esta.

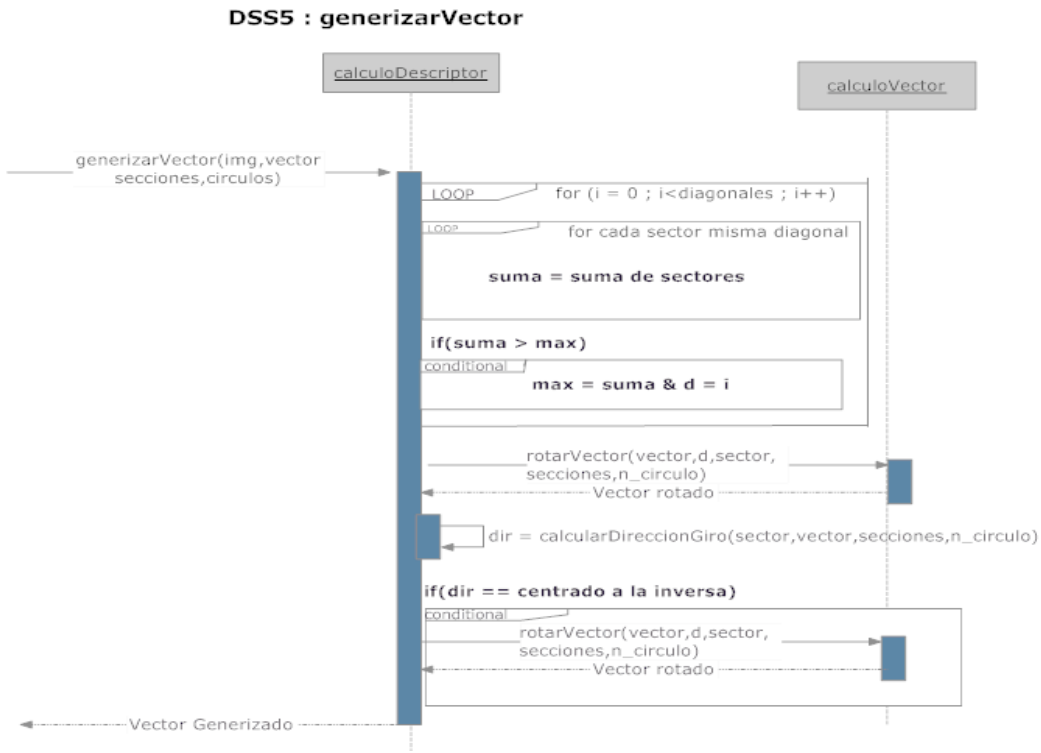


Imagen 3.15: DSS 5

Por último en el diagrama (3.16) mostramos los pasos necesarios para calcular los vecinos de cada sector. Como queremos que la aplicación sea robusta frente a puntos de vista, necesitamos que la influencia de cada píxeles se produzca en todos sus sectores vecinos. Para poder realizar esta tarea necesitamos saber con antelación las vecindad entre sectores.

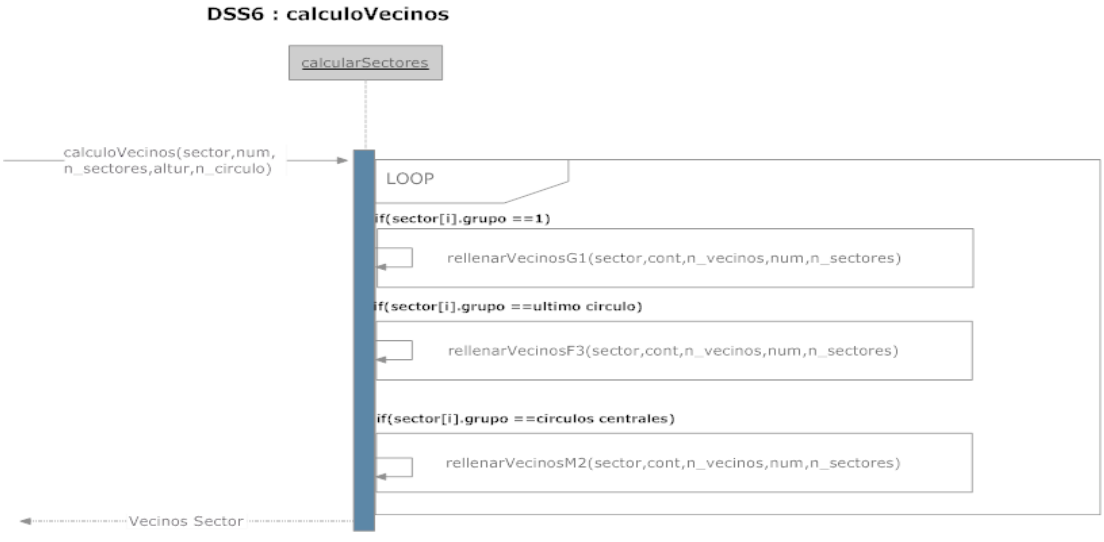


Imagen 3.16: DSS 6

4. Resultados

En este apartado vamos a indicar todos los datos utilizados en los procesos de pruebas de las diferentes aplicaciones del descriptor BSM circular, así como los métodos, parámetros y experimentos necesarios para comprobar el correcto funcionamiento de la aplicación. El apartado lo vamos a distribuir en diferentes secciones. Cada sección contiene los resultados obtenidos en cada una de las aplicaciones en las que interviene el descriptor BSM circular.

4.1 Sección Imágenes

Para realizar las pruebas donde comprobamos si la aplicación secciona correctamente las imágenes hemos utilizado el ejecutable Descriptor_Objetos_Param.exe. Este ejecutable muestra una imagen seccionada si le introducimos la Opción 1.

4.1.1 Parámetros

Los parámetros pasados al ejecutable para seccionar imágenes son:

Experimento A:

- Pasamos como primer parámetro la silla1.jpg de tamaño *524x462* píxeles.
- El número de círculos lo variaremos entre 6 y 10.
- El número de sectores también varia entre 6 y 10.
- Por último introducimos 1 en el parámetro opción, para que nos muestre la imagen seccionada.

Experimento B:

- Pasamos como primer parámetro la sillab.jpg de tamaño *381x553* píxeles.
- El número de círculos lo variaremos entre 6 y 10.
- El número de sectores también varia entre 6 y 10.
- Por último introducimos 1 en el parámetro opción, para que nos muestra la imagen seccionada.

4.1.2 Experimentos

• **Experimento A**: Seccionamos una imagen donde el número de píxeles de su altura es superior a la anchura. Utilizamos diferentes combinaciones de círculos y sectores para comprobar la robustez. Las imágenes obtenidas en la aplicación se muestran en Imagen 4.1 e Imagen 4.2.

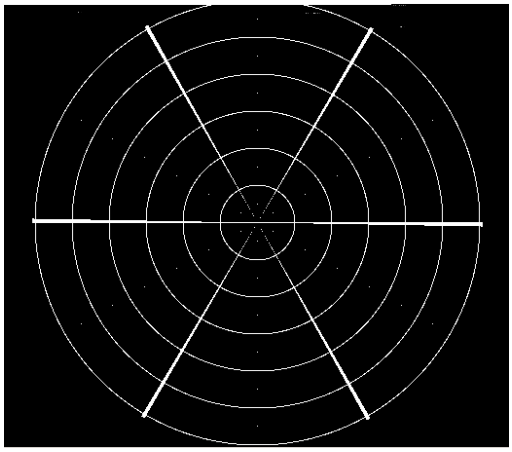


Imagen 4.1: Imagen silla1 seccionada en 36 sectores

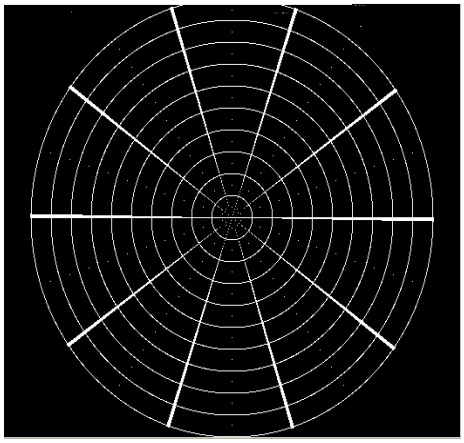


Imagen 4.2: Imagen silla1 seccionada en 100 sectores

Podemos comprobar en las imágenes 4.1 y 4.2 que el descriptor BSM circular, secciona correctamente cualquier imagen independientemente del número de sectores necesarios. Observamos que muestra también los puntos medios de cada sector que serán útiles para crear el vector descriptor, y que denominamos centroide.

• **Experimento B**: Seccionamos una imagen donde el número de píxeles de su altura es inferior a la anchura. Utilizamos diferentes combinaciones de círculos y sectores para comprobar la robustez. Las imágenes obtenidas en la aplicación se muestran en Imagen 4.3 e Imagen 4.4.

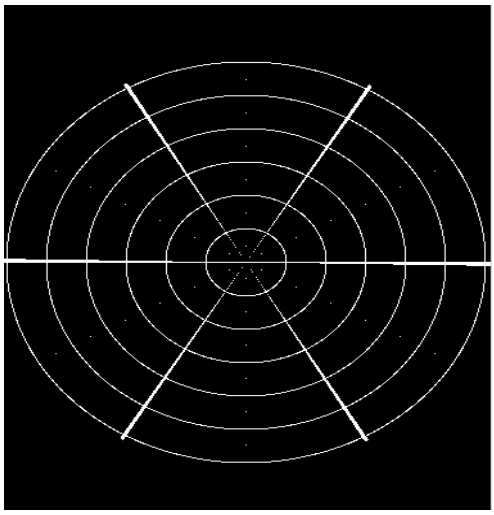


Imagen 4.3: Imagen sillab seccionada en 36 sectores

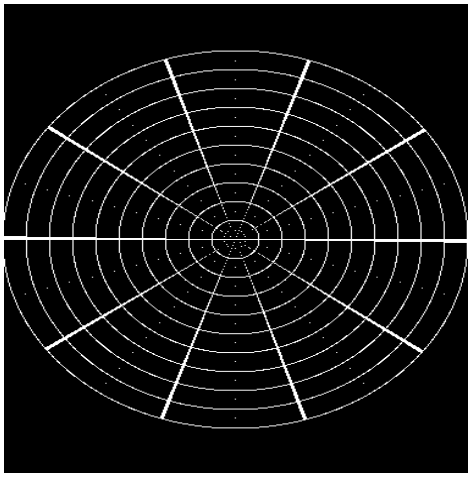


Imagen 4.4: Imagen sillab seccionada 100 sectores

Podemos observar en las imágenes que el descriptor BSM circular secciona cualquier imagen independientemente de su tamaño, porque controla si el ancho de la imagen es superior a su altura.

4.1.3 Cuantificación

Mirando los resultados obtenidos en las 4 imágenes mostradas (Imagen 4.1, 4.2, 4.3 y 4.4) y teniendo en cuenta que hemos realizados infinidad de pruebas con todo tipo de imágenes, podemos considerar que el 100% de las imágenes introducidas en el descriptor BSM circular se seccionan correctamente.

4.2 Descriptor Objetos

Mostraremos los resultados obtenidos en el proceso de pruebas del descriptor de objetos BSM circular. Para comprobar el correcto funcionamiento del descriptor BSM circular hemos utilizado los 2 ejecutables creados. Estos ejecutables son el

Descriptor_Objetos_Consola.exe y Descriptor_Objetos_Param.exe, y se diferencian únicamente en la manera de introducir los parámetros.

Hemos utilizado 2 conjuntos de imágenes en formato jpg. El primer conjunto esta formado por imágenes que contienen contornos de sillas, colocadas de diferentes maneras y en grados de inclinación distintos. Y el segundo conjunto esta formado por imágenes que contienen diversidad de puertas extraídas de planos. Los objetos se encuentran centrados en todas las imágenes sobre un fondo blanco.

4.2.1 Parámetros

Los parámetros utilizados para la descripción de objetos son:

Descripción Sillas

- Pasamos como primer parámetro 3 sillas diferentes para comprobar si la descripción es la misma. Utilizamos la silla1.jpg, silla3.jpg y silla5.jpg.
- El número de círculos será de 30.
- El número de sectores será de 40.
- Por último introducimos un 2 en el parámetro opción, para mostrar el vector descriptor de forma visual.

Descripción Puertas

- Pasamos como primer parámetro 3 puertas diferentes para comprobar si la descripción es la misma. Utilizamos la puerta1.jpg, puerta3.jpg y puerta5.jpg.
- El número de círculos será de 30.
- El número se sectores será de 40.
- Por último introducimos un 2 en el parámetro opción, para mostrar el vector descriptor de forma visual.

4.2.2 Experimentos

En este apartado se muestran algunos de los experimentos realizados y los resultados obtenidos en el proceso de descripción.

4.2.2.1 Descripción Sillas

Describimos 3 sillas vistas desde el lateral de diferente tamaño y que se encuentran posicionadas en la imagen con diferente grado de inclinación. Mostramos las imágenes antes de ser descritas y el vector descriptor visual que se obtiene de cada una de las imágenes.

La primera imagen contiene una silla colocada en su posición correcta como muestra la Imagen 4.5. Su descripción visual la podemos ver en la Imagen 4.6.

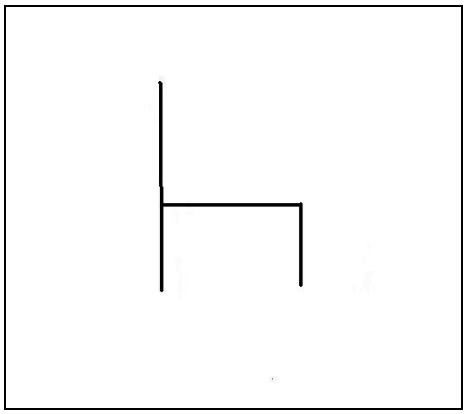


Imagen 4.5: Silla1 original

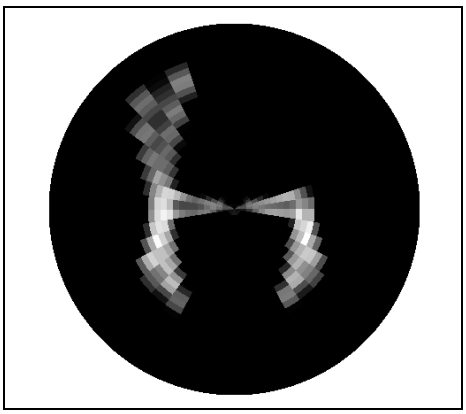


Imagen 4.6: Descripción silla1

Podemos comprobar en la Imagen 4.6 que cuando mostramos la descripción visual de la Imagen 4.5 aparece su contorno difuminado en la posición correcta. De esta manera podemos comprobar que el valor de los píxeles no influye únicamente en los sectores donde reside; lo hace también en los sectores vecinos.

La segunda imagen contiene una silla colocada en su posición inversa como muestra la Imagen 4.7. Su descripción visual la podemos ver en la Imagen 4.8.

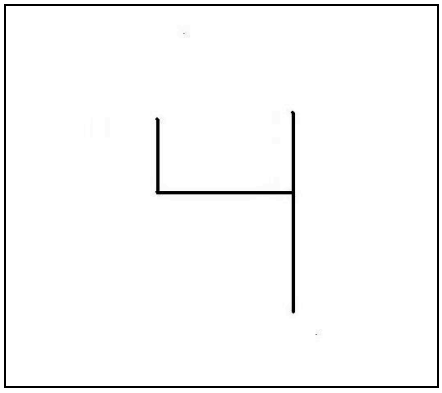


Imagen 4.7: Silla 3 original

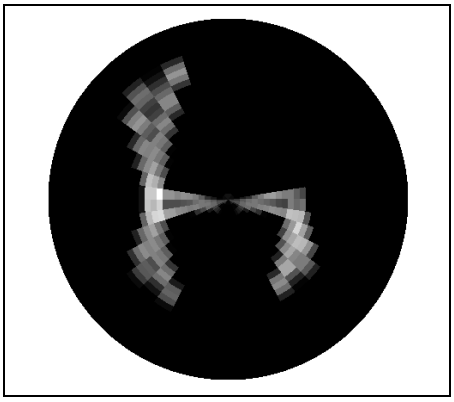
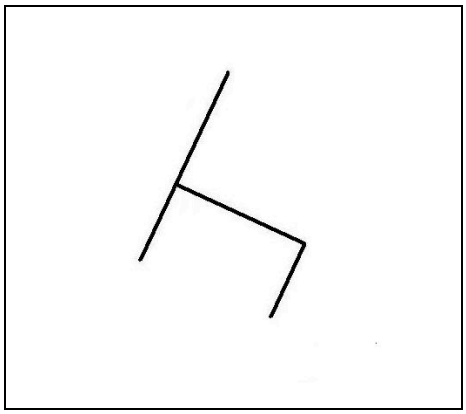


Imagen 4.8: Descripción silla 3

Podemos observar en la imagen 4.8 que la descripción visual de la silla 4.7 colocada a la inversa es prácticamente igual que la descripción 4.6 correspondiente a la silla colocada correctamente. Esto nos indica que el descriptor rota perfectamente.

La tercera imagen contiene una silla inclinada 30 grados como muestra la Imagen 4.9 y su descripción visual la podemos ver en la Imagen 4.10.



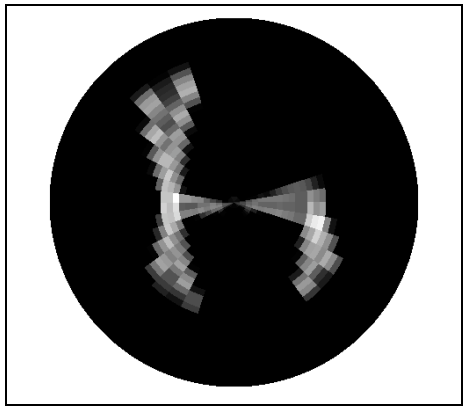


Imagen 4.9: Silla 5 original

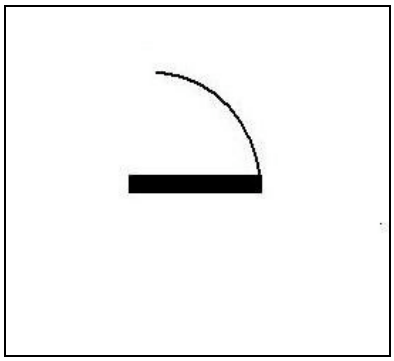
Imagen 4.10: Descripción silla 5

Podemos observar en la imagen 4.10 que la descripción de la silla inclinada 30 grados coincide con las 2 anteriores descripciones. Por lo tanto podemos decir que el descriptor BSM circular es invariante a rotación.

4.2.2.2 Descripción Puertas

Describimos 3 puertas de diferente tamaño que se encuentran posicionadas en la imagen con diferente grado de inclinación. Mostramos las imágenes antes de ser descritas y el vector descriptor visual que se obtiene de cada una de las imágenes.

La primera imagen contiene una puerta colocada en posición horizontal respecto el eje de las X como muestra la Imagen 4.11 y su descripción visual la podemos observar en la Imagen 4.12.



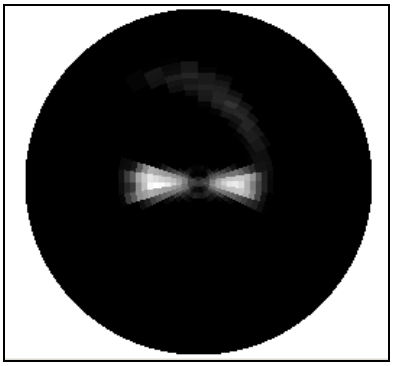


Imagen 4.11: Puerta 1 original

Imagen 4.12: Descripción puerta 1

Podemos observar en la imagen 4.12 que la descripción visual muestra la puerta en la misma posición pero difuminada.

La segunda imagen contiene una puerta rotada 90 grados sobre el eje de las x como muestra la Imagen 4.13 y su descripción visual la podemos observar en la Imagen 4.14.

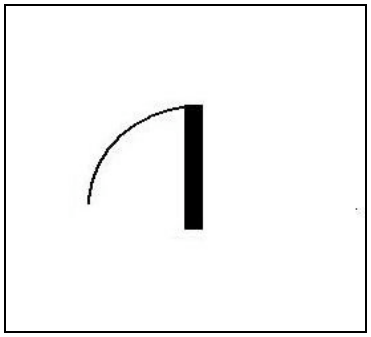


Imagen 4.13: Puerta 3 original

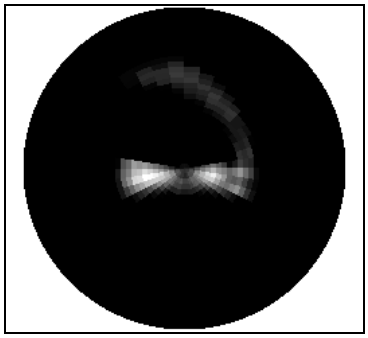


Imagen 4.14: Descripción puerta 3

Observamos como la descripción visual mostrada en la imagen 4.14 es prácticamente igual que la descripción de la puerta anterior que tenía otro grado de inclinación.

La tercera imagen contiene una puerta inclinada 30 grados como muestra la Imagen 4.15 y su descripción visual la podemos observar en la Imagen 4.16.

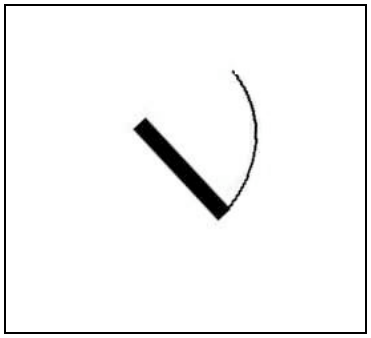


Imagen 4.15: Puerta 5 original

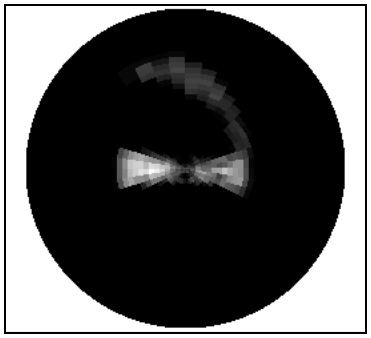


Imagen 4.16: Descripción puerta 5

Podemos observar en la imagen 4.16 que volvemos a obtener la misma descripción visual que con las anteriores puertas. Por lo tanto podemos decir que el descriptor BSM circular es invariante a rotación.

Estas son las pruebas realizadas con el descriptor BSM circular para comprobar su correcto funcionamiento, observando si cumple los requisitos mínimos comentados en el apartado **3**.

4.3 Utilización DescriptorDll

Hemos creado una dll para poder utilizar el descriptor BSM circular desde otras aplicaciones software. Para comprobar su correcto funcionamiento hemos realizado los siguientes pasos:

- Crear un nuevo proyecto visual Basic llamado DescriptorPrueba.
- Incluir la librería DescriptorDll.lib.
- Añadir la dll del descritor de objetos BSM circular.
- Implementar una clase main que realice una llamada al método calcularVector e imprima el vector descriptor por pantalla.

Los parámetros introducidos en el método son:

- Como primer parámetro la imagen silla1.jpg
- El número de círculos utilizados es 6.
- El número de sectores utilizados es 6.
- Por último introducimos 1,2 o 3 en el parámetro opción, para comprobar que funcionan las 3 utilidades de la aplicación.

Para ver si funciona correctamente hemos ejecutado el proyecto con diferentes imágenes y en el 100% de los casos nos ha impreso el vector descriptor correctamente.

4.4 Aprendizaje

En esta sección se describen los datos que son necesarios para poder realizar el proceso de entrenamiento, así como los experimentos realizados y resultados obtenidos. Entenderemos por imágenes positivas, aquellas que se corresponden con el objeto original y negativas el resto.

4.4.1 Parámetros

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.

4.4.2 Experimentos

En este apartado se muestran algunos de los experimentos realizados y los resultados obtenidos en el proceso de aprendizaje.

4.4.2.1 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 4.17 y la otra se ha entrenado para detectar puertas como la de la Imagen 4.18 en un mapa.

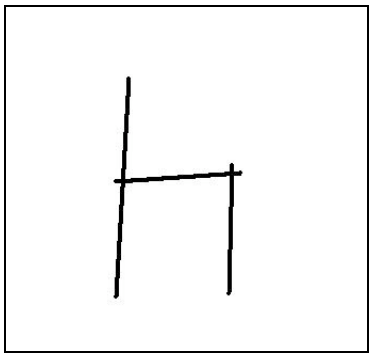


Imagen 4.17: Silla utilizada para entrenar el clasificador de sillas

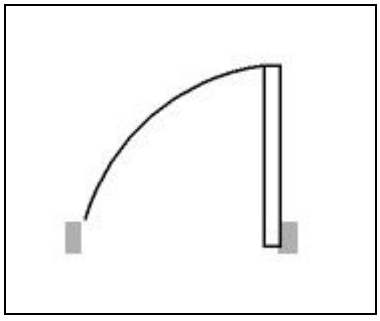


Imagen 4.18: Puerta utilizada para entrenar el clasificador de puertas

4.4.2.2. 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 4.19. 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.

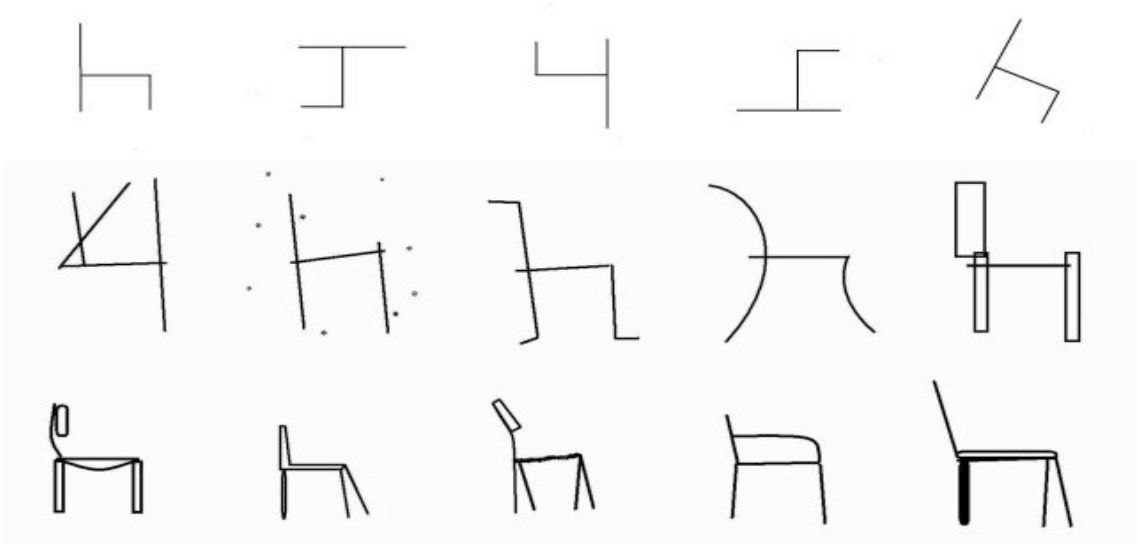


Imagen 4.19: Modelos de sillas utilizadas para entrenar el clasificador

4.4.2.3. 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 4.20). Se ha obtenido una cascada de clasificadores de 5 niveles.

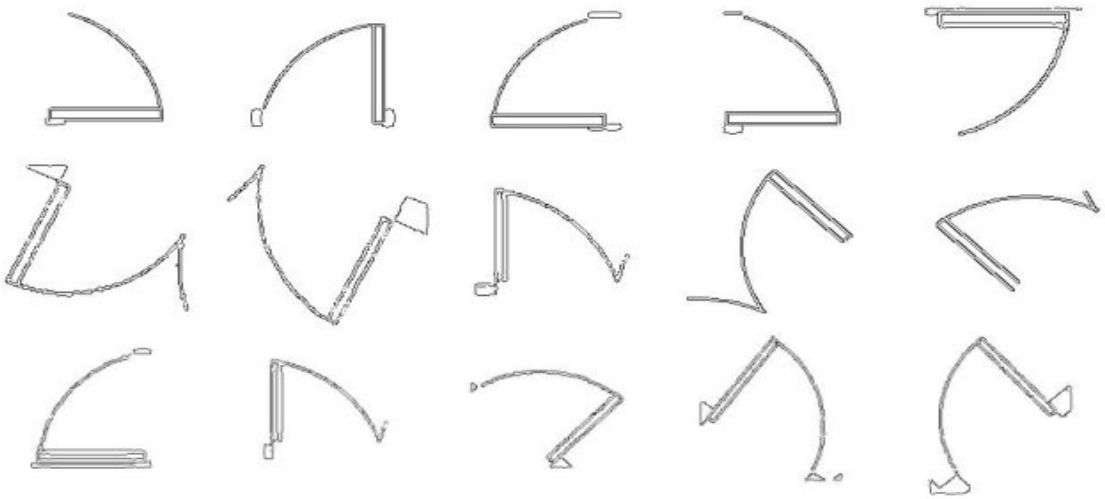


Imagen 4.20: Modelos de puertas, con el método Canny aplicado, utilizadas para entrenar el clasificador

4.5 Detección Objetos

Para detectar objetos en una imagen utilizamos la interficie del detector BSM creada por Alberto Escudero. Esta interficie trabaja con las cascadas entrenadas y el descriptor BSM circular que hemos creado.

4.5.1 Parámetros

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.

4.5.2 Experimentos

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 4.21). En esta interfaz podemos indicar la imagen que deseamos analizar, el nombre que le daremos a la imagen de salida y un conjunto de parámetros que influenciarán en la velocidad de procesamiento de la aplicación.

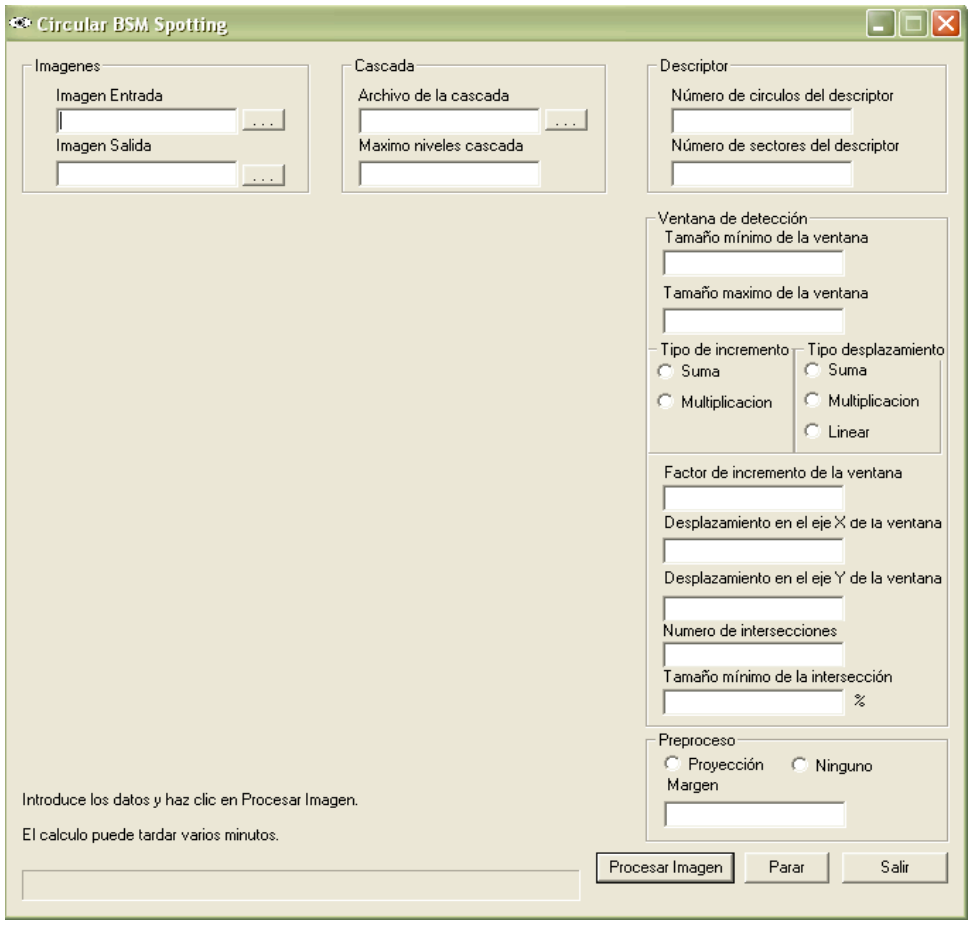


Imagen 4.21: Interfaz programada para la introducción de los datos necesarios para el funcionamiento del detector

4.5.2.1 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

Para la detección obtenida en la Imagen 4.22 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 una intersección del 40% para considerar la detección como positiva.

- No se ha utilizado ningún tipo de preproceso.

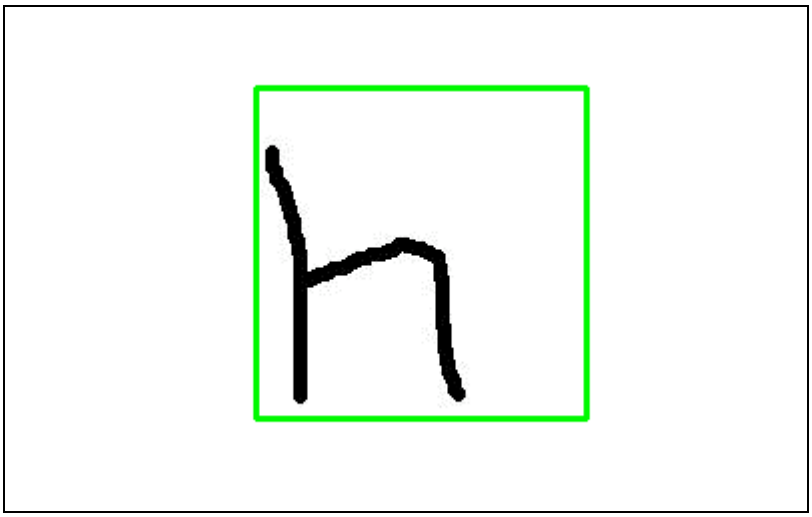


Imagen 4.22: Detección de una única silla

Detección de varias sillas sin preproceso en la imagen

Se han aplicado los siguientes valores:

- Un tamaño mínimo de ventana de 160 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 ésta.

- Fijamos un mínimo de 5 intersecciones del 70%.

- No se ha utilizado ningún tipo de preproceso.

- Con estos datos hemos obtenido la Imagen 4.23.

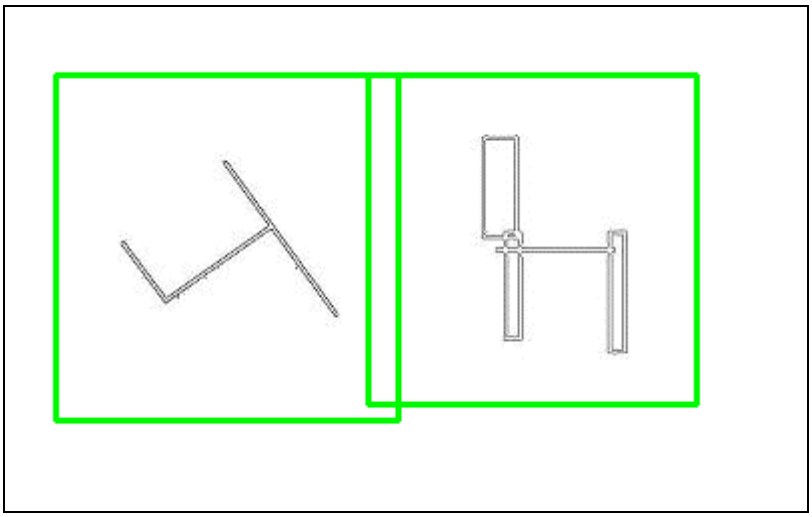


Imagen 4.23: Detección de varias sillas en una misma imagen

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 4.24.

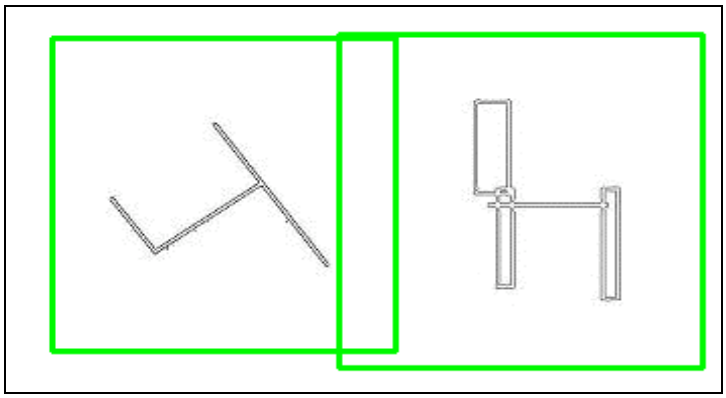


Imagen 4.24: Detección de varias sillas en una misma imagen aplicando el preproceso de proyección

Detección de 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 4.25:

- 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 ésta.
- Fijamos un mínimo de 3 intersecciones del 40%.
- No se ha utilizado ningún tipo de preproceso.

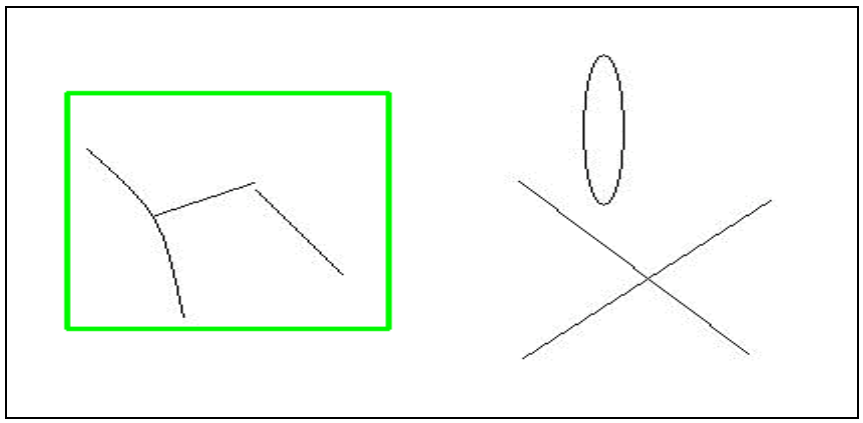


Imagen 4.25: Detección de una silla con otro objeto en la imagen. El objeto tiene cierta similitud con una silla y fue necesario incrementar el número de intersecciones para eliminar el falso positivo

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.

4.5.2.2. 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 4.26 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.

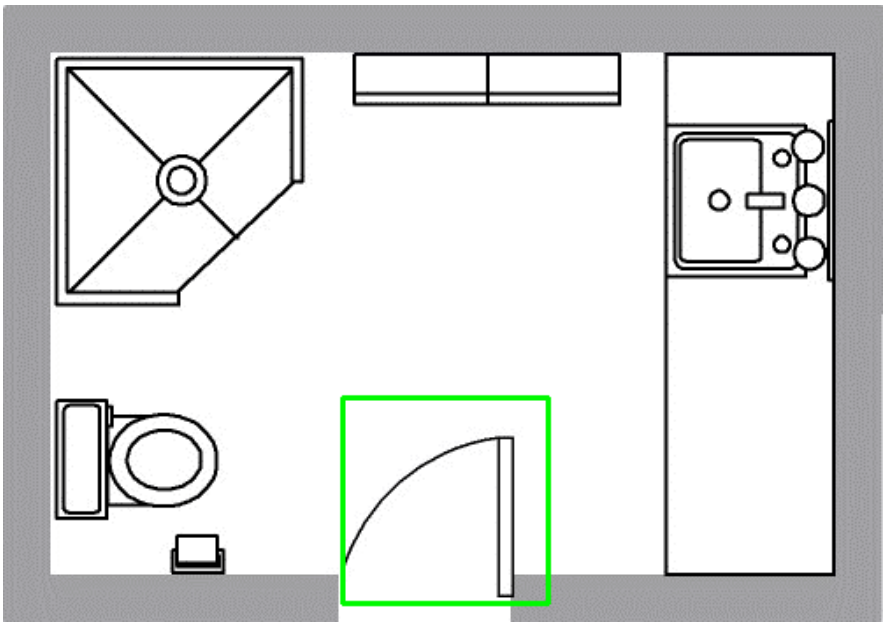


Imagen 4.26: Detección de una puerta en un plano

En la Imagen 4.27 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.

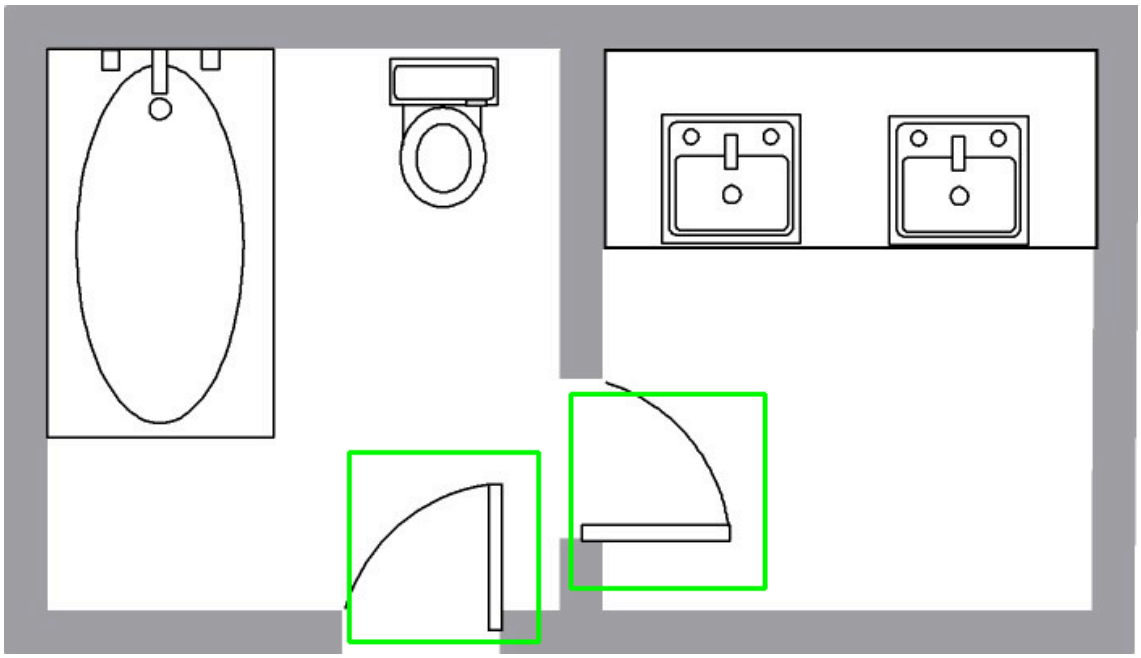


Imagen 4.27: Detección de varias puertas con diferente orientación

En la Imagen 4.28 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.

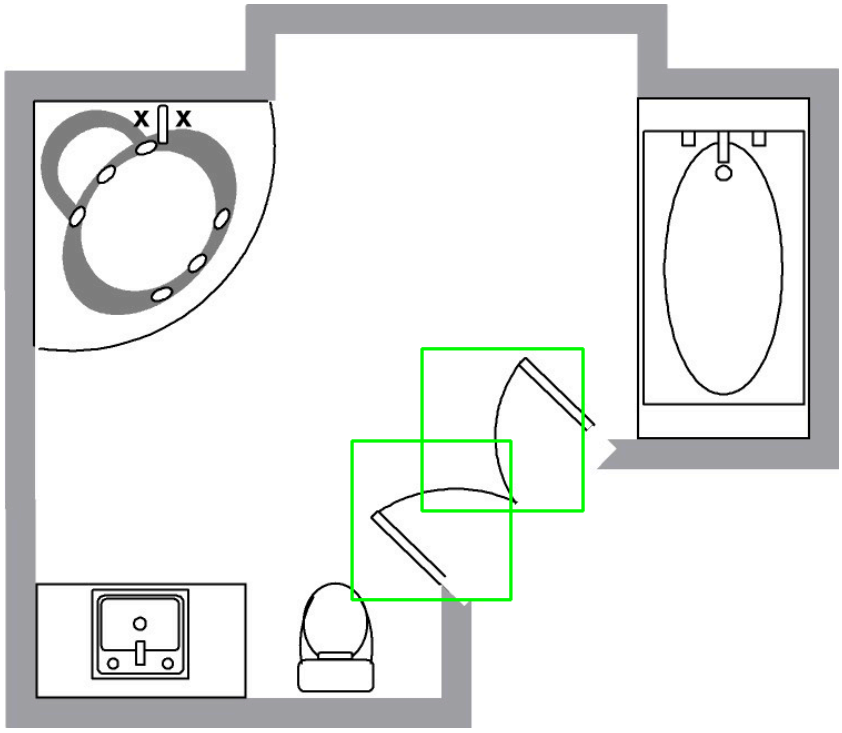


Imagen 4.28: Detección de varias puertas con diferente sentido de apertura

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 en este caso el número de falsos positivos ha sido mayor, dado el gran número de elementos que componen un plano.

5.Conclusiones

En este proyecto hemos desarrollado un software en lenguaje visual C++ capaz de resolver de forma eficaz el problema de la descripción de objetos en imágenes.
En este apartado comentaremos las conclusiones extraídas durante el transcurso del mismo. Dividiremos las conclusiones en dos secciones, cada una de las cuales corresponde a las diferentes utilidades de la aplicación.

5.1 Sección Imágenes

Para el proceso de seccionar imágenes hemos creado un método capaz de distribuir las dimensiones de la imagen cuantitativamente y formar una plantilla circular dividida en sectores. A diferencia del BSM creado por Sergio Escalera, utilizamos un descriptor circular porque es la forma más sencilla de conseguir robustez frente a la rotación de los objetos. Consideramos dos técnicas para calcular los círculos concéntricos de la plantilla, pero la distribución lineal es más efectiva que la distribución exponencial. Esto se puede observar al seccionar imágenes utilizando círculos concéntricos exponenciales porque crea muchos círculos cercanos al eje central y pocos ocupando el resto de la imagen. Debido a esta mala distribución a la hora de describir el objeto, la gran mayoría de píxeles pertenecerían al círculo exterior, y la descripción no sería eficaz.

Por otro lado, el cálculo de sectores vecinos no se podía generalizar para cada sector porque variaba en función del grupo al que pertenecía (central, medio o periférico), y supuso la creación de tres algoritmos diferentes.
Teniendo en cuenta estos factores, hemos creado un algoritmo capaz de seccionar cualquier tipo de imagen independientemente de sus dimensiones, con una eficacia del 100%. Esto se puede observar en el apartado de Resultados, donde mostramos imágenes seccionadas correctamente.

5.2 Descriptor Objetos

El proceso de calcular el vector descriptor depende totalmente de la distribución correcta de los sectores en la imagen y el cálculo de sus vecinos.
Basándonos en el descriptor BSM de Sergio Escalera, hemos creado un método que se basa en la estructura del objeto para describirlo. Con esto se consigue, además de ser robusto frente a cambios de iluminación y puntos de vista, robustez frente a deformaciones elásticas y distorsiones no uniformes. Para realizar un cálculo correcto a la hora de describir un objeto en la imagen, necesitamos un previo filtrado utilizando el método Canny. Este método filtra la imagen original hasta obtener el contorno de los objetos que contiene.
La técnica utilizada consiste en obtener aquellos píxeles de la imagen que pertenecen al contorno del objeto y guardar mediante un algoritmo creado su peso, tanto en el sector al que pertenece como en sus sectores vecinos. De esta manera conseguimos que la descripción sea robusta frente a deformaciones rígidas o elásticas.

5.3 Generalizar Vector

Una vez creado el vector descriptor, nos encontramos con el problema que para cada objeto de la misma familia, si estaba colocado con un grado de inclinación diferente del resto, su descripción no era correcta.
Para solucionar el problema creamos un método capaz de calcular las diagonales de mayor densidad de la imagen. Teniendo en cuenta que la diagonal principal sería la misma que la del objeto, rotamos en consecuencia. De esta manera conseguíamos obtener un vector descriptor invariante a rotación, que como podemos observar en el apartado de resultados 4.2 ofrece una eficacia del 100% sobre los datos testeados.

5.4 Descriptor Visual

Para comprobar si las descripciones que hacia el descriptor BSM circular eran correctas, tuvimos que representar estos valores en una nueva imagen. Debido a que estos valores estaban en un rango entre 0 y 1, creamos un método para obtener una escala de grises entre 0 y 255. Éstos son los valores necesarios para poder representar el objeto en una imagen a escala de grises.
Podemos comprobar que para cada objeto de la misma familia, independientemente de la colocación y grado de inclinación, la descripción visual obtenida es común.

5.5 Conclusiones Generales

Después de un proceso de elaboración, implementación y pruebas, hemos conseguido crear un software capaz de describir cualquier tipo de objeto encontrado en una imagen.
En el proceso de aprendizaje de objetos necesitábamos crear una aplicación compatible con Matlab para ser utilizado desde el clasificador. Esto supuso crear una nueva interficie denominada Mex-File, que podemos encontrar como parte del software.
Uno de los requisitos fundamentales para que la descripción sea 100% fiable, es que el objeto debe estar centrado en la imagen. Esta conclusión es de vital importancia en el proceso de detección de objetos creado por Alberto Escudero, porque deberá recorrer la imagen utilizando ventanas y desplazamientos, hasta conseguir centrar el objeto en una o varias de ellas. El descriptor BSM circular no sólo es robusto frente a cambios de iluminación, puntos de vista y deformaciones rígidas o elásticas, sino que además, asegura robustez frente a rotación gracias a la utilización de un descriptor circular que permite rotar sectores y alinear objetos similares.

6.Bibliografía

[1] David G. Lowe, "Object recognition from local scale-invariant features," International Conference on Computer Vision, Corfu, Greece (September 1999), pp. 1150-1157.

[2] Y. Ke, R. Sukthankar, *PCA-SIFT: a more distinctive representation for local image descriptors*, Proc. CVPR , pp. 511–517 (2004).

[3] Herbert Bay, T. Tuytelaars and L. Van Gool "SURF: Speeded Up Robust Features", Proceedings of the 9th European Conference on Computer Vision, Springer LNCS volume 3951, part 1, pp 404--417, 2006.

[4] David G. Lowe. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 60(2):91–110, November 2004.

[5] S. Escalera, A. Fornés, O. Pujol, J. Lladós and P. Radeva, Progress in Pattern Recognition, Image Analysis and Applications, November 2007.

• A continuación mostramos los enlaces que han sido útiles para la creación del proyecto. De estos enlaces he extraído información sobre la Vision Artificial, Descriptor SIFT, PCA-SIFT y SURF.

http://es.wikipedia.org/wiki/Computer_Vision

http://en.wikipedia.org/wiki/Scale-invariant_feature_transform

http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

http://leibniz.iimas.unam.mx/~yann/Vision/Clase04.pdf

http://www.vision.ee.ethz.ch/~surf/eccv06.pdf

https://www.u-cursos.cl/ingenieria/2008/1/EL708/1/material_docente/objeto/162393

• Los siguientes enlaces son útiles para entender el uso de las librerías OpenCv.

http://en.wikipedia.org/wiki/OpenCV

http://eupt2.unizar.es/cmedrano/tutorial_opencv.pdf

• Enlaces necesarios para la creación del Mex-File

http://cnx.org/content/m12348/latest/

http://www.codeproject.com/KB/cpp/mexFunction.aspx

7. Apéndices

Apéndice A: CD Descriptor Objetos BSM circular

A continuación adjuntamos el CD que contiene el proyecto software completo, las imágenes utilizadas en el proceso de pruebas y la memoria en formato pdf.

El contenido del CD lo hemos estructurado en diferentes carpetas. Cada una de las carpetas contiene los archivos necesarios para su uso independiente del resto de utilizades. La estructura del CD es la siguiente:

1) **Descriptor_Objetos:** Este directorio contiene el software necesario para describir objetos encontrados en imágenes. Se divide a su vez en 2 subdirectorios:
    i. Código Fuente: Contiene los proyectos software con el código fuente bien documentado.

ii. Ejecutables: En este directorio encontramos los 2 ejecutables que hemos creado para describir objetos, un Manual de Usuario y una carpeta con todas las imágenes que podemos utilizar en la aplicación.

2) **LibreríaDescriptor:** Este directorio contiene el software necesario para crear la librería del proyecto descriptor, además de la propia librería y el fichero DescriptorDll.dll. Se divide a su vez en 2 subdirectorios:

   i. Código Fuente: Contiene el proyecto software necesario para crear la librería DescriptorDll.

   ii. SoftwarePrueba: Carpeta donde encontramos el proyecto creado para comprobar el correcto funcionamiento de la dll. En este directorio tenemos también el Manual de usuario útil para usar la aplicación y una carpeta con imágenes.

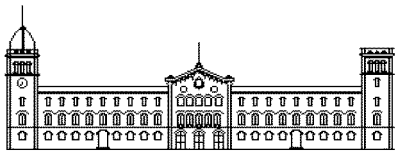
3) **MexFile:** En este directorio encontramos el código fuente del proyecto software necesario para crear la librería MexFile. Esta librería formará parte del proceso de aprendizaje en el proyecto de Alberto Escudero. Además la carpeta contiene la librería Descriptor_Mex.dll, un Manual de Usuario y una imagen, útiles para utilizar la librería desde Matlab.

4) **Detección_Objetos:** Este directorio contiene la interficie Circular BSM Spotting que utilizamos en el proceso de detección de objetos en imágenes. También contiene todas las librerías que necesita la aplicación, 2 cascadas de entrenamiento, un Manual de Usuario y 2 carpetas para almacenar la imágenes originales y las de salida de la aplicación.

5) **Software_Adicional:** En esta carpeta podemos encontrar los ficheros de instalación para Windows de las librerías OpenCV e IPL.

6) **Memoria:** En este directorio encontramos la memoria del proyecto Descriptor de Objetos basado en Forma en formato pdf.

7) **Comprimidos:** Hemos incluido en el CD una carpeta que contiene todos los proyectos presentados anteriormente en formato zip.



ENGINYERIA TÈCNICA EN INFORMÀTICA DE SISTEMES
UNIVERSITAT DE BARCELONA

Treball fi de carrera presentat el dia de de 200
a la Facultat de Matemàtiques de la Universitat de Barcelona,
amb el següent tribunal:

Dr. President

Dr. Vocal 1

Dr. Secretari

Amb la qualificació de: