SketchSearch

Matías Eric Viglione Muñoz

Resumen—Cada día las bases de datos de imágenes son más grandes, lo que provoca que millones y millones de imágenes se acumulen y el hecho de buscar entre ellas se vuelve una tarea realmente complicada. Durante años diferentes métodos de búsqueda han ido surgiendo para intentar facilitar este trabajo, tales como búsqueda por texto o búsqueda por imagen. Este proyecto propondrá una forma distinta de buscar entre ellas, estamos hablando de la búsqueda por bocetos. SketchSearch, es una aplicación que realiza la búsqueda de imágenes a partir de un boceto realizado por el usuario, esto permitirá al usuario más libertad a la hora de buscar entre ellas, dado a que permite ser mucho más concreto a la hora de buscar que otros métodos de búsqueda tradicionales. Esta aplicación permite al usuario una pantalla principal donde se realizará el boceto de la imagen o conjunto de estas que se desea buscar, a continuación, se hará un cálculo basado usando uno de los diferentes algoritmos de reconocimiento de características, tales como Oriented Fast and rotated BRIEF (ORB), ShapeContext y Hausdorff distance, que ofrece la aplicación. Esto llevara a una galería donde aparecerán las imágenes ordenadas de forma descendente de más semejante a menos al boceto dibujado.

Palabras claves—Bocetos, Imágenes, Búsqueda, ORB, ShapeContext, Hausdorff

Abstract—Every day the images databases are bigger, which causes millions and millions of images to accumulate and the fact of searching among them becomes a really complicated task. For years different search methods have been emerging to try to facilitate this work, such as text search or image search. This project will propose a different way of searching among them, we are talking about the search by sketches. SketchSearch, is an application that performs the search of images from a sketch made by the user, this will allow the user more freedom when searching among them, given that it allows to be much more specific when looking for others traditional search methods. This application allows the user a main screen where the image or set of images to be searched will be drawn, then a calculation will be made based on one of the different feature recognition algorithms, such as Oriented Fast and rotated BRIEF (ORB), ShapeContext and Hausdorff distance, offered by the application. This will lead to a gallery where the images arranged in descending order will appear, similar to less than the drawn sketch.

Index Terms—Sketch, Images, Search, ORB, ShapeContext, Hausdorff

1 Introducción

CADA día las bases de datos que contienen imágenes son cada vez más grandes, lo que hace que la búsqueda de imágenes concretas se haga un trabajo complicado y tedioso.

Actualmente ya existen métodos de búsqueda que intentan simplificar el problema, pero estos siguen dando resultados limitados y extensos. Lo que se me ha propuesto hacer es una nueva forma de intentar simplificar la búsqueda de imágenes, está en cuestión, consiste en una búsqueda por bocetos.

Se busca conseguir que a partir de un boceto dibujado por el usuario el software sea capaz de encontrar imágenes similares a lo dibujado en el boceto, de esta forma poder hacer una búsqueda de una forma mucho más concreta y si además luego lo intentamos a juntar a otros métodos de búsqueda existentes podríamos terminar haciendo un buscador que simplifique bastante el problema inicial.

- *E-mail: matiaseric.viglione@e-campus.uab.cat*
- *Mención realitzada: Enginyeria de Computació*
- *Tutor: Josep Lladós*
- *Curso 2018/19*

La búsqueda por boceto proporciona mejoras que muchas otras formas de búsqueda no pueden ofrecer, entre ellas la capacidad de hacer búsquedas que permitan al usuario buscar una imagen en concreto a partir del recuerdo que tenga este sobre la imagen, tan solo tiene que dibujar la imagen tal cual la recuerde y el buscador la encontrara.

También puede permitir la búsqueda de imágenes sobre objetos cuyo nombre se desconozca o se desconozca en parte, como podría ser cierto modelo de avión, utensilio de cocina o raza de pájaro.

En las siguientes paginas vamos a ver el progreso que se ha seguido para la creación de tal aplicación. Empezaremos, en la sección 2, con relato breve de los objetivos que se han propuesto cumplir para la finalización de este trabajo. En la sección 3, hablaremos del estado de arte donde cubriremos todos aquellos proyectos que actualmente existen que cumplen con una función similar a la propuesta, con tal de que podemos observar como se encuentra actualmente el campo donde nuestra aplicación estará ambientada. Una vez puestos en situación, en la sección 4 empezaremos ya a hablar de nuestra aplicación, empezando por redactar la arquitectura y el flujo de la aplicación, donde aclararemos la relación que tienen las diferentes clases entre si y como hace la aplicación para llegar del dibujo

a la galería con las imágenes más semejantes, y después se explicara la implementación que lleva esto a cabo. Para finalizar este apartado se hablará de la interfaz que lleva al usuario de pantalla en pantalla hasta cumplir con el objetivo de la aplicación.
En último lugar, en la sección 5 haremos un estudio de la aplicación, pondremos una serie de pruebas a las que se vera obligada a pasar, tanto en nivel de interfaz como de funcionalidad, pasando luego a la sección 6 con una conclusión final donde hablaremos de estos resultados y el porque de ellos, al mismo tiempo de posibles mejoras que se le podrían dar a la aplicación.

2 Objetivos

El objetivo principal del proyecto será el de diseñar y desarrollar una aplicación Android que permita la búsqueda en una base de datos de imágenes a partir de bocetos, *sketch-based image retrieval*. El proyecto se divide en dos subapartados a los cuales explicaremos con más detalle a continuación, una primera sería los objetivos funcionales de la aplicación con la que el usuario tendrá que manejarse con tal de poder realizar búsquedas y un apartado de objetivos tecnológico desde el punto de vista de comparación boceto-imagen.

2.1 Aplicación

En primer lugar, está la aplicación. Esta tendrá varias pantallas y cada una con un objetivo en clave. Las pantallas serian:

- Una pantalla de dibujo, desarrollar una pantalla de dibujo donde el usuario pueda dibujar la imagen que desea buscar.
- Una pantalla de galería, implementar una galería donde se encuentre las imágenes resultados ordenadas según la semejanza al boceto.
- Una pantalla de configuración, la cual debe permitir:
    - Algoritmo de comparación con el que se desea hacer la búsqueda
    - Activar o desactivar la búsqueda por ventana

2.2 Comparación

Este segundo apartado ya se basa en la implementación de los diferentes algoritmos de búsqueda, hasta ahora funciona con Shape Context, ORB y Hausdorff distance. Este apartado ha de poder ejecutar estos tres algoritmos de una forma óptima y que no cause problemas, todo ello para el lenguaje de programación Java el cual no tiene de base una buena forma de procesar imágenes, por lo cual el trabajo se puede llegar a complicar.

3 Estado del Arte

Actualmente ya existen otros proyectos que implementan métodos parecidos al buscado.

Unofficial Google Image Search by Drawing [1], sistema de búsqueda por imagen implementado por Google en lenguaje HTML, el cual envía una petición de búsqueda a un servidor y este le devuelve los resultados usando "Google Search by Image"

Splash [2], aplicación desarrollada por 500px, de búsqueda por esbozo muy parecido al sistema anteriormente descrito, con la diferencia que tiene detección de colores en la imagen. Splash mantiene su propia base de datos de imágenes y no hace una búsqueda exhaustiva por Internet como hace ahora Unofficial Google Image Search by Drawing.

Otras aplicaciones usan un sistema de búsqueda imagen como *Quick Draw* [3] de Google Creative Lab, este proyecto no es una aplicación de comparación de imágenes como las anteriores si no un identificador de dibujos, funciona como un juego, en el cual te dan 20 segundos para que dibujes un objeto propuesto por ellos y dejar que la maquina lo adivine. El interés de este proyecto es que funciona con una red neuronal, la cual aprende de cada dibujo realizado, por lo cual si acierta el objeto lo guardara como futuras referencias y si no lo acierta aprende de este para poder interpretar de más formas el objeto en cuestión.

También existe una gran cantidad de motores que realizan Reverse Image Search, lo cual consiste en una búsqueda de imágenes a partir de imágenes. Estas son algunas de las mejores.

- **Google Image** [4]
  La más conocida con diferencia, y tiene la base de datos de imágenes más grande del mundo. En 2011, Google introdujo una función para realizar Reverse Image Search, funciona con el simple hecho de arrastrar una imagen o copiar el link de esta. Usa diferentes algoritmos tanto de forma, tamaño, color y resolución para comparar la imagen.
- **TinEy**e [5]
  Desarrollado por Idee Inc., es la página más usada para hacer búsqueda de imágenes mediante Reverse Image Search. Tan solo tienes que subir la imagen deseada o proveer el link de esta. Soporta tanto formato PNG, JPG como GIF, menores de 20mb. No reconoce objetos o personas, sino la imagen como un conjunto y tiene extensiones en Firefox, Chrome y Safira.
- **Yandex** [6]
  El buscador más grande de Rusia, contiene un buscador de imágenes Reverse Image Search. Permite subir la imagen vía URL o subiéndola directamente desde el ordenador. También contiene un filtro para filtrar el tamaño de la imagen.
- **Bing Image Match** [7]
  En marzo de 2014, Microsoft creo su propio buscador de imágenes llamado 'Bing Image Match', los usuarios pueden tanto subir la imagen como el hipervínculo a esta y devolverá las imágenes similares. Aun así, los resultados no son ni la mitad de buenos que TinyEye o GoogleImage

4 Metodología

La aplicación Android se ha programado mediante el lenguaje Java, mediante el programa Android Studio.

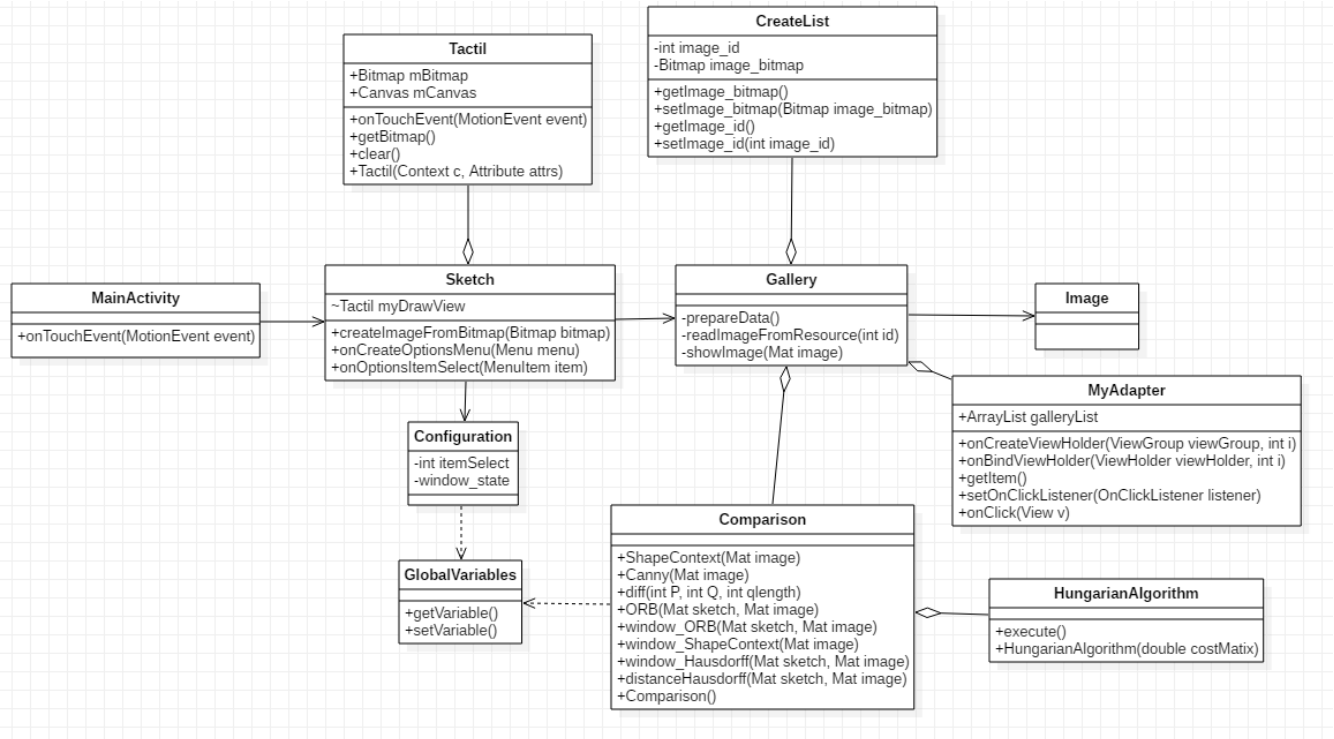


Ilustración 1: Diagrama de clases

Se ha utilizado también la famosa librería de procesamiento de imágenes llamada OpenCV, la cual se ha tenía que instalar ya que no venía de serie.

El trabajo consta de las siguientes clases

- MainActivity, pantalla de inicio de la aplicación
- Sketch, pantalla de dibujo de la aplicación
- Tactil, contiene los comandos para dibujar
- CreateList, lista para implementar la galería
- MyAdapter, adaptador para la lista de imágenes de la galeria
- Gallery, pantalla de la galería y procesado de esta misma
- Image, pantalla de imagen abierta
- Comparison, contiene los algoritmos de comparación implementados
- Configuration, pantalla de configuración de la aplicación
- GlobalVariables, contiene las variables globales de la aplicación
- HungarianAlgorithm, contiene el algoritmo Hungarian utilizado en la comparación de imágenes

4.1 Arquitectura y Flujo

4.1.1 Arquitectura

Como se puede ver en el diagrama de clases, *ilustación 1,* todo empieza en la clase MainActivity la cual da paso a la clase Sketch, esta tiene un objeto de la clase Tactil, la cual le permite dibujar. La clase Sketch accede a la clase Configuration o Gallery, en la clase Configuration modificamos las variables de GlobalVariables, las cuales más tarde serán usadas en Gallery en la instancia del objeto Comparison, en la clase Galley tenemos una lista de objetos CreateList, la cual tendrá los elementos necesarios para crear la galería, esta lista es transformada en la vista de la galería gracias al adpatador MyAdapter, también declarado en Gallry. Luego también tiene un objeto Comparison, el cual nos permite aplicar los algoritmos de comparación a las imágenes, este utilizara las variables de GlobalVariables para saber que algoritmo usar y con que parámetros, y si es búsqueda por ventana o no. Para finalizar Gallery puede acceder a la clase Image para abrir la imagen que se desee.

4.1.2 Flujo

El flujo que se sigue en la aplicación es el siguiente.

- Dibujar boceto
- Para cada imagen en la galería
    - Hacer Canny
    - Aplicar algoritmo de comparación con boceto
    - Listar en diccionario, bitmap de la imagen con distancia al boceto
- Despues ordenamos la lista resultado por distancia, de menor a mayor
- Adaptamos la lista con tal de recibir una vista en forma de galería de las imágenes

Con esto recibimos una galería de imágenes ordenadas de más semejante a menos.

4.2 Implementación

Ahora pasaremos a explicar de forma profunda los algoritmos utilizados para la comparación. Las comparaciones se realizan en la calase Comparison, la cual es llamada por Gallery. Se han utilizado un total de 3 algoritmos de comparación para realizar esta aplicación y un algoritmo de

Edge Detection para tranformar las imágenes a imágenes en líneas semejante a un boceto. Estos son:

4.2.1 Canny Edge detector [8]

Desarrollado por John F. Canny en 1986, es un detector de contornos, con este algoritmo conseguimos que las imágenes terminen con una apariencia mas cercana a la de un boceto, con tal de poder comparar ambos.

Canny sigue el siguiente proceso

1. Aplica un filtro Gaussiano con tal de suavizar y eliminar el ruido
2. Encuentra los gradientes de intensidad de la imagen
3. Aplica la supresión no máxima para deshacerse de la respuesta espuria a la detección de bordes
4. Aplica doble umbral para determinar bordes
5. Restare el borde por histéresis

Para finalizar, no quedara una imagen negra con la imagen dibujada en contornos blancos, como lo que queremos es lo contrario ya que el boceto es blanco con contornos negros, voltearemos los colores de la imagen resultado.

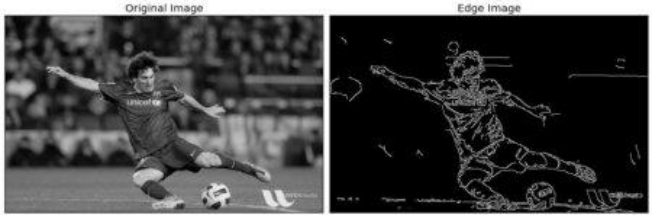


Ilustración 2: Detección de bordes con Canny Edge detector

4.2.2 ShapeContext [9]

Propuesto por Serge Belongie y Jitendra Malik en el 2000, ShapeContext es un descriptor de características usado en el reconocimiento de objetos.

Sa basa en la comparación de siluetas de objetos, su principal idea es la de crear un histograma basado en los puntos en un sistema de coordenadas polar. Entonces, en el contorno de una imagen escoges x números de puntos y para cada punto calculamos su distancia Euclidiana y su Angulo entre cada uno de ellos, los normalizamos y mediante un espacio logarítmico colocamos cada punto en el espacio al cual pertenecen, de esta forma obtenemos un histograma descriptivo de nuestra silueta, la cual podremos comparar con otro histograma de otra silueta con tal de calcular la semejanza entre ambas siluetas y de esta forma decidir qué tan parecidas son ambas imágenes.

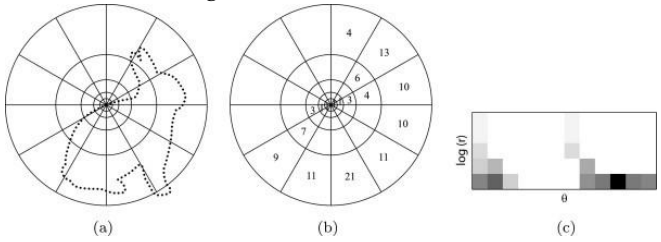


Ilustración 3: Construcción de histograma de ShapeContext

4.2.3 ORB (Oriented FAST and rotated BRIEF) [10]

Desarrollado en OpenCV por Ethan Rublee en el año 2011, es un detector de características local rápido y robusto, el cual puede ser usado en la visión por computación para el reconocimiento de objetos o la reconstrucción 3D. Una alternativa eficiente y viable a SIFT y SURF, aunque con una magnitud de velocidad casi el doble que estos. Para su ejecución utiliza los tan conocidos detector de puntos clave FAST y los descriptores BRIEF, ambos obtienen una buena ejecución en muy poco tiempo.

El detector de puntos clave FAST, consiste en escoger un pixel y comparar la iluminación de los 16 pixeles de alrededor formando un círculo alrededor del pixel y si encuentra al menos 8 pixeles más oscuros o brillante, entonces se escogerá como un punto clave. Como FAST no tiene componentes de multi escala, ORB construye una pirámide de resoluciones de la imagen y hace que FAST detecte puntos en cada una de estas capas, con tal de evitar problemas de escala. Fast tampoco ofrece componentes de orientación, por lo que ORB detecta la orientación de cada punto clave observando los niveles de intensidad alrededor del punto, de esta forma ORB obtiene los puntos clave.

Los descriptores Brief, consisten en coger todos los puntos clave FAST y convertirlos en un vector de características binario, que juntos, puedan representar un objeto. Brief comienza su proceso alisando la imagen con un kernel gaussiano con de evitar que el descriptor sea sensible al ruido de alta frecuencia. Entonces brief escoge un par de pixeles aleatorios alrededor de un punto clave. El primer pixel escogido de forma aleatoria se dibuja a partir de una distribución gaussiana centrada alrededor del punto clave con una desviación o propagación de la sigma. El segundo pixel aleatorio se dibuja a partir de una distribución gaussiana centrada alrededor del primer pixel con una desviación estándar o propagación de la sigma partida por dos. Si ahora el primer pixel es más brillante que el segundo se le asigna un valor de 1, si no un 0. Este proceso se repetirá por tantos bits tenga el vector, hasta que obtengamos la cadena binaria.

Una vez teniendo los descriptores de ambas imágenes, podemos usarlos para compararlos entre ellos y que nos dé el número de puntos similares que ha encontrado, lo cual podemos usar para determinar qué tan parecida es una imagen a otra.

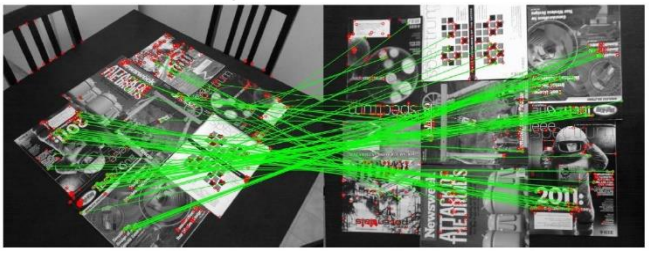


Ilustración 4: Comparación de descriptores de ORB

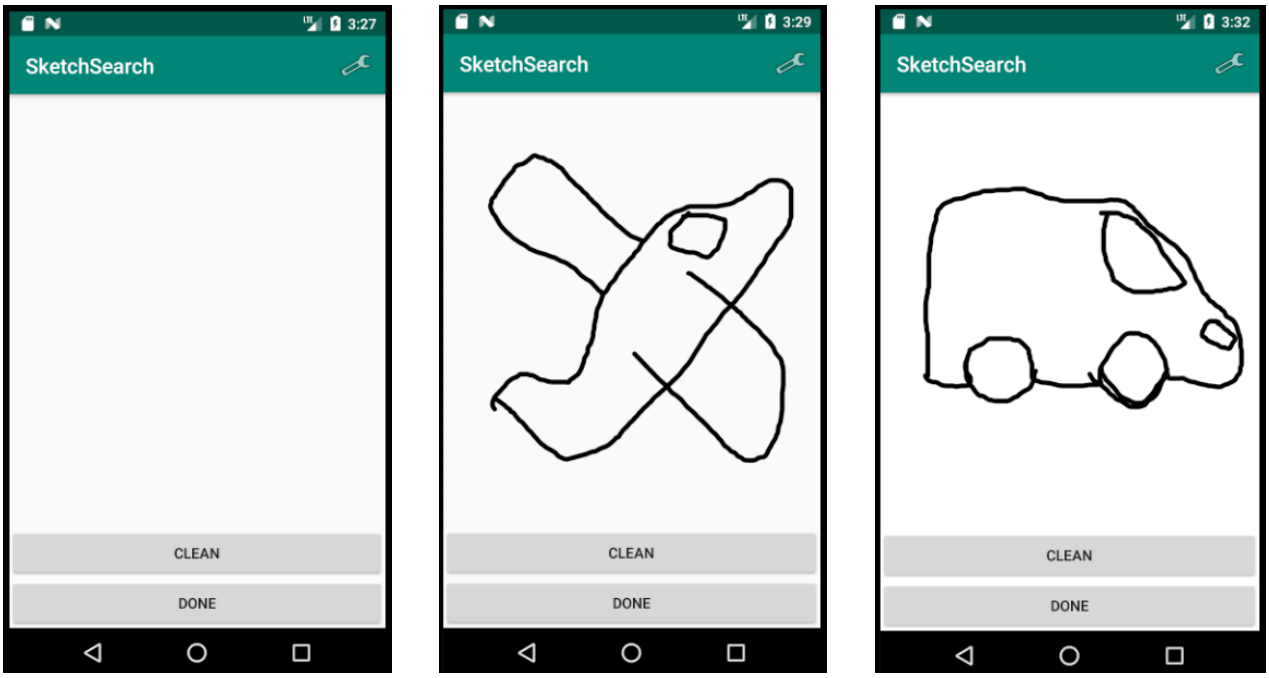


Ilustración 6: Pantalla de Dibujo

4.2.4 Hausdorff distance [11]

Introducida por Hausdorff en su libro Grundzüge der Mengenlehre [12], publicado el 1914, mide la distancia que hay entre dos subconjuntos de métricas en el espacio.

Su cálculo es simple, mediante dos imágenes A y B, buscaremos sus contornos y escogeremos una serie de puntos en cada una. Ahora por cada punto de la imagen A se seguirá el mismo proceso.

1. Cogemos punto
2. Calculamos distancia Euclidiana con cada punto de la imagen B
3. Guardamos la distancia más corta, ya que representa ser el punto más cercano del punto de A.

Una vez tenemos todas las distancias nos quedaremos con la distancia más larga, para saber cómo de lejos se encuentran ambas imágenes. De esta forma podemos hacer la comparación entre imágenes.

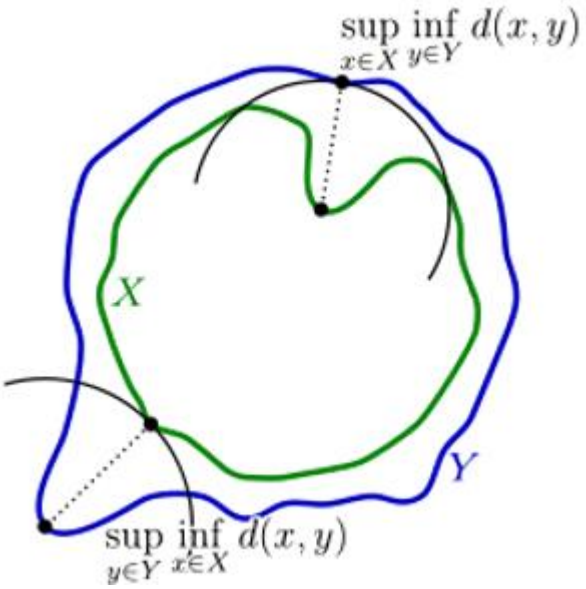


Ilustración 5: Calculo de distancia entre la figura X y Y

4.2.5 Otros Algoritmos

Durante el proceso ha habido una serie de algoritmos que han sido considerados o probados, pero por algún motivo u otro se han terminado descartando. Estos son.

- SIFT (Scale-Invariant Feature Transform)

En 2004, D.Lowe creo este nuevo algoritmo. Funciona de forma similar a ORB, también se basa en la detección de puntos clave a los cuales se les define una descripción la cual podemos usar para la comparación de imágenes. Este algoritmo dejar de ser gratuito para OpcenCV, lo cual provocó su retirada de la librería, es uno de los motivos por el cual este algoritmo no fue escogido para la aplicación, pero el más importantes es que ORB, hace el mismo trabajo, pero a una velocidad mucho mayor, lo cual lo convierte en un algoritmo mucho más eficiente y hace que SIFT se convierta en algo irrelevante.

- MSER (Maximally stable extremal regions)

Propuesta por Matas, es un algoritmo detector de manchas, hecho para detectar correspondencias entre dos imágenes con diferente Angulo de visión.

Este algoritmo fue descartado por no tener implementación en Java, lo cual suponía que se debiera programar a mano desde 0, al final por falta de tiempo no se realizó y se decidió darles más importancia a otros algoritmos que en teoría darían mejor resultado.

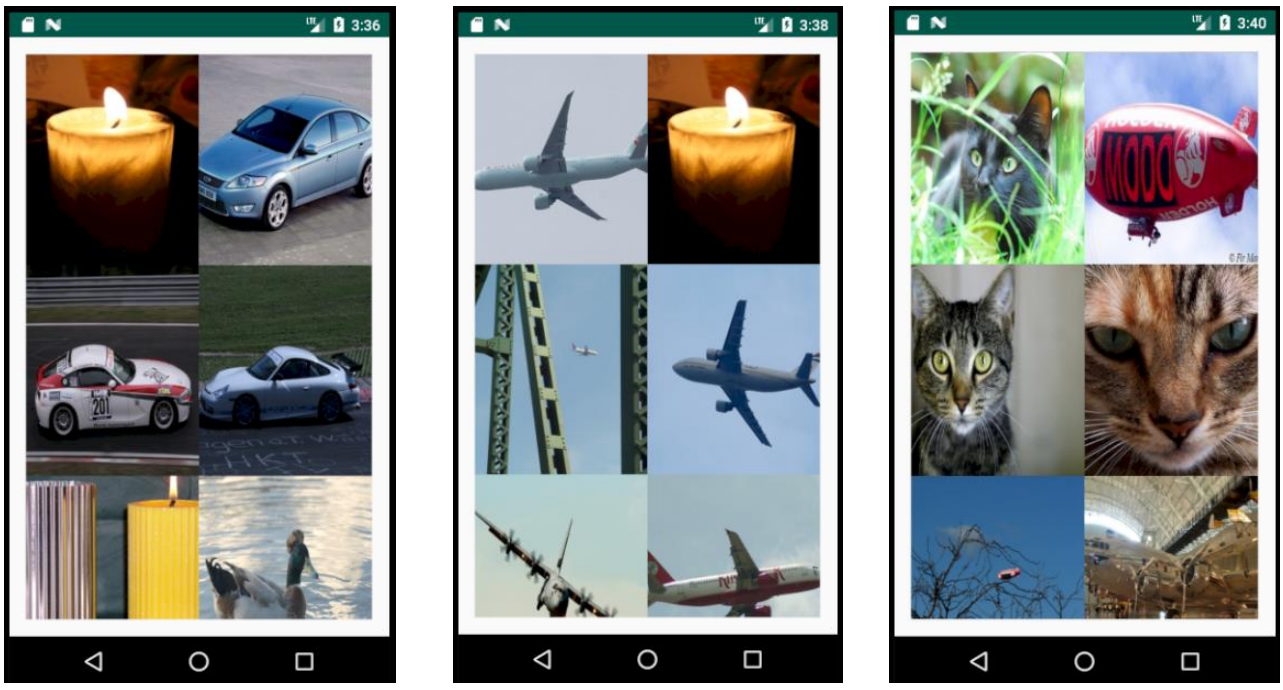


Ilustración 7: Pantalla de Galería

- Vocabulary Tree

Es una forma compacta de hacer recuperación de imagen. Formado por tres pasos y utiliza otras técnicas de visión por ordenador, tales como SIFT.

El primer paso es construir un árbol kmeans usando descriptores SIFT, cada nodo hoja de este árbol contiene una bolsa de descriptores SIFT.

El segundo paso es construir una base de datos de imágenes usando el árbol que hemos hecho en el anterior paso. Se puede ver como una cuantización de una imagen dentro de un vector espacial.

El tercer paso es listar la imagen contra las imágenes de la base de datos.

Este algoritmo fue descartado por falta de tiempo, dado a que era el algoritmo con menor prioridad y el que se esperaban peores resultados.

4.3 Interfaz del Usuario

Ahora iremos pantalla por pantalla, explicando que es lo que el usuario ve y que hace internamente cada acción que el usuario realiza.

4.3.1 Pantalla de Dibujo

La pantalla principal de la aplicación, donde se encuentra la tabla de dibujo, como se ve en la *ilustración 6,* esta tabla se encuentra definida en la clase Tacticl, donde define un Canvas al que le asigna un Bitmap.

Así el usuario cuando presiona la pantalla se detectan las coordenadas de la posición donde se ha pulsado, y entonces cuando el usuario se mueva por la pantalla para realizar su dibujo, el programa va detectando los nuevos puntos cada un pequeño tiempo y calcula sus distancias Euclidianas entre sí, de esta forma la clase puede ir dibujando líneas entre los puntos, de una forma tan constante que no parecen líneas rectas. Una vez el usuario deje de presionar la pantalla, el programa elimina las coordenadas y de esta forma permite que volvamos a empezar el proceso si se ve necesario realizar más de una línea para dibujar le dibujo completo.

También implementamos un método clear, el cual llena la pantalla de color blanco con tal de eliminar el dibujo, así podemos volver a empezar.

De esta forma en la clase Sketch, que es donde se visualizara la tabla de dibujo, podemos crear un objeto de la clase Tactil al cual le asignaremos la vista de la tabla definida en el layout de la clase, así tan solo Sketch debe llamar a la función getmBitmap, para terminar, recibiendo el bitmap de Tactil donde se encuentra el dibujo.

Sketch contiene dos botones, uno para limpiar el dibujo y el otro para finalizar con el proceso y transmitir el resultado para que sea procesado y comparado con otras imágenes.

4.3.2 Pantalla de Galería

La pantalla donde visualizaremos los resultados de nuestra búsqueda, tal y como se ve en la *ilustración 7,* esta pantalla tiene una clase principal llama Gallery, que es donde se procesaran las comparaciones y se colocaran las imágenes según su orden de semejanza.

Gallery cuenta de una función principal que es la que se encarga de procesar toda la galería, entre ello las comparaciones y el orden en el que van a salir, según las elecciones del usuario, la comparación se efectuara entre una de los tres tipos de algoritmos que hay, y si se quiere realizar por ventanas o no.

Una vez tiene los resultados de la comparación, compara resultados de cada una de las imágenes y las ordena en una lista de la clase CreateList, a la cual se le asignara un adaptador de la clase MyAdapter, este adaptador hace

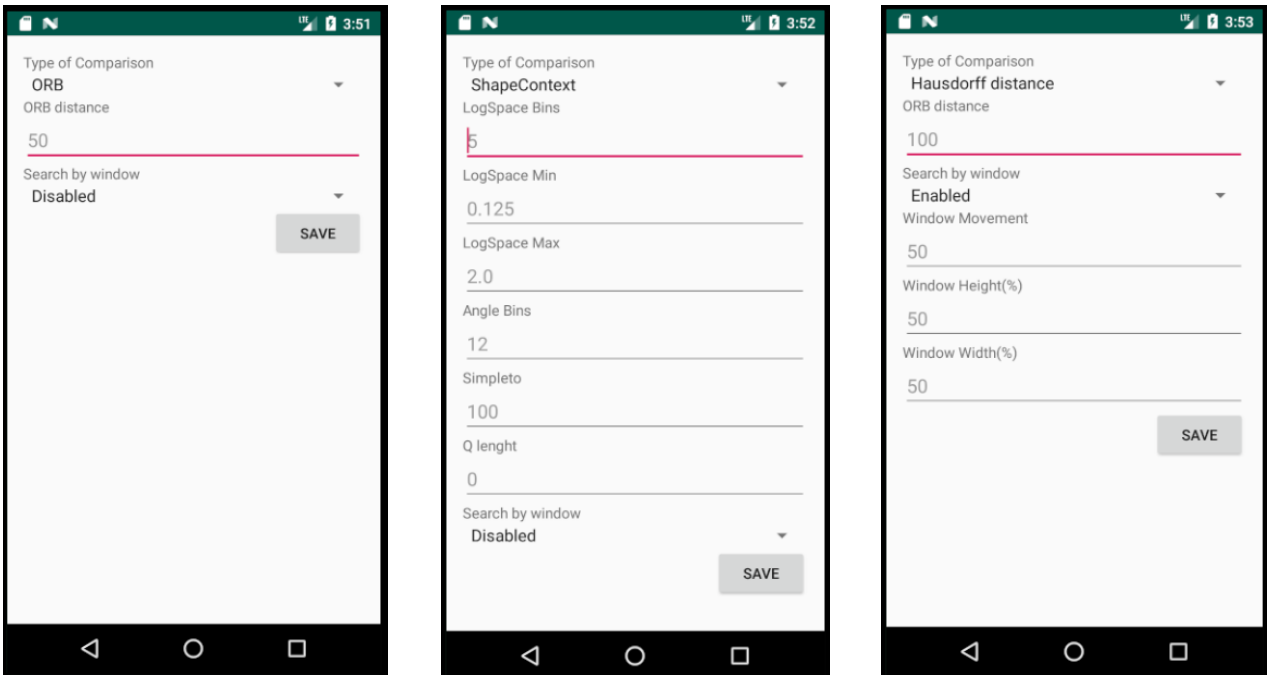


Ilustración 8: Pantalla de Configuración

que sea posible que las imágenes salgan de forma descendientes y una a una, para que quede tal y como una galería debería ser.

Mas adelante cada una de estas imágenes tendrá la posibilidad de ser presionada para poder abrirse, por lo cual Gallery envía a la actividad Image, el numero de la imagen que ha sido presionada, con tal de que Image se encarga de mostrarla por pantalla.

4.3.3 Pantalla de Configuración

En esta pantalla, la que se muestra en *ilustración 8*, la cual puede ser accedida desde la pantalla de dibujo presionando al botón que se encuentra en la Toolbar, se encuentra las opciones para que el usuario puede realizar cambio en el algoritmo de comparación.

Encontraremos un Spinners, uno que hace referencia a cuáles algoritmos de búsqueda se quiere utilizar y el otro que indica si queremos hacer una búsqueda por ventana o no.

En el primero podremos elegir si queremos que se realice una comparación ORB, Shape Context o Hausdorff distance, una vez seleccionada el algoritmo que queremos, podemos definir los parámetros abajo del algoritmo en cuestión, solo si el usuario lo ve necesario.

En el segundo spinner podemos encontrar la opción de activar o desactivar la búsqueda por ventana. Si se desea buscar por ventana, saldrán los parámetros de la ventana en cuestión, cuanto queremos que la ventana se mueva y como de grande deseamos que esta sea.

5 Resultados

Ahora para el estudio de los resultados haremos diferentes tipos de prueba con tal de mostrar la eficiencia de la aplicación. Para esto dividiremos las pruebas en dos tipos principalmente, entre pruebas de interfaz y pruebas de resultados de búsqueda.

En la primera observaremos como la gente reacciona al tener el dispositivo por primera vez en sus manos y en la segunda observaremos si los resultados de la búsqueda se corresponden al dibujo previamente dibujado.

5.1 Interfaz

Para esta fase hemos creado una plantilla de pruebas que cada usuario deberá conseguir pasará con tal de ver la eficiencia del dispositivo. Las pruebas en cuestión son las siguientes.

1. Consigue hacer una búsqueda
2. Entender el resultado de la búsqueda
3. Consigue limpiar la pantalla de dibujo
4. Cambia de algoritmo de búsqueda
5. Has una búsqueda por ventana
6. Cambia el parámetro de búsqueda.

Estas pruebas fueron realizadas por un total de 10 personas, en las que se encontrarán personas de diferentes edades, a las cuales se les informara previamente la función principal de la aplicación con tal de poder realizar el test.
El resultado será apuntado en una tabla donde cada fila representara a una persona, la cual indicaremos su edad, y marcaremos cuales pruebas fueron capaces de realizar con éxito.
De esto calcularemos el porcentaje de en ciertos de cada prueba con tal de poder extraer conclusiones luego.

5.2 Resultados de búsqueda

Estas pruebas serán más extensas que las anteriores, y se tratara de realizar diferentes búsquedas, probando cada búsqueda con cada uno de los 3 algoritmos, y apuntaremos los resultados encontrados.

Sujeto	Edad	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5	Prueba 6	Total
1	25	1	1	1	1	1	1	**6**
2	22	1	1	1	0	1	1	**5**
3	21	1	1	1	1	1	1	**6**
4	50	1	1	1	0	0	0	**3**
5	52	1	1	1	1	1	1	**6**
6	81	1	0	1	0	0	0	**2**
7	12	1	0	1	0	0	0	**2**
8	22	1	1	1	1	1	1	**6**
9	30	1	1	1	1	1	1	**6**
10	55	1	1	1	1	1	1	**6**

Tabla 1: Resultados Interfaz

Tag	airplane	car	cat	ape	door	guitar	knife	hammer	tree	violin	window	elephant
Cantidad	10	10	5	5	10	5	5	5	7	5	5	5

Tabla 2: Bas de Datos de aplicación

Airplane	ORB	ShapeContext	Hausdorff
Sin ventana	55% (2,86s)	30% (15,82s)	43% (1,72s)
Ventana	10% (10,52s)	5% (60,43s)	22% (5,43s)

Tabla 3: Búsqueda Avión

Para apuntar los resultados haremos lo siguiente. Para cada búsqueda le asignaremos un tag, este tag hará referencia al objeto que se está tratando buscar. Al mismo tiempo, llenaremos la base de datos de imágenes de la aplicación con un cierto número de imágenes, cada una clasificada con un tag también, así que ahora apuntamos el resultado de la búsqueda, apuntado la cantidad de veces que cada tag aparece y se tendrá en cuenta que tan semejante es el tag encontrado al buscado.

También hay que tener en cuenta que el orden de aparición es importante, ya que cuando más alto aparezca más semejanza deberían tener. Así que a los valores más altos les daremos una puntuación mayor. Por lo que, si la aparición de un tag suma 1 al contador, multiplicaremos el que se encuentra en primera posición por 10, la segunda por 9, el tercero por 8 y así sucesivamente hasta llegar a 1, que será donde dejaremos de contar imágenes

Entre otras cosas se apuntará el tiempo de búsqueda de cada ejecución y al final se hará una media ya que es otro importante dato a tener en cuenta.

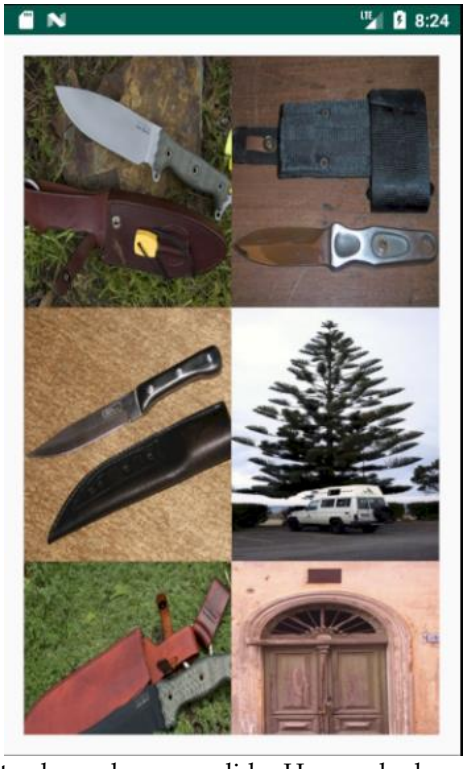
Para el cálculo de estos resultados se mantendrá los valores de los parámetros por defecto sin variación.

6 Conclusiones

En este proyecto se ha realizado una aplicación de búsqueda de imágenes a partir de bocetos, en modo general

Puerta	ORB	ShapeContext	Hausdorff
Sin ventana	63% (2,71s)	72% (15,75s)	55% (1,61s)
Ventana	17% (10,12s)	10% (60,23s)	15% (6,01s)

Tabla 4: Búsqueda de Puerta



Knife	ORB	ShapeContext	Hausdorff
Sin ventana	76% (2,63s)	64% (15,73s)	78% (1,71s)
Ventana	15% (8,15s)	11% (61,45s)	17% (5,14s)

Tabla 5: Búsqueda de Cuchillo

los objetivos planteados se han cumplido. Hemos de destacar que la comparación de bocetos con imágenes es aún un reto científico y no está del todo implementado, por ello no es de esperar que a partir de un boceto obtengamos de modo preciso las imágenes de la clase perteneciente.

6.1 Conclusiones en la interfaz

En cuestión de la interfaz, como se puede ver en *tabla 1*, la gran mayoría han podido superar las pruebas sin problemas, aunque podemos decir que aquellos que menos pruebas han superado son personas mayores, las cuales el manejo de la tecnología les supera o personas muy jóvenes, que aún no han desarrollado una conexión con los aparatos electrónicos y por ello no entienden la aplicación. Por el resto, se puede ver que han sabido desarrollarse sin problemas, y han conseguido pasar todas las pruebas. Considero que el modelo de la interfaz es correcto y no tiene fallas, al menos potenciales.

6.2 Conclusiones en la comparación

Ahora en el apartado de búsqueda de imágenes podemos ver que los resultados son un tanto pobres. Esto es algo que ya se esperaba desde el principio, ya que conseguir un algoritmo que pueda comparar algo tan abstracto como lo que puede ser un dibujo a una imagen real de una forma eficiente, es muy difícil. Para que el algoritmo funcione correctamente el dibujo y la imagen han de ser muy parecidas, pero cada imagen es única en su estado, lo cual provocaría que la comparación fuera exacta con una imagen y no con varias. Para ello, deberíamos permitir que el algoritmo perdonase distancias más largas, con tal de abarcar un mayor número de comparaciones, pero si perdonamos demasiado, el algoritmo terminara detectando imágenes no deseadas.

Aun así, se puede ver que el algoritmo funciona mejor cuando las imágenes son más simples, como la puerta, *tabla 4,* o el cuchillo, *tabla 5,* esto debe ser debido a que dibujarlos con más detalles lleva mucho menos trabajo que los otros dibujos como el avión, *tabla 3.*
Por otra parte, el mayor enemigo de esta aplicación es la optimización, los algoritmos son muy complejos y esto provoca que si el numero de la base de datos es muy grande, este no termine nunca. Esto provoca que la base de datos se vea limitada, y por lo tanto haya menos casos para que la comparación pueda ser más precisa. Si pudiéramos meter mucha más información, es muy probable que el algoritmo funcione mejor.

6.3 Posibles mejoras

- **Mejora de Optimización,** posiblemente el mayor enemigo del proyecto. La aplicación necesita algoritmos que calculen la semejanza de forma mas rápida, dado a que los actuales son lentos lo que provoca que un mayor número de imágenes cause mucho tiempo de cálculo, y si no tenemos las imágenes necesarias la búsqueda puede no resultar ser del todo buena.
- **Mejorar Interface,** habría que cambiar la interfaz de la aplicación con tal de poder adaptarse a todas las edades, cuanta mas gente pueda usar la aplicación con mas facilidad mejor para todos.
- **Implementación de nuevos algoritmos,** implementar aquellos algoritmos que han quedado descartados por falta de tiempo y puede que alguno más. Con tal de buscar comparaciones más optimas.
- **Implementar nuevas funciones,** se podría aplicar otras funciones que mejoren el uso a los usuarios, tal como historiales de búsqueda o búsqueda por color.

6.4 Conclusiones finales

Visto de otra forma, si la aplicación se basara más en el uso de encontrar imágenes en específico más que un subconjunto de estas, este funciona bastante bien, dado a que si el usuario recuerda la imagen que quiere buscar y la dibuja con la mayor exactitud posible, es muy probable que la encuentre, esta es una función útil que un buscador por palabras no sería capaz de hacer, ya que describir imágenes precisamente con palabras es mucho más complejo que no tan solo dibujarlas.

Para finalizar he de decir, que consideró que la aplicación funciona lo mejor que puede en la dedicación supuesta para un TFG, puede que con más tiempo pudiese haber solucionado problemas como la optimización, pero, aun así, considero que el trabajo está bastante completo y funciona de forma correcta.

AGRADECIMIENTOS

Este trabajo se ha realizado bajo la supervisión de Josep Lladós Canet, a quien me gustaría dar mi más profundo agradecimiento, por hacer posible la realización de este proyecto.

Además de agradecer su paciencia y dedicación que tubo para poder ayudarme en la realización de este proyecto.

Agradecer a mi familia, por su soporte tanto emocional como físico durante todo el proyecto.

Especialmente a mi hermano, por ayudarme en los cálculos de los diferentes algoritmos e instruirme en conocimientos de los cuales anteriormente desconocía.

A mi hermana, por quedarse a mi lado durante largas noches con tal de ofrecerme el soporte para continuar trabajando.

También a mis amigos, por sus múltiples consejos a la hora de realizar el proyecto y por servir de apoyo mutuo durante todos esos días donde el trabajo era abrumador.

BIBLIOGRAFÍA

[1] «Unofficial Google Image Search by Drawing,». Available: https://lifehacker.com/search-google-images-by-drawing-what-youre-looking-for-5881047.

[2] 500px, «Splash,». Available: https://labs.500px.com/.

[3] G. C. Lab, «Quick Draw!,». Available: https://quickdraw.withgoogle.com/#.

[4] Google, «Google Images,». Available: https://images.google.com/.

[5] I. Inc., «TinEye,». Available: https://tineye.com/.

[6] «Yandex,». Available: https://www.yandex.com/images/.

[7] Microsfot, «Bing Image Match,». Available: https://www.bing.com/images/discover?FORM=ILPMFT.

[8] J. F. Canny, «Canny Edge detector,» . Available: https://en.wikipedia.org/wiki/Canny_edge_detector.

[9] S. B. &. J. Malik, «ShapeContext,». Available: https://medium.com/machine-learning-world/shape-context-descriptor-and-fast-characters-recognition-c031eac726f9.

[10] E. Rublee, «Oriented FAST and rotated BRIEF,». Available: https://medium.com/software-incubator/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf.

[11] Hausdorff, «Hausdorff distance,». Available: http://cgm.cs.mcgill.ca/~godfried/teaching/cg-projects/98/normand/main.html.

[12] F. Hausdorff, Grundzüge der Mengenlehre, https://archive.org/details/grundzgedermen00hausuoft/page/n7.

APENDICE

1. Resultados Extras

Car	ORB	ShapeContext	Hausdorff
Sin ventana	51% (2,5s)	67% (16,01s)	56% (1,67s)
Ventana	12% (11,42s)	7% (62,75s)	17% (5,26s)

Tabla 6: Búsqueda de Coche

Tree	ORB	ShapeContext	Hausdorff
Sin ventana	41% (2,55s)	32% (16,05s)	37% (1,87s)
Ventana	7% (8,73s)	3% (59,32s)	5% (6,11s)

Tabla 7: Búsqueda de Árbol

Hammer	ORB	ShapeContext	Hausdorff
Sin ventana	67% (2,64s)	52% (15,25s)	64% (1,77s)
Ventana	11% (8,17s)	9% (60,73s)	15% (5,63s)

Tabla 8: Búsqueda de Martillo