UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de un sistema cognitivo de visión para la navegación robótica

Projecte Final de Carrera

Ingeniería Técnica Superior de Informática de Sistemas

Autor: Rafael Aracil López

Director: Fernando López García

28/07/2012

Resumen

El proyecto no trata de la obtención de puntos característicos, trata de la navegación robótica o la navegación de un robot. Para ello se utiliza la obtención de puntos característicos en los frames para reconocer la parte en la que se encuentra un robot en un circuito dado, en este caso dentro de unas instalaciones universitarias. Una vez localizados dichos puntos, según con el método concreto que se utilice, lo siguiente que realizaríamos sería a través de los puntos obtenidos, para cada punto, buscar el vecino mas próximo de cada punto, esto quiere decir que a través de la base de datos realizada de diferentes frames/imágenes, compararíamos descriptores para así poder decir si son puntos correspondientes a unos u otros, y poder así clasificarlos según el criterio decidido.

Durante el proyecto, se estudiarán e implementarán dos maneras y métodos diferentes de realizar lo descrito anteriormente.

La primera manera desarrollada será a través del detector de características SIFT (Scale-Invariant Feature Transform), que se centra en buscar puntos característicos que cumplen criterios espacio-escalares. Los descriptores se calculan a través de la orientación de los gradientes de cada punto.

La segunda manera es SURF, uno de los algoritmos más utilizado para la extracción de puntos de interés en el reconocimiento de imágenes. La extracción de los puntos la realiza detectando en primer lugar los posibles puntos de interés y su localización dentro de la imagen.

El lenguaje y entorno utilizado para nuestro trabajo será MATLAB.

Palabras clave: navegación robótica, puntos caracteristicos, frames, videos, descriptores, clasificar, SIFT(Scale-Invariant Feature Transform), SURF(Speed Up Robust Features), Matlab, algoritmo, vecino mas próximo.

Tabla de contenidos

1. Introducción

El proyecto no trata de la obtención de puntos característicos, trata de la navegación robótica o la navegación de un robot. Para ello se utiliza la obtención de puntos característicos en los frames para reconocer la parte en la que se encuentra un robot en un circuito dado, en este caso dentro de unas instalaciones universitarias. Cuando se toman fotografías de un mismo escenario desde cámaras en distinta posición, las imágenes obtenidas obviamente tendrán puntos comunes, pero diferirán en localización, escala, orientación, etc.

Por ese motivo es interesante desarrollar un software capaz de encontrar y asociar dichos puntos independientemente de cómo estén presentados.

El primer paso, como se ha comentado anteriormente, es buscar los puntos de la imagen que aporten información realmente importante. En general, estos puntos denominados de interés o característicos, tienen en común una serie de propiedades. Algunas de ellas son, por ejemplo, que se pueden encontrar de una manera sencilla y clara matemáticamente. Es decir, al someter la imagen a una operación matemática determinada, dichos puntos destacan significativamente. Además, tienen una posición muy bien definida y el conjunto de píxeles vecinos que hay alrededor del punto característico aporta una gran cantidad de información local relevante.

El siguiente paso, una vez calculado dónde están las zonas de interés, es describir la zona en cuestión. Los puntos concretos que nos proporcionan cualquiera de los métodos anteriores nos aportan información de localización, sin embargo, no es suficiente para una posterior búsqueda de correspondencias entre imágenes. Por ese motivo, una vez los métodos anteriores nos dicen dónde se puede extraer información, hay que describir el vecindario del punto. Dicha descripción de los alrededores del punto se llevará a cabo en un radio determinado. Cuanto mayor sea este radio, mayor coste computacional conllevará su cálculo, pero se describirá con mayor amplitud cada región de la imagen. Por el contrario, no interesan descriptores demasiado grandes, ya que en ese caso se perdería el concepto de 'localidad' en los descriptores. Por tanto, hay que llegar a un término medio que proporcione un algoritmo eficiente y a su vez que describa suficientemente cada zona importante.

La descripción de los vecindarios nos permitirá asociar los puntos clave de una imagen de entrenamiento con otra de test, y por tanto seremos capaces de identificar puntos correspondientes entre ellas. Por ese motivo es tan importante que todos estos descriptores locales sean insensibles a cambios de escala, rotaciones, traslaciones, etc. Mientras en una imagen A (de entrenamiento) aparece un objeto en una posición determinada, en una imagen B (de test) puede aparecer el mismo objeto rotado o escalado de diferente forma. Sin embargo, el algoritmo debe ser invariante a esos cambios, gracias a las propiedades singulares de los puntos característicos y sus correspondientes descriptores. Históricamente, han existido multitud de técnicas que nos permiten buscar puntos de interés.

Hay teorías más complejas que trabajaban con diferentes escalas de una imagen para buscar puntos característicos (scale-space theory , 90's), a partir de la parametrización del tamaño del filtro con la que son filtradas. Un ejemplo es el método SIFT (Scale-Invariant Feature Transform) publicado y patentado por David Lowe.

Este método está basado en la teoría de scale-space comentada anteriormente. En el SIFT se buscan puntos que son muy claros o muy oscuros en comparación a su

vecindario. Por tanto, se trabajará con la imagen en escala de grises. Una vez sean detectados, a diferencia del método anterior, se trabajarán con los histogramas de los gradientes locales alrededor del punto. Después de un proceso complejo, podremos construir los descriptores para poder calcular correspondencias entre diferentes fotografías.

Diferentes implementaciones de SIFT:

- PCA-SIFT:

- VLFEAT
  Librería de código abierto de la visión en C (con una interfaz de MEX a MATLAB), incluyendo una implementación de SIFT

- LIP-VIREO
  Un conjunto de herramientas para la extracción de características punto clave (los binarios para Windows, Linux y SunOS), incluyendo una implementación de SIFT

- SIFT en C #
  El algoritmo SIFT en C # utilizando Emgu CV y una versión modificada del algoritmo paralelo.

- SIXT Escala algo invariante Transformar

Nosotros vamos a utilizar VLFEAT, debido a la potencia de MATLAB.

El extractor de características locales SURF, como otros descriptores de su clase, supone una técnica de extracción de puntos de interés de una imagen. Mediante el SURF, para los puntos seleccionados de la imagen se obtiene un conjunto de descriptores muy distintivos e invariantes ante cambios en escala, iluminación y perspectiva. Por ello, el emparejamiento de estos puntos entre dos imágenes distintas es altamente robusto. El coste computacional que presenta es, además, inferior en comparación con otros extractores similares como SIFT

SURF es otro de los algoritmos más utilizado para la extracción de puntos de interés en el reconocimiento de imágenes. La extracción de los puntos la realiza detectando en primer lugar los posibles puntos de interés y su localización dentro de la imagen.

Es mucho más rápido que el método SIFT, ya que los keypoints contienen muchos menos descriptores debido a que la mayor cantidad de los descriptores son 0. Este descriptor se puede considerar una mejora debido a que la modificaciones que supondría en el código no serían excesivas, ya que el descriptor SURF utiliza la gran mayoría de las funciones que utiliza el descriptor SIFT.

Diferentes implementaciones de SURF:

- METODO ORIGINAL

Es la versión original de SURF y la raíz de las demás implementaciones posteriores.

- OPENSURF
Al igual que el código original este es utilizado para detectar puntos de interés y descriptores entre imágenes. Creado por Chris Evans y está desarrollado en C++ y C#.

- OPENCV SURF
Son librerías dentro de OpenCV que permiten implementar el código en C++.

- SURFMEX
Interfaz MATLAB de SURF para Windows, se compila y llama a OpenCV para ejecutar.

- OPENSURF MATLAB
Es la aplicación SURF creada por Chris Evans en MATLAB

- JOPENSURF
Es una implementación de SURF realizada en Java.

Ambas propuestas (SIFT y SURF) han sido seleccionadas en su implementación MATLAB, ya que la potencia de este programa permite que se puedan obtener más descriptores que en otros por ejemplo, o que tenga un código más accesible que otros. También se eligen porque son las mas fieles al código original.

2. SIFT (Scale Invariant Feature Transform)

HISTORIA

Se ha demostrado que la idea de utilizar características locales es el método mas eficaz de reconocimiento visual, localización de robots, etc.

El primero que lo usó fue Christoph von der Malsburg usando filtros de Gabor orientados a escalas diferentes en el mismo grafico.

Mas adelante fue David Lowe el que mezcló estas características de la escala, con su escala SIFT, y ha resultado ser muy potente en el ámbito comentado anteriormente.
Entonces hablando del algoritmo SIFT de Lowe, se pueden detectar objetos incluso con oclusión parcial , ya que gracias a la escala uniforme , la orientación y a invariantes de la distorsiona fin y cambios de iluminación.

DESCRIPCIÓN

El término SIFT proviene de *Scale-Invariant Feature Transform*. Es decir, es una transformación de la información que proporciona una imagen en coordenadas invariantes a la escala en el ámbito local. A partir de las características locales, se busca conseguir invariancia a la escala, orientación, parcialmente a cambios de iluminación, etc. También se puede utilizar para buscar correspondencias entre diferentes puntos de vista de una misma escena. Estas características locales se almacenan en los denominados descriptores.

ALGORITMO

El algoritmo tendría cuatro características que tendría que encontrar y que definiremos a continuación.

Serían las siguientes:

1. **Detección de máximos y mínimos espacio-escala:** El primer paso es la búsqueda de puntos en la imagen que puedan ser keypoints. Se realiza usando diferencias de funciones gaussianas para hallar puntos interesantes que sean invariantes a la escala y a la orientación.

2. **Localización de los keypoints:** De los puntos obtenidos en el apartado anterior se determinan la localización y la escala de los mismos, de los cuales se seleccionan los keypoints basándose en la medida de la estabilidad de los mismos.

3. **Asignación de la orientación:** A cada localización del keypoint se le asigna una o más orientaciones, basado en las orientaciones de los gradientes locales de la imagen.

4. **Descriptores de los keypoints:** Los gradientes locales se miden y se transforman en una representación que permite importantes niveles de la distorsión de la forma local y el cambio en la iluminación.

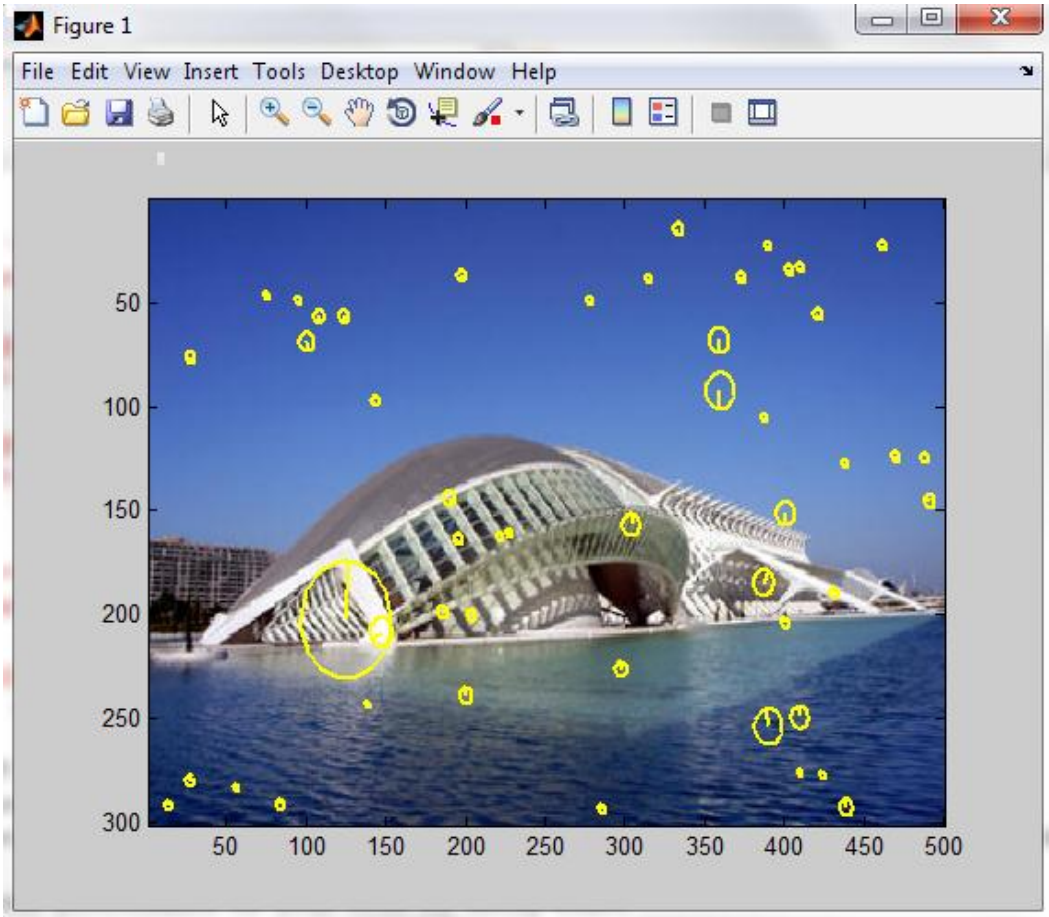

Figura 1 – Frame sacado con MATLAB con los Keypoints

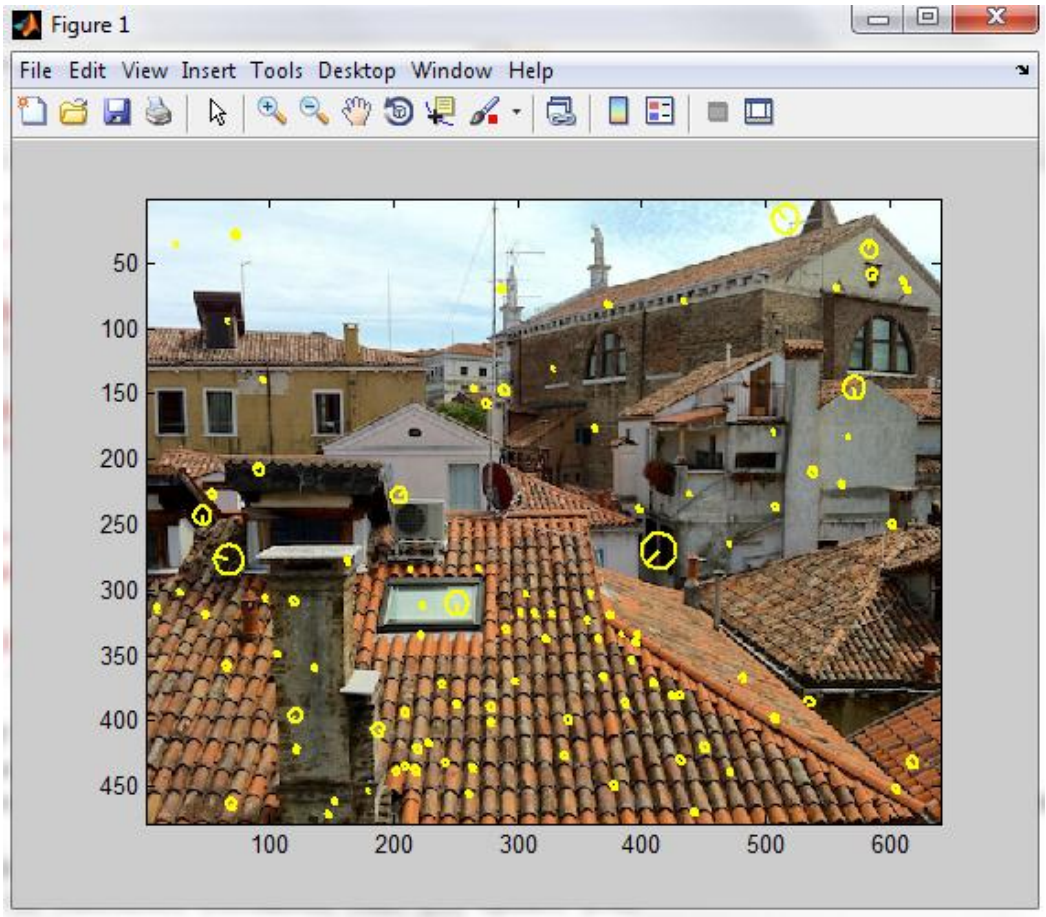

Figura 2 – **Frame sacado con MATLAB con los Keypoints**

En el siguiente dibujo podemos apreciar el esquema que tendría el algoritmo que vamos a definir a continuación.

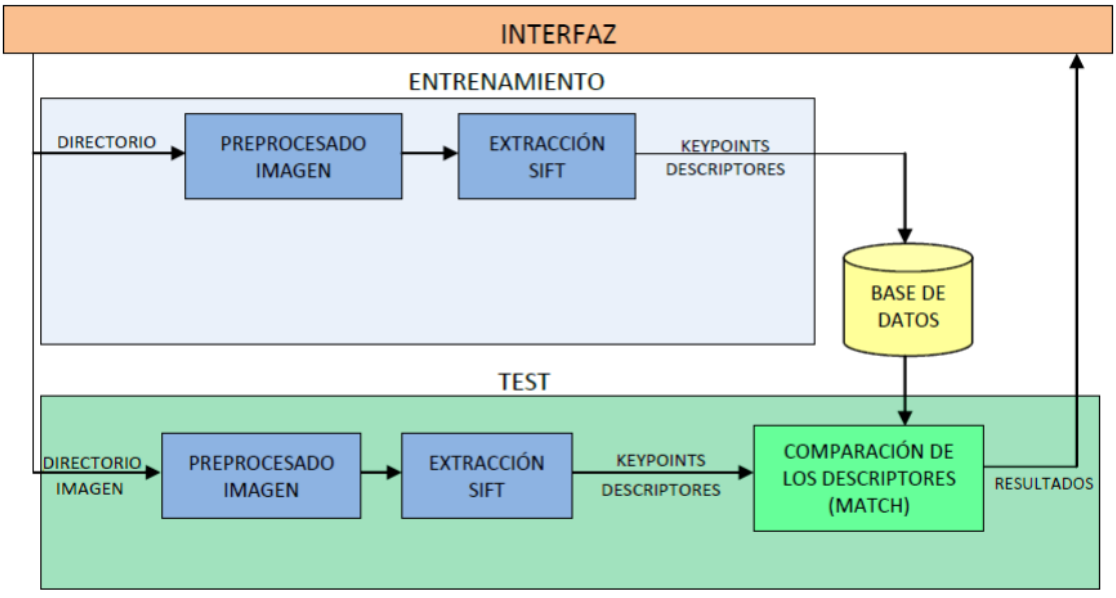

Figura 3 – **Esquema del proceso para realizar nuestro proyecto de clasificación SIFT**

Detección de máximos y mínimos espacio-escala

Esta es la etapa donde los puntos de interés, que se llaman puntos clave (keypoint) en el marco de SIFT, se detectan. Para ello, la imagen se procesa con filtros gaussianos a diferentes escalas, y luego se calcula la diferencia de los sucesivos puntos Gaussianos que hemos encontrado en la imagen. Los máximos y mínimos de esta función proporcionan las características más estables, con lo que esto serán nuestros keypoints. En concreto, un DOG de la imagen $D(x, y, \sigma)$ viene dada por $D(x, y, \sigma) = L(x, y, k_i\sigma) - L(x, y, k_j\sigma)$, donde $L(x, y, k\sigma)$ es la convolución de la imagen original I (x, y) con el desenfoque gaussiano $G(x, y, k\sigma)$ a escala $k\sigma$, Es decir,

$$L(x, y, k\sigma) = G(x, y, k\sigma) * I(x, y)$$

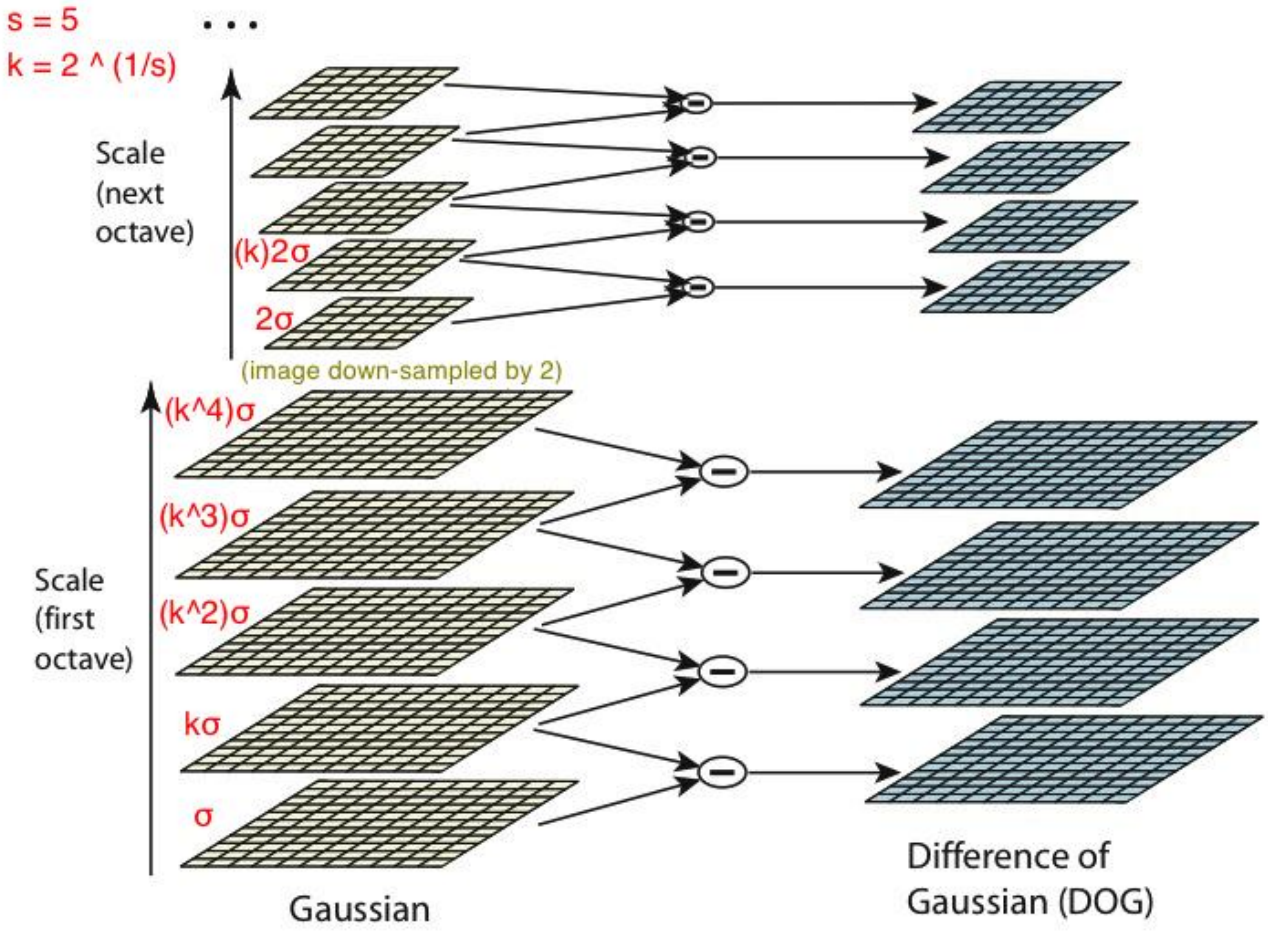

Figura 4 – Filtros Gaussianos

Localización de los keypoints

El scale-space de detección da como resultado demasiados candidatos Keypoint, algunos de los cuales son inestables. El siguiente paso en el algoritmo es realizar un ajuste detallado de los datos más cercanos para la localización exacta, la escala y proporción de curvaturas principales. Esta información permite poder rechazar puntos que tienen bajo contraste (y por tanto son sensibles al ruido) o están mal localizados.

Lo primero que haremos será, para cada keypoint que obtenemos, haremos una interpolación de los datos para determinar con precisión la posición de cada uno, con lo que mejoramos la estabilidad de nuestro método de obtención. Todo esto se realiza mediante la expresión de Taylor de la ecuación de diferencias gaussianas $D(x, y, \sigma)$.Esta nueva expresión vendría dada como:

$$D(p) = D + \frac{\partial D^T}{\partial p} p + \frac{1}{2} p^T \frac{\partial^2 D}{\partial p^2} p$$

Sabiendo que p es el keypoint obtenido del apartado anterior, derivamos la expresión y nos queda de la siguiente manera:

$$\hat{p} = -\left(\frac{\partial^2 D^{-1}}{\partial p^2} \cdot \frac{\partial D}{\partial p} \right) = -\begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial x \partial y} \\ \frac{\partial^2 D}{\partial y \partial x} & \frac{\partial^2 D}{\partial y^2} \end{bmatrix}^{-1} \cdot \begin{bmatrix} \frac{\partial D}{\partial x} \\ \frac{\partial D}{\partial y} \end{bmatrix}$$

A partir de la expresión anterior, podemos sacar una expresión que nos sirve para eliminar los puntos que tienen muy bajo contraste.

Sustituyendo la ecuación anterior podemos sacar otra que nos sirve para calcular el contraste del punto para poder asi eliminar lo que no nos sirven. La ecuación sería la siguiente:

$$D(\hat{p}) = D + \frac{1}{2} \frac{\partial D^T}{\partial p} \hat{p}$$

A partir de esta ecuación, tenemos que establecer un umbral mínimo al que deben de llegar los keypoints para no ser rechazados. Entonces todos aquellos que no cumplan la condición | D(pˆ) |> 0.03 serán eliminados.

Ahora también se deben descartar aquellos puntos que posean una pobre localización a lo largo de un borde, ya que la función diferencia de gaussianas posee una alta respuesta a lo largo de los bordes.

Un pico pobremente definido en la función diferencia de gaussianas indica que habrá una larga curvatura principal en la dirección del borde, pero pequeña en la dirección perpendicular del mismo. La curvatura principal se puede procesar a partir de una matriz hessiana de 2 × 2, evaluada en el keypoint.

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Las derivadas necesarias para obtener la matriz H se han calculado tomando diferencias con los vecinos del punto de muestreo. Los autovalores de la matriz H son proporcionales a las curvaturas principales de D. Se toma α como el mayor de los auto valores y β como el menor.

Entonces, se puede obtener la suma de los autovalores a partir de la de H y su producto del determinante:

$$\mathbf{Tr(H) = Dx,x + Dy,y = \alpha + \beta}$$

$$\mathbf{Det(H) = Dx,xDy,y - (Dx,y)^2 = \alpha\beta}$$

Si las curvaturas tienen signos diferentes, entonces dicho punto se debe descartar, ya que no está representando a un máximo o a un mínimo. Entonces, si se toma r como la relación que existe entre los dos autovalores
(α = rβ), se obtiene:

$$\mathbf{T\ r(H)^2/Det(H) = (\alpha + \beta)^2/\alpha\beta = (r\beta + \beta)^2/r\beta^2 = (r + 1)^2/r}$$

Solo depende de la relación que existe entre los autovalores. El $\mathbf{(r + 1)^2/r}$ es mínimo cuando los dos auto valores son idénticos y aumenta conforme va creciendo r.

Entonces para verificar la relación entre las curvaturas solo es necesario calcular la relación existente entre el cuadrado de la traza de H y su determinante.

$$\mathbf{T\ r(H)^2/Det(H) < (r + 1)^2/r}$$

Un valor umbral bastante razonable, sería tomar r = 10. Así, de este modo, descartamos los puntos inestables por una pobre localización en un borde.

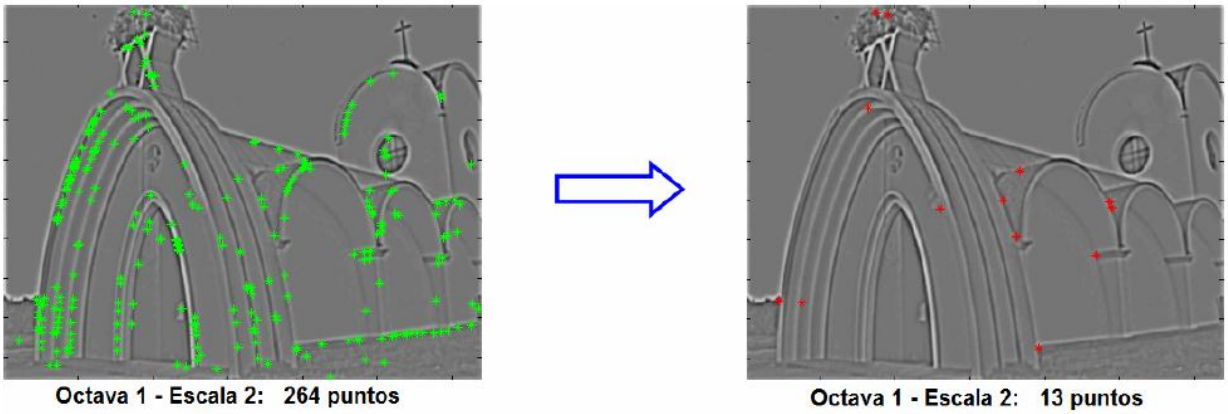

Figura 5 - **En verde podemos apreciar todos los keypoints, y en rojo los keypoints finales que no han sido descartados.**

Asignación de la orientación

La asignación de una orientación a los keypoints es muy importante, ya que si se consigue una orientación coherente basada en las propiedades locales de la imagen, el descriptor puede ser representado en relación de dicha orientación y por lo tanto ser invariante a la rotación. Este enfoque contrasta con la de otros descriptores invariantes a la orientación, que buscan propiedades de las imágenes basadas en medidas invariantes a la rotación. La desventaja de este enfoque es que limita el número de descriptores que se pueden usar y rechaza mucha información de la imagen.

Para establecer una orientación adecuada, este método se basa en el gradiente local de la imagen alrededor de los keypoints. Para ello se utilizará la imagen suavizada por la gaussiana a la mayor escala determinada por el keypoint, de modo que todos los cálculos se realizan de un modo invariable a la escala. Por cada imagen de muestreo, L(x, y), la magnitud del gradiente, m(x, y), y la orientación, θ(x, y), se precalculan usando diferencias de píxeles:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = \tan^{-1}\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right)$$

Un histograma de orientación se forma a partir de las orientaciones de los gradientes de los puntos de muestreo que se encuentran dentro de la región que rodea al keypoint. El histograma de orientación posee 36 divisiones que abarcan los 360 o del rango de orientaciones. Cada muestra añadida al histograma es pesada por su magnitud del gradiente y por una máscara gaussiana circular con un valor de σ 1.5 veces el valor que posea el keypoint.

Los picos en el histograma de orientaciones corresponden a las direcciones dominantes de los gradientes locales. Se detecta el mayor pico del histograma, y entonces, si existen otros máximos superiores al 80 % del pico principal, ´estos se utilizarán para crear otros keypoints con nuevas orientaciones.

Por lo tanto, existirán varios keypoints con la misma localización, pero con diferentes orientaciones. Solamente el 15 % de los keypoints disponen de orientaciones múltiples, pero estos puntos contribuyen significativamente en la estabilidad del método. Por último, para determinar con mayor exactitud la localización del pico en el histograma, este se interpola con una parábola que es fijada por los tres puntos más cercanos a cada pico.

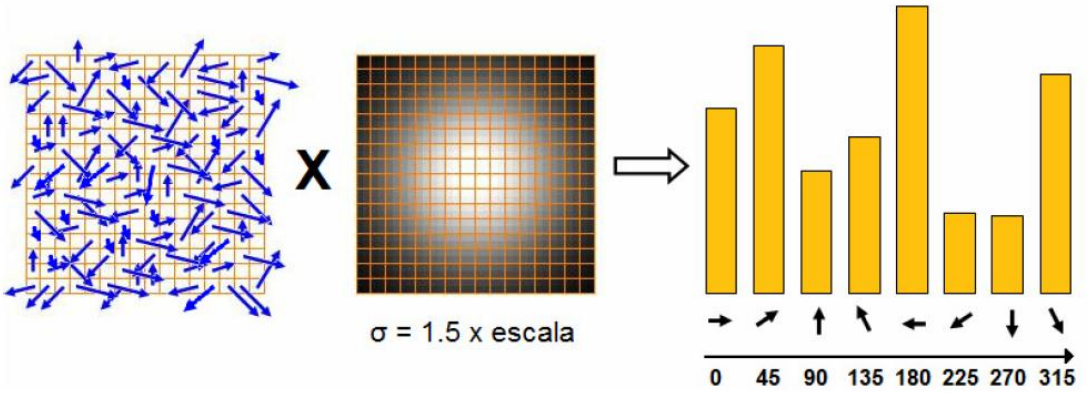

Figura 6 – **Izquierda: Región de gradientes 16x16. Centro: Ventana circular gaussiana. Derecha: Histograma final keypoint**

Descriptores de los keypoints

Una vez que ya se tienen todos los keypoints de la imagen, se tiene que segmentar la vecindad del keypoint en 4 × 4 regiones de 4 × 4 píxeles. Una vez que ya se ha dividido la vecindad del keypoint, se genera un histograma de orientación de gradiente para cada región. Para ello, se utiliza una ponderación gaussiana con un ancho σ = 4 píxeles.

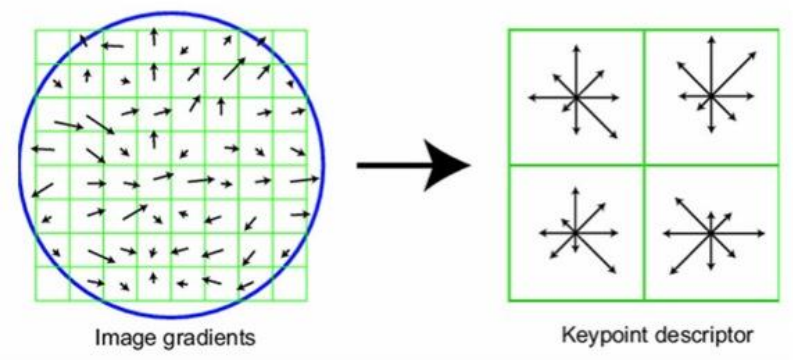

Figura 7 – Izquierda: Gradientes de la imagen. Derecha: Descriptores

Pero dicha construcción presenta un gran problema, ya que en el caso de un pequeño desplazamiento espacial, la contribución de un pixel puede pasar de una casilla a otra, lo que provoca cambios repentinos del descriptor. Este desplazamiento también puede deberse al hecho de una pequeña rotación.

Para evitar estos problemas, todos los píxeles contribuyen a todos los vecinos, tanto en magnitud como en orientación. Esta contribución se multiplica por un peso 1 – d, donde d es la distancia al centro de la casilla. Esta distancia está normalizada al tamaño de una región, es decir, la longitud de una región de 4 píxeles tiene una distancia d = 1. Además, el valor mínimo del peso 1 – d es de 0, ya que los pesos no pueden ser negativos.

Como los histogramas de orientación de cada región están divididos en 8 barras, por cada vecindad del keypoint se puede construir un histograma tridimensional de 4 × 4 × 8 valores, formando así un vector con todos estos valores.

ALGORITMO COMPARACIÓN DEL VECINO MAS PROXIMO (NN)

Una vez que ya se tienen calculados los descriptores, ya sabemos que son un conjunto de 128 elementos que nos indican la orientación alrededor de un punto de interés, así que cuando encontremos características similares en diferentes puntos, esto quiere decir que están refiriéndose a la misma zona. Nosotros hemos usado en nuestro proyecto el algoritmo del vecino más próximo.

$$dif_i = \sqrt{(a_i - b_i)^2}$$

$$dif\_total = \sum_{1}^{128} dif_i$$

Donde *difi* es la diferencia euclídea entre el elemento *ai* de un descriptor de la imagen A, y *bi* su correspondiente de un descriptor de la imagen B. La variable *dif_total* es la suma de las diferencias euclídeas que hay entre los 128 elementos de ambos descriptores.

Realizando esta serie de operaciones entre cada descriptor de la imagen A con cada uno de la imagen B, podremos decidir cuales son iguales, eligiendo siempre el que haya dado una diferencia euclídea total menor.

```
function test_targets = NN(train_patterns, train_targets, test_patterns, K)

% Classify using the Nearest neighbor algorithm
% Inputs:
%   train_patterns  - Train patterns (data)
%   train_targets   - Train targets  (classes)
%   test_patterns   - Test  patterns (classes)
%   K               - Number of nearest neighbors
%
% Outputs
%   test_targets    - Predicted targets (classes)

L  = length(train_targets);
Uc = unique(train_targets);

if (L < K),
  error('You specified more neighbors than there are points.')
end

N           = size(test_patterns, 2);
test_targets = zeros(1,N);

for i = 1:N,
   dist          = sum((train_patterns - test_patterns(:,i)*ones(1,L)).^2);
   [m, indices]    = sort(dist);
   n             = hist(train_targets(indices(1:K)), Uc);
   [m, best]       = max(n);
   test_targets(i) = Uc(best);
end
```

3. SURF (Speed Up Robust Features)

HISTORIA

SURF (speed up robust feature) es un detector es otro detector de variables locales, y fue presentado por primera vez por Herbet Bay en el 2006 y se inspira en el descriptor SIFT, pero presentando ciertas mejoras, como:

- Velocidad de cálculo considerablemente superior sin ocasionar perdida del rendimiento.
- Mayor robustez ante posibles transformaciones de la imagen.

Estas mejoras se consiguen mediante la reducción de la dimensionalidad y complejidad en el cálculo de los vectores de características de los puntos de interés obtenidos, mientras continúan siendo suficientemente característicos e igualmente repetitivos.

DESCRIPCIÓN

SURF es otro de los algoritmos más utilizado para la extracción de puntos de interés en el reconocimiento de imágenes. La extracción de los puntos la realiza detectando en primer lugar los posibles puntos de interés y su localización dentro de la imagen.

Es mucho más rápido que el método SIFT, ya que los keypoints contienen muchos menos descriptores debido a que la mayor cantidad de los descriptores son 0. Este descriptor se puede considerar una mejora debido a que la modificaciones que supondría en el código no serían excesivas, ya que el descriptor SURF utiliza la gran mayoría de las funciones que utiliza el descriptor SIFT.

ALGORITMO

A continuación voy a describir cual es el procedimiento del algoritmo SURF para detectar puntos de interés (keypoints), asignación de la orientación y por ultimo obtención del descriptor SURF.

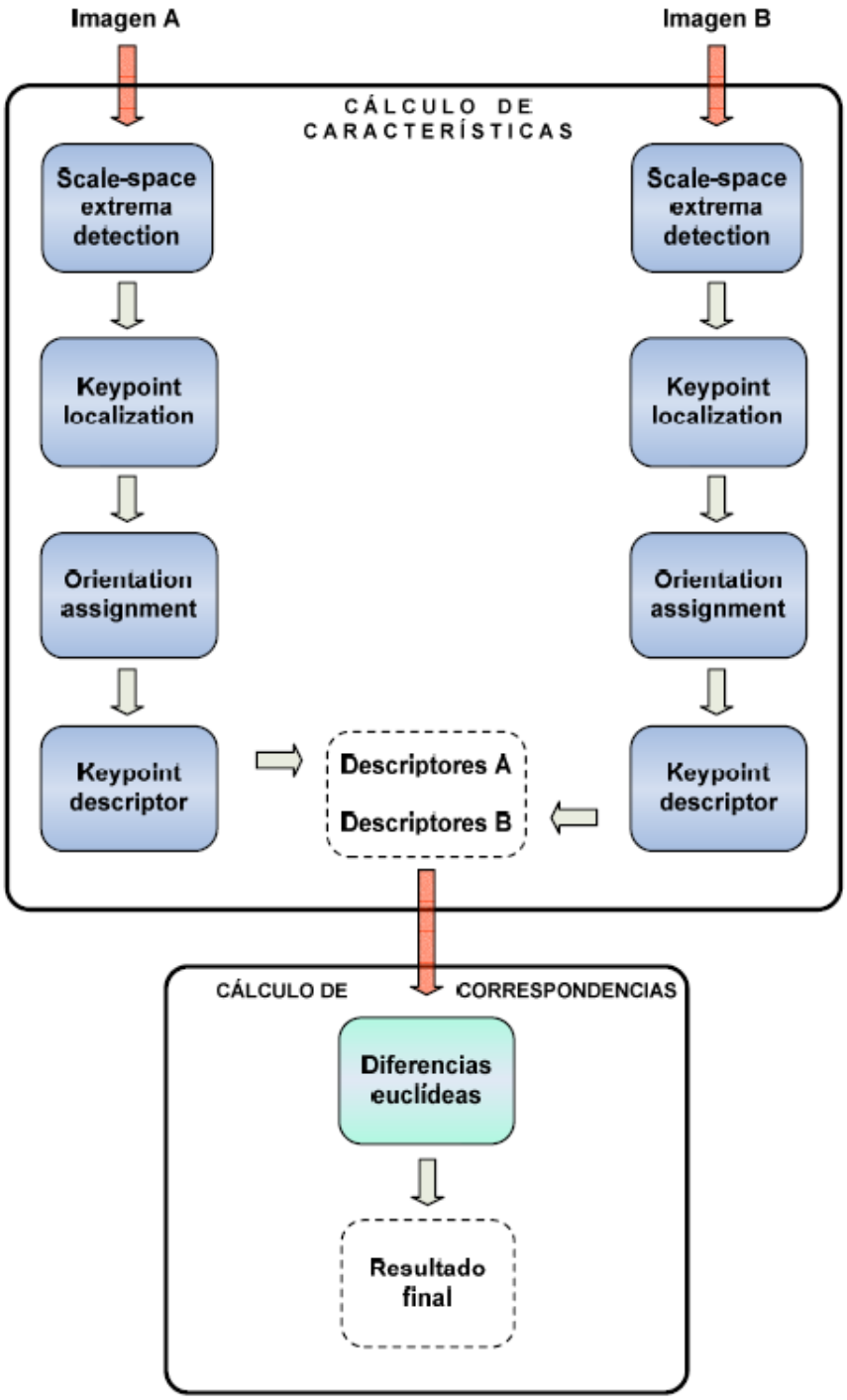

Figura 8 - **Esquema del proceso para realizar nuestro proyecto de clasificación SURF**

Detección de puntos de interés.

La primera de las etapas del descriptor SURF es idéntica a la del descriptor SIFT en cuanto a la detección de puntos de interés se refiere.

El descriptor SURF hace uso de la matriz Hessiana, más concretamente, del valor del determinante de la matriz, para la localización y la escala de los puntos. El motivo para la utilización de la matriz Hessiana es respaldado por su rendimiento en cuanto a la velocidad de cálculo y a la precisión. Lo realmente novedoso del detector incluido en el descriptor SURF respecto de otros detectores es que no utiliza diferentes medidas para el cálculo de la posición y la escala de los puntos de interés individualmente, sino que utiliza el valor del determinante de la matriz Hessiana en ambos casos. Por lo tanto dado un punto p = (x, y) de la imagen, la matriz Hessiana H (p,*o*) del punto p perteneciente a la escala *o* se define como:

$$H\left(p,\sigma\right)=\left[\begin{array}{cc} L_{xx}\left(p,\sigma\right) & L_{xy}\left(p,\sigma\right) \\ L_{xy}\left(p,\sigma\right) & L_{yy}\left(p,\sigma\right) \end{array}\right]$$

Donde $L_{xx}(\mathrm{x},\sigma)$ es la convolución de segundo orden de la Gaussiana, $\frac{\delta^2}{\delta x^2}g(\sigma)$ con la imagen I en el punto x,y similarmente para $L_{xy}(\mathrm{x},\sigma)$ y $L_{yy}(\mathrm{x},\sigma)$.

Las aproximaciones de las derivadas parciales se denotan como Dxx, Dxy y Dyy, y el determinante se calcula de la siguiente manera.

$$det\left(H_{aprox.}\right)=D_{xx}D_{yy}-\left(0{,}9D_{xy}\right)^2$$

Donde el valor de 0,9 está relacionado con la aproximación del filtro Gaussiano.

En la siguiente imagen se puede observar la representación de la derivada parcial de segundo orden de un filtro gaussiano discretizado y la aproximación de la Derivada implementada en el caso del descriptor SURF.

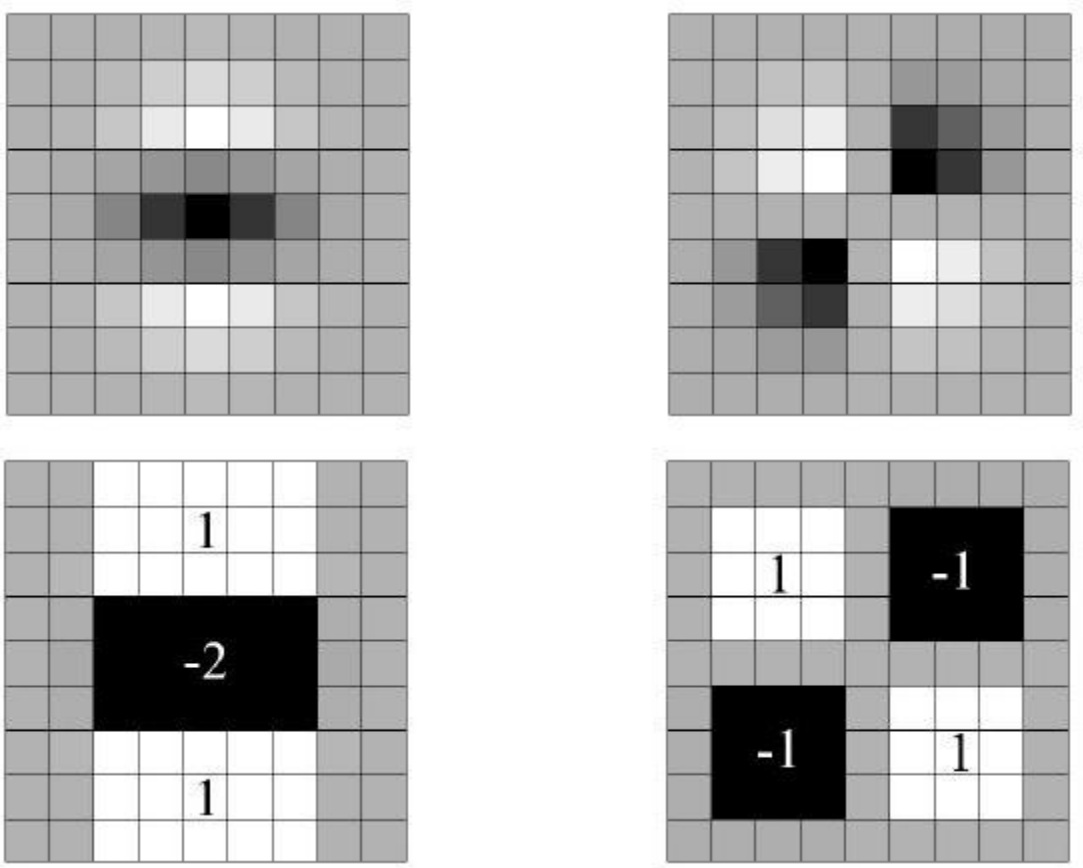

Figura 9 - En la siguiente imagen se puede observar la representación de la derivada parcial de segundo orden de un filtro gaussiano discretizado y la aproximación de la derivada implementada en el caso del descriptor SURF.

La imagen de salida obtenida tras la convolución de la imagen original con un filtro de dimensiones 9 x 9, que corresponde a la derivada parcial de segundo orden de una gaussiana con o = 1;2, es considerada como la escala inicial o también como la máxima resolución espacial (s = 1;2, correspondiente a una gaussiana con o = 1;2). Las capas sucesivas se obtienen mediante la aplicación gradual de filtros de mayores dimensiones, evitando así los efectos de aliasing en la imagen. El espacio escala para el descriptor SURF, al igual que en el caso del descriptor SIFT, está divido en octavas. Sin embargo, en el descriptor SURF, las octavas están compuestas por un número fijo de imágenes como resultado de la convolución de la misma imagen original con una serie de filtros cada vez más grande. El incremento o paso de los filtros dentro de una misma octava es el doble respecto del paso de la octava anterior, al mismo tiempo que el primero de los filtros de cada octava es el segundo de la octava predecesora.

Finalmente para calcular la localización de todos los puntos de interés en todas las escalas, se procede mediante la eliminación de los puntos que no cumplan la condición de máximo en un vecindario de 3 x 3 x 3. De esta manera, el máximo determinante de la matriz Hessiana es interpolado en la escala y posición de la imagen.

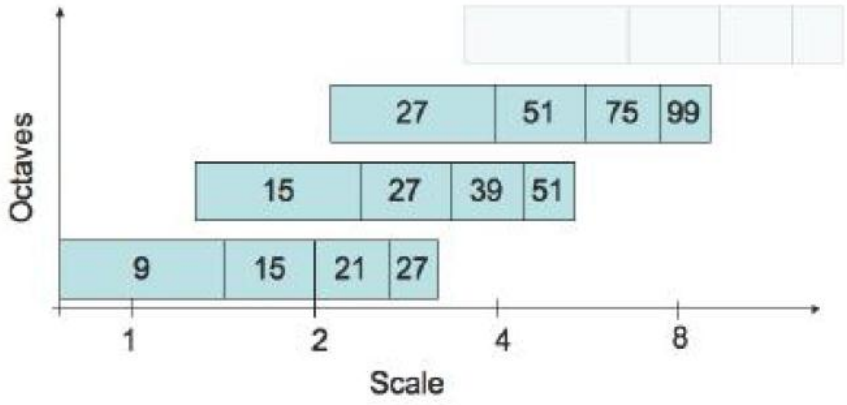

Figura 10 – Escala para el descriptor SURF

Asignación de la Orientación.

La siguiente etapa en la creación del descriptor corresponde a la asignación de la orientación de cada uno de los puntos de interés obtenidos en la etapa anterior.

Es en esta etapa donde se otorga al descriptor de cada punto la invariancia ante la rotación mediante la orientación del mismo.

El primer paso para otorgar la mencionada orientación consiste en el cálculo de la respuesta de Haar en ambas direcciones x e y mediante las funciones siguientes:

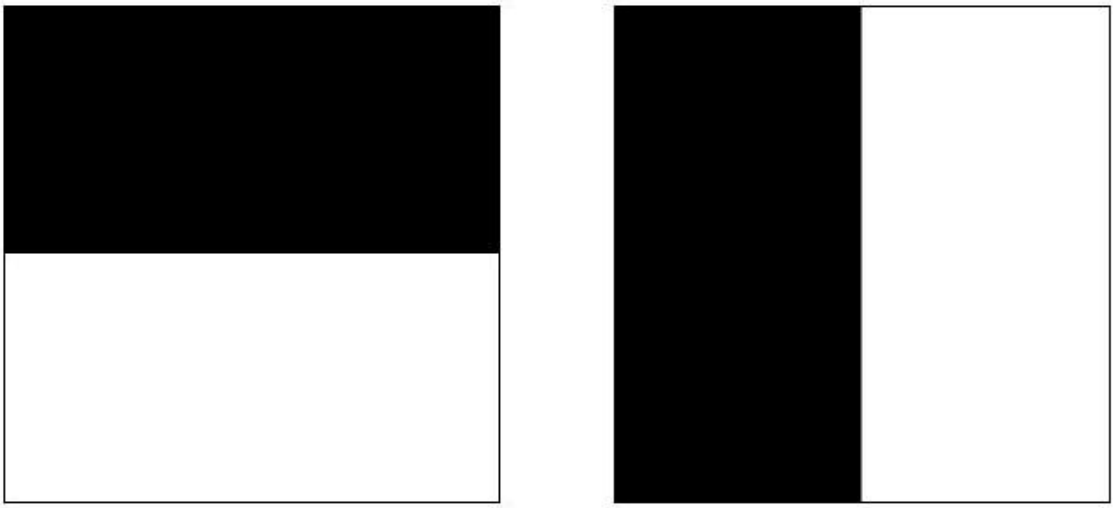

Figura 11 – Calculo de la respuesta Haar. Negro: -1. Blanco: +1.

Donde el color negro es el valor -1 y el blanco es +1.

La etapa de muestreo depende de la escala y se toma como valor s. Se toma el valor 4s, siendo s la escala en la que el punto de interés ha sido detectado, por tanto dependiente también de la escala, como referencia, donde a mayor valor de escala mayor es la dimensión de las funciones onduladas.

Tras haber realizado todos estos cálculos, se utilizan imágenes integrales nuevamente para proceder al filtrado mediante las máscaras de Haar y obtener así las respuestas en ambas direcciones. Son necesarias únicamente 6 operaciones para obtener la respuesta en la dirección x e y. Una vez que las respuestas onduladas han sido calculadas, son ponderadas por una gaussiana de valor *o* = 2,5s centrada en el punto de interés. Las respuestas son representadas como vectores en el espacio colocando la respuesta horizontal y vertical en el eje de abscisas y ordenadas respectivamente. Finalmente, se obtiene una orientación dominante por cada sector mediante la suma de todas las respuestas dentro de una ventana de orientación móvil cubriendo un ángulo de PI/3.

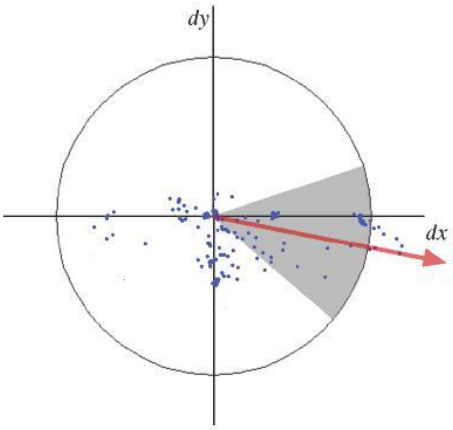

Figura 12 – Calculo direcciones x e y

Descriptores SURF

Se construye como primer paso una región cuadrada de tamaño 20s alrededor del punto de interés y orientada en relación a la orientación calculada en la etapa anterior. Esta región es a su vez dividida en 4 x 4 sub-regiones dentro de cada una de las cuales se calculan las respuestas de Haar de puntos con una separación de muestreo de 5 x 5 en ambas direcciones. Por simplicidad, se consideran dx y dy las respuestas de Haar en las direcciones horizontal y vertical respectivamente relativas a la orientación del punto de interés.

Para dotar a las respuestas dx y dy de una mayor robustez ante deformaciones geométricas y errores de posición, éstas son ponderadas por una gaussiana de valor *o* = 3.3s centrada en el punto de interés. En cada una de las sub-regiones se suman las respuestas dx y dy obteniendo así un valor de dx y dy representativo por cada una de las sub-regiones. Al mismo tiempo se realiza la suma de los valores absolutos de las respuestas j dx j y j dy j en cada una de las sub-regiones, obteniendo de esta manera, información de la polaridad sobre los cambios de intensidad.

En resumen, cada una de las sub-regiones queda representada por un vector v de componentes:

$$\nu = \left(\sum d_x, \sum d_y, \sum |\, d_x \,|, \sum |\, d_y \,|\right)$$

y por lo tanto, englobando las 4 x 4 sub-regiones, resulta un descriptor SURF con una longitud de 64 valores para cada uno de los puntos de interés identificados.

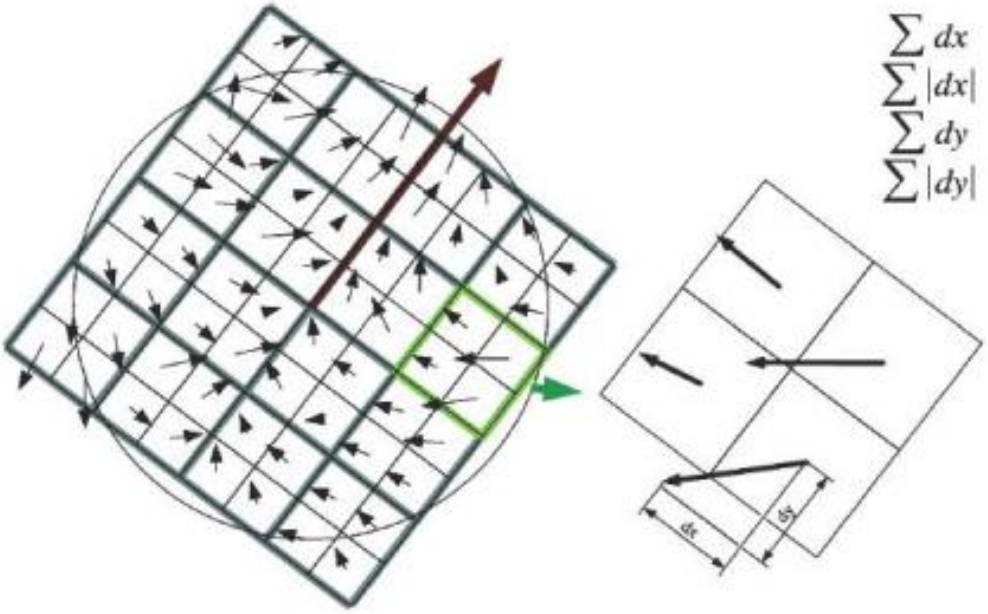

Figura 13 – **Descriptores SURF**

4. Experimentos

Después de toda la explicación dada en los anteriores puntos, vamos a pasar a definir en que consiste exactamente nuestro proyecto y a explicarlo paso por paso.

Nuestro proyecto consiste en un sistema cognitivo de visión en un robot en movimiento.

Esto quiere decir que nuestro robot será capaz de decir donde está exactamente, será capaz de detectar y comparar lugares, a partir de la base de datos que le hemos suministrado.

Entonces nos centraremos en unos videos de la escuela, en la que se hacen un recorrido en un sentido y un recorrido en el sentido inverso.

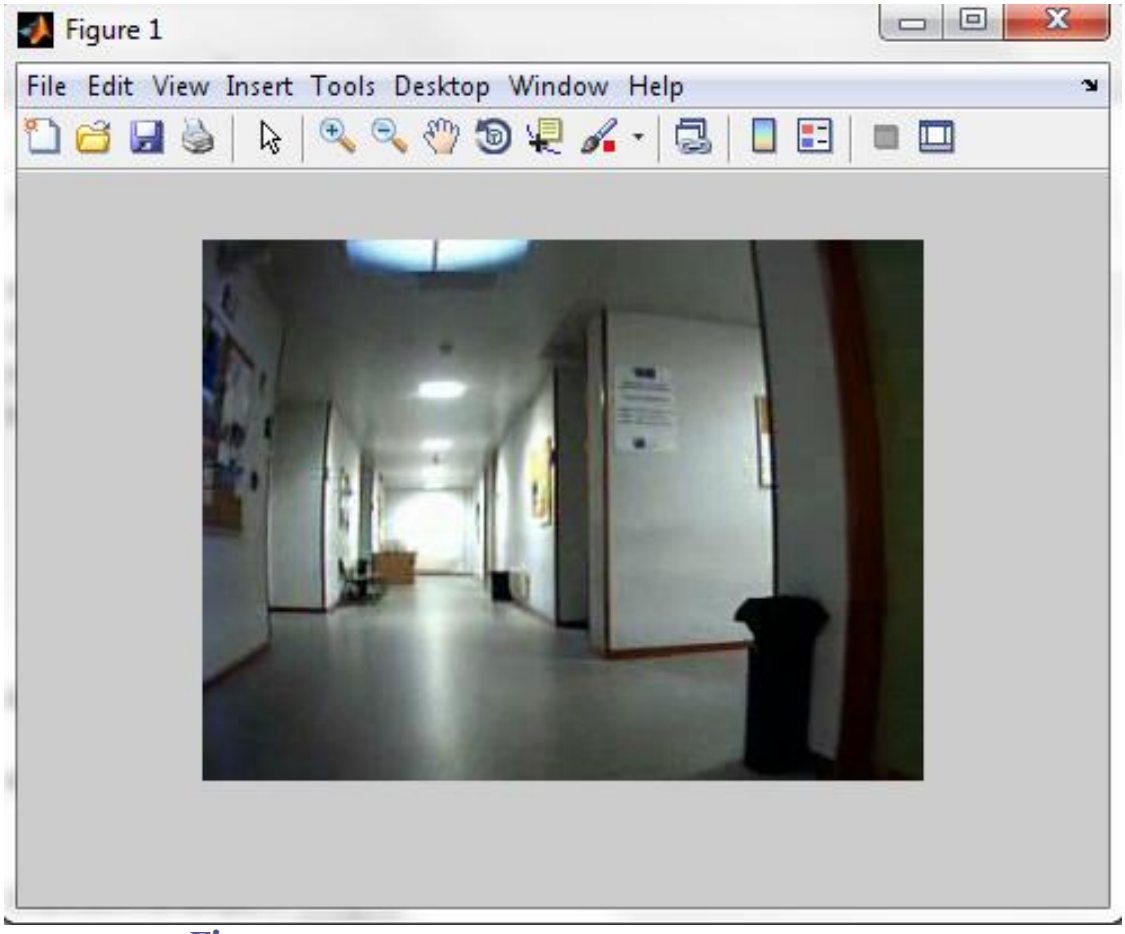

Figura 14 – **Imagen del video de la escuela de Industriales**

SIFT

En nuestro proyecto, en el apartado de SIFT, como hemos explicado anteriormente, nos hemos decantado por la versión del algoritmo implementado pos Andrea Vivaldi, denominado VLFEAT, el cual hemos escogido por ser un algoritmo sencillo de entender y fácil de utilizar para nuestro propósito.

Los pasos a seguir en este proyecto serían los siguientes, como hemos explicado con anterioridad:

- Creación de la base de datos a partir de los videos de entrenamiento.
- Creación y asignación de un vector de clases del video de test para luego ser usado.
- Ejecución del algoritmo para la comparación de imágenes (NN).

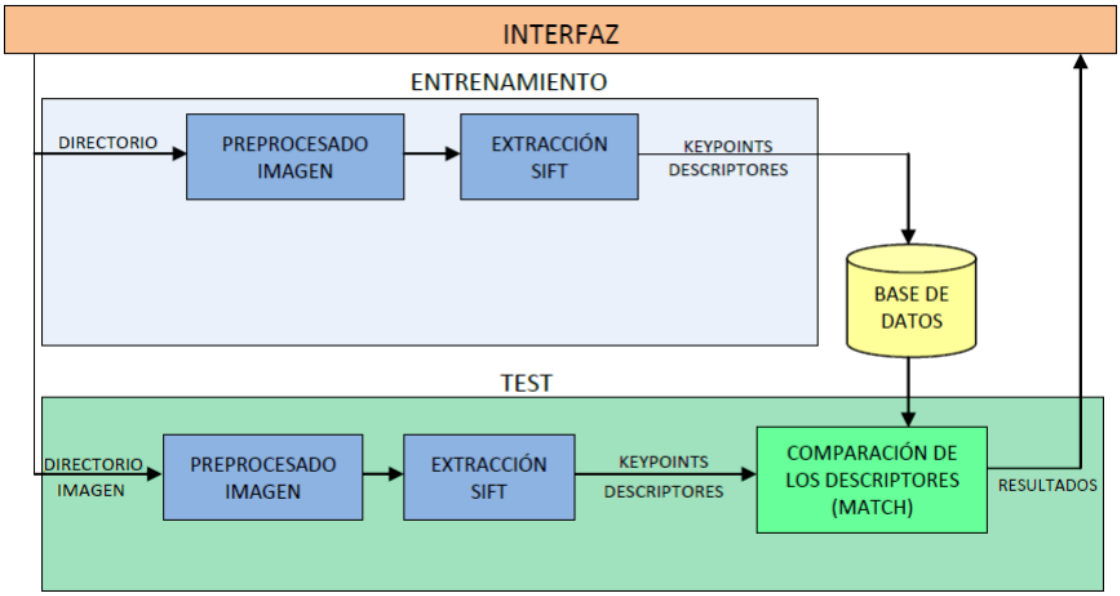

Figura 15 – Esquema para realizar la clasificación SIFT

Creación de la base de datos a partir de los videos de entrenamiento

En este paso como bien hemos descrito en el apartado SIFT de esta misma memoria, nos dedicaremos a encontrar los puntos de interés (keypoints) y sus respectivos descriptores para crear la base de datos que usaremos a lo largo de todo el proyecto para poder así comparar las imágenes de los diferentes videos.

El código a utilizar será el siguiente:

```
c=1;
a=[];
descriptores=[];
obj = mmreader('usc-dec-1.avi');

for i=526:664
%del 526 al 664 pasillo1
fprintf('Leyendo el frame: %d\n',i);
fprintf('\n');
%                       vid = read(obj,i);
vid = read(obj,i);
f = vid;
rgb = single(rgb2gray(f)) ;
% imshow(vid);
% f = getframe;
% [im,map] = frame2im(f);
% im = imresize(im, [240 320]);
% if isempty(map)
% rgb = im;
% else
% rgb = ind2rgb(im,map);
% end
% rgb = single(rgb2gray(rgb)) ;
[f,d] = vl_sift(rgb) ;
descriptores=[descriptores d];
tamanyo=tamanyo+length(d);

for j=(length(descriptores)-length(d):length(descriptores))
a(j)=c;

end

end
```

Esto sería un fragmento de nuestro extenso código , cogiendo como ejemplo los frames del video que van desde el 526 hasta el 664.

Vamos a explicar por partes el trozo de código expuesto anteriormente.
La primera línea, la que contiene la instrucción “c=1” lo que hacemos es declararnos la variable fija , que luego usaremos para rellenar el vector de clases

descrito posteriormente , llamado a , para poder así clasificar cada frame con que corresponde , si con el pasillo 1, la habitación 2, etc.

En la siguiente línea lo que hacemos es declararnos el vector vacío que contendrá a las clases de los frames.

Este vector acabaría teniendo la siguiente forma:

	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1

Figura 16 – **Vector de clases de video entrenamiento**

Con lo que el vector es un vector del tipo single de enteros, con el siguiente formato:

a= 1 x numero_de_descriptores

La línea que contiene la instrucción “descriptores=[];” es el vector que utilizaremos para almacenar los descriptores de cada keypoint , uno detrás de otro , esto es lo que denominaríamos nuestra base de datos completa, ya que contendría tanto los descriptores del video1 de entrenamiento como los del videio2 de entrenamiento.

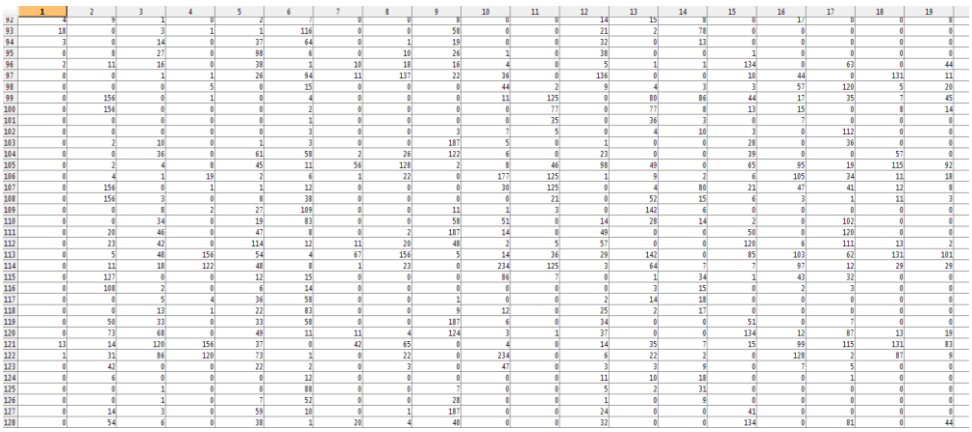

Figura 17 – **Matriz de descriptores (Base de datos)**

En los que podemos observar que el vector tendría la siguiente forma:

Descriptores= 128 x numero_de_keypoints

Donde los 128 son el numero de descriptores que saca nuestro algoritmo SIFT para cada keypoint.

La siguiente instrucción, la que contiene la linea "obj = mmreader('usc-dec-1.avi');" es una función de matlab que permite crear un objeto a partir del video y dejarlo en el espacio de trabajo de matlab para poder ser utilizado después.

Ya dentro del bucle de cada frame, podemos ver que lo que necesitamos es pasar el frame que obtenemos con la función "vid = read(obj,i);" a escala de grises con la instrucción "rgb = single(rgb2gray(f));", ya que el algoritmo SIFT trabaja imágenes en escala de grises.

Figura 19 – Imagen normal

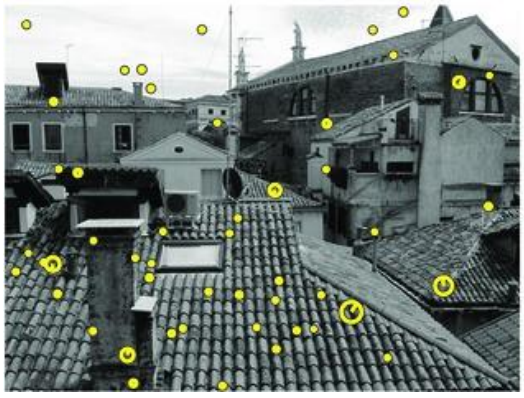

Figura 18 – Imagen en escala de grises y con los keypoint

Luego ya ejecutamos la instrucción que utiliza el algoritmo SIFT que hemos seleccionado "[f,d] = vl_sift(rgb);" que lo que hace es crear a partir de la imagen y los datos que extrae, un una matriz que contiene una matriz f y una matriz d, en la que podemos observar que la matriz f contiene la localización de la los keypoints en la primera y segunda fila (x,y), la escala en la tercera fila y la orientación en la cuarta fila.

La matriz D en cambio contiene diversas columnas , en lo que cada columna representa un keypoint diferente y contiene los 128 descriptores que necesitamos y luego utilizaremos.

Variable Editor - f

Stack: Base

f <4x247 double>

	1	2	3	4	5	6	7
1	4.6566	4.6566	9.9814	9.9268	10.7146	10.7146	11.0292
2	225.8061	225.8061	113.8911	230.6998	61.8552	61.8552	93.1926
3	2.0982	2.0982	2.0037	2.1443	2.0000	2.0000	1.8465
4	1.4754	-0.4218	1.2494	1.0614	1.4008	0.5262	0.5422

Figura 20 – Matriz con las diferentes características de los keypoint

Variable Editor - d

Stack: Base

d <128x247 uint8> Print

	1	2	3	4	5	6	7
1	41	0	54	53	34	0	38
2	0	0	32	0	27	41	25
3	0	0	1	0	0	23	12
4	0	0	0	0	0	0	2
5	13	0	10	11	69	0	4
6	13	0	55	33	39	33	34
7	43	0	24	34	0	43	42
8	88	0	17	43	0	0	44
9	21	9	14	64	66	0	114

Figura 21 – Matriz de descriptores

Si quisiéramos ver como saca la imagen y los datos solo tendríamos que plotear los valores de la siguiente manera:

```
obj = mmreader('usc-dec-1.avi');
for i=1:1
% del 1 al 199 pasillo1
fprintf('Leyendo el frame: %d\n',i);
fprintf('\n');
image(vid);
vid = read(obj,i);
f = vid;
rgb = single(rgb2gray(f)) ;

 [f,d] = vl_sift(rgb) ;
 perm = randperm(size(f,2)) ;
 sel = perm(1:50) ;
 %h1 = vl_plotframe(f(:,sel)) ;
 h2 = vl_plotframe(f(:,sel)) ;
 %set(h1,'color','k','linewidth',3) ;
 set(h2,'color','y','linewidth',2) ;
 h3 = vl_plotsiftdescriptor(d(:,sel),f(:,sel)) ;
set(h3,'color','g') ;
end
```

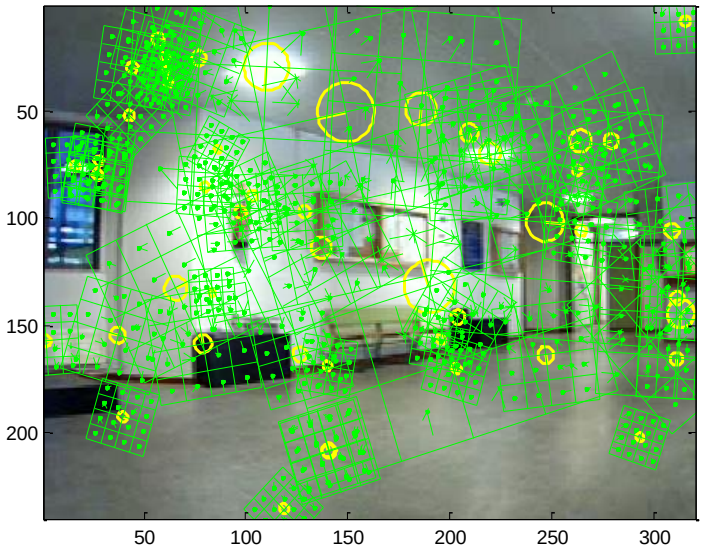

Figura 22 – **En amarillo: Keypoint. Verde: Descriptores**

Las siguientes instrucciones servirían para almacenar la información en los vectores que hemos creado anteriormente, el denominado descriptores, que como su nombre indica guardaríamos todos los descriptores de los dos videos de entrenamiento, y el vector a, que contendría las clases de cada frame que contiene los videos de entrenamiento.

Para que nuestra base de datos quede lista, solo nos quedaría realizar unas ultimas instrucciones de MATLAB, las necesarias para almacenar los vectores en ficheros .mat, que son los que utilizaremos enla segunda parte de nuestro experimento para cargar las bases de datos creadas anteriormente.

El código sería el siguiente:

```
savefile = 'BdatosCompleta.mat';
save(savefile, 'descriptores');
savefile2='ClasesCompleta.mat';
save(savefile2,'a');
```

Lectura de videos de test y clasificación a través del algoritmo NN.

Una vez tenemos creada nuestra base de datos con todos los descriptores y las clases de los dos videos de entrenamiento, ahora toca poner en practica los videos de test para poder comprobar el funcionamiento de la técnica SIFT en el tema que se ha elegido para este proyecto, a través del algoritmo NN, o lo que es lo mismo, el vecino mas próximo. Esto nos proporcionará la obtención de resultados para saber cuanto es de rápido el algoritmo SIFT, y sobre todo , cuanto es de fiable , si tiene mejor porcentaje de acierto que el otro algoritmo que describiremos mas adelante, el SURF.

El código que usaremos para realizar nuestro experimento es el siguiente:

```
entrenamiento=load('BdatosCompleta.mat','descriptores');
entrenamiento=single(entrenamiento.descriptores);
clas=load('ClasesCompleta.mat','a');
clas=single(clas.a);
clasestest=zeros(1,1817);
descriptorestest=[];
creapruebaclases;
m=0;
Resul1=[];
obj = mmreader('usc-dec-3.avi');
for i=1:obj.NumberOfFrames

pasillo1=0;
pasillo2=0;
pasillo3=0;
hab1=0;
hab2=0;
hab3=0;
hab4=0;
vid = read(obj,i);

f = vid;

rgb = single(rgb2gray(f)) ;
[f,d] = vl_sift(rgb) ;
descriptorestest= d;

fprintf('Leyendo el frame: %d\n',i);
fprintf('\n');



s=NN(entrenamiento,clas,single(descriptorestest),1);

for j=1:length(s)
    if s(j)==1
        pasillo1=pasillo1 +1;
    end
      if s(j)==2
        pasillo2=pasillo2 +1;
      end
      if s(j)==3
        pasillo2=pasillo3 +1;
      end
      if s(j)==11
        hab1=hab1 +1;
```

```
      end
      if s(j)==12
        hab2=hab2 +1;
      end
      if s(j)==13
        hab3=hab3 +1;
      end
      if s(j)==14
        hab4=hab4 +1;
      end

end

if pasillo1>pasillo2  && pasillo1>pasillo3 && pasillo1>hab1 &&
pasillo1>hab2 && pasillo1>hab3 && pasillo1>hab4
    clasestest(i)=1;
end

if pasillo2>pasillo1  && pasillo2>pasillo3 && pasillo2>hab1 &&
pasillo2>hab2 && pasillo2>hab3 && pasillo2>hab4
    clasestest(i)=2;
end
if pasillo3>pasillo1  && pasillo3>pasillo2 && pasillo3>hab1 &&
pasillo3>hab2 && pasillo3>hab3 && pasillo3>hab4
    clasestest(i)=3;
end
if hab1>pasillo1  && hab1>pasillo2 && hab1>pasillo3 && hab1>hab2 &&
hab1>hab3 && hab1>hab4
    clasestest(i)=11;
end
if hab2>pasillo1  && hab2>pasillo2 && hab2>pasillo3 && hab2>hab1 &&
hab2>hab3 && hab2>hab4
    clasestest(i)=12;
end
if hab3>pasillo1  && hab3>pasillo2 && hab3>pasillo3 && hab3>hab1 &&
hab3>hab2 && hab3>hab4
    clasestest(i)=13;
end
if hab4>pasillo1  && hab4>pasillo2 && hab4>pasillo3 && hab4>hab1 &&
hab4>hab2 && hab4>hab3
    clasestest(i)=14;
end


if(clasestest(i)==clasesprueba1(i))
 m=m+1;
end
 Resul1(i)=[m/i];
 R=Resul1(i)*100;
 fprintf('Porcentaje de acierto: %d \n',R);
 fprintf('\n');
fprintf('\n');fprintf('\n');
fprintf('\n');
 savefile = 'Test1.mat';
 save(savefile, 'clasestest');

 savefile= 'ResultadoTest1.mat';
 save(savefile,'Resul1');

end
```

El cual vamos a explicar detenidamente a continuación, paso por paso , de una manera sencilla lo que hacemos para poder obtener los resultados de clasificación, donde la parte clave del código es el método NN, que acto seguido explicaremos como funciona.

Lo primero que necesitaremos hacer será crearnos las variables necesarias para poder utilizarlas luego en los métodos, donde las variables "entrenamiento" y "clas" son las variables que contendrán la base de datos creada anteriormente , con los descriptores y las clases respectivamente, ejecutaremos el método "creapruebaclases", donde nos crearemos un vector que contendrá la clase de cada frame del video , generada por nosotros , que nos servirá para saber si lo que nosotros sabemos que es el pasillo numero 1, al ejecutar el test , nos da que ese mismo frame para el código es el pasillo 1 o es el pasillo 2, con lo que podremos obtener una tasa de acierto mas adelante.

Mas adelante lo que hacemos es leer el video como hacíamos anteriormente cuando creábamos la base de datos , y obtenemos un los descriptores con el método "vl_sift", cosa que ya explicamos como funcionaba en el apartado anterior. Al crear esta matriz de descriptores, es donde toma un papel importante el método NN, que dispone del siguiente código:

```
function test_targets = NN(train_patterns, train_targets, test_patterns, K)

% Classify using the Nearest neighbor algorithm
% Inputs:
%   train_patterns  - Train patterns (data)
%   train_targets   - Train targets  (classes)
%   test_patterns   - Test  patterns (classes)
%   K               - Number of nearest neighbors
%
% Outputs
%   test_targets    - Predicted targets (classes)

L  = length(train_targets);
Uc = unique(train_targets);

if (L < K),
  error('You specified more neighbors than there are points.')
end

N           = size(test_patterns, 2);
test_targets = zeros(1,N);

for i = 1:N,
    dist           = sum((train_patterns - test_patterns(:,i)*ones(1,L)).^2);
    [m, indices]    = sort(dist);
    n              = hist(train_targets(indices(1:K)), Uc);
    [m, best]       = max(n);
    test_targets(i) = Uc(best);
end
```

Al que se le pasa como argumentos la base de datos creada de descriptores, las clases de los frames obtenidas para la base de datos, las clases de los frames que hemos creado nosotros mismos para clasificar los frames de los test, y una constante K que es el numero de vecinos que vamos a encontrar, que por defecto será 1.

Resumiendo lo que hace el algoritmo es lo siguiente:

$$dif_i = \sqrt{(a_i - b_i)^2}$$

$$dif\_total = \sum_{1}^{128} dif_i$$

Donde *difi* es la diferencia euclídea entre el elemento *ai* de un descriptor de la imagen A, y *bi* su correspondiente de un descriptor de la imagen B. La variable *dif_total* es la suma de las diferencias euclídeas que hay entre los 128 elementos de ambos descriptores.

Realizando esta serie de operaciones entre cada descriptor de la imagen A con cada uno de la imagen B, podremos decidir cuales son iguales, eligiendo siempre el que haya dado una diferencia euclídea total menor.

Siguiendo con el código anterior lo que hemos hacemos para terminar es una vez que tenemos las clases de los keypoints y descriptores del test , analizamos cual es el patrón/clase que mas se repite, porque este será la clase del frame del test del video de test.

El método se realiza tantas veces como frames hayan en el video de test , y se realiza todo 2 veces , uno por cada video.

En el siguiente capítulo analizaremos los resultados obtenidosy los compararemos.

SURF

En nuestro proyecto, en el apartado de SURF, como hemos explicado anteriormente , nos hemos decantado por la versión del algoritmo implementado pos Chris Evans, denominado OPENSURF MATLAB, el cual hemos escogido por ser un algoritmo sencillo de entender y fácil de utilizar para nuestro propósito, con mayor potencial que los demás.

El método que hemos utilizado, te permite sacar entre otras opciones diferentes , si quieres extraer 64 descriptores o 128, lo que nos viene francamente bien para poder decidir si es mejor mas o menos descriptores, y poder también comparar el SURF de 128 con el SIFT que también es de 128.

Cuando aplicamos el método del OPENSURF, nos encontramos con las siguientes cosas:

Figura 23 – **Ejecucion del código OpenSurf en 2 imagenes**

Como podemos ver, en el ejemplo que encontramos dentro de la implementación, es encontrar los keypoints de cada imagen, y luego los comparamos para poder ver cuales son iguales o casi idénticos.

Nosotros nos vamos a quedar con la primera parte del método, ya que, para poder clasificarlos ya tenemos nuestro algoritmo NN.

El método OPENSURF tiene este formato:

```
function ipts=OpenSurf(img,Options)
% Ipts = OpenSurf(I, Options)
%
% inputs,
```

```
%   I : The 2D input image color or greyscale
%   (optional)
%   Options : A struct with options (see below)
%
% outputs,
%   Ipts : A structure with the information about all detected
Landmark points
%     Ipts.x , ipts.y : The landmark position
%     Ipts.scale : The scale of the detected landmark
%     Ipts.laplacian : The laplacian of the landmark neighborhood
%     Ipts.orientation : Orientation in radians
%     Ipts.descriptor : The descriptor for corresponding point
matching
%
% options,
%   Options.verbose : If set to true then useful information is
%                     displayed (default false)
%   Options.upright : Boolean which determines if we want a non-
rotation
%                       invariant result (default false)
%   Options.extended : Add extra landmark point information to the
%                   descriptor (default false)
%   Options.tresh : Hessian response treshold (default 0.0002)
%   Options.octaves : Number of octaves to analyse(default 5)
%   Options.init_sample : Initial sampling step in the image (default
2)
% Add subfunctions to Matlab Search path
functionname='OpenSurf.m';
functiondir=which(functionname);
functiondir=functiondir(1:end-length(functionname));
addpath([functiondir '/SubFunctions'])

% Process inputs
defaultoptions=struct('tresh',0.0002,'octaves',5,'init_sample',2,'upri
ght',false,'extended',false,'verbose',false);
if(~exist('Options','var')),
    Options=defaultoptions;
else
    tags = fieldnames(defaultoptions);
    for i=1:length(tags)
         if(~isfield(Options,tags{i})),
Options.(tags{i})=defaultoptions.(tags{i}); end
    end
    if(length(tags)~=length(fieldnames(Options))),
        warning('register_volumes:unknownoption','unknown options
found');
    end
end

% Create Integral Image
iimg=IntegralImage_IntegralImage(img);

% Extract the interest points
FastHessianData.thresh = Options.tresh;
FastHessianData.octaves = Options.octaves;
FastHessianData.init_sample = Options.init_sample;
FastHessianData.img = iimg;
ipts = FastHessian_getIpoints(FastHessianData,Options.verbose);

% Describe the interest points
if(~isempty(ipts))
```

```
    ipts = SurfDescriptor_DecribeInterestPoints(ipts,Options.upright,
Options.extended, iimg, Options.verbose);
end
```

Donde como podemos observar, este método nos da un amplio abanico de opciones, ya que nos deja por ejemplo elegir si queremos rotación o no, si queremos 64 o 128 descriptores, seleccionar el umbral de Hesse, seleccionar el numero de octavas a analizar o si queremos hacer primero una prueba para luego hacer el método.

A nosotros de todo esto nos interesa la opción extended, que es la que nos deja seleccionar si queremos 64 o 128 descriptores, cosa que luego sabremos si es mejor o peor en función del porcentaje de acierto que tengamos.

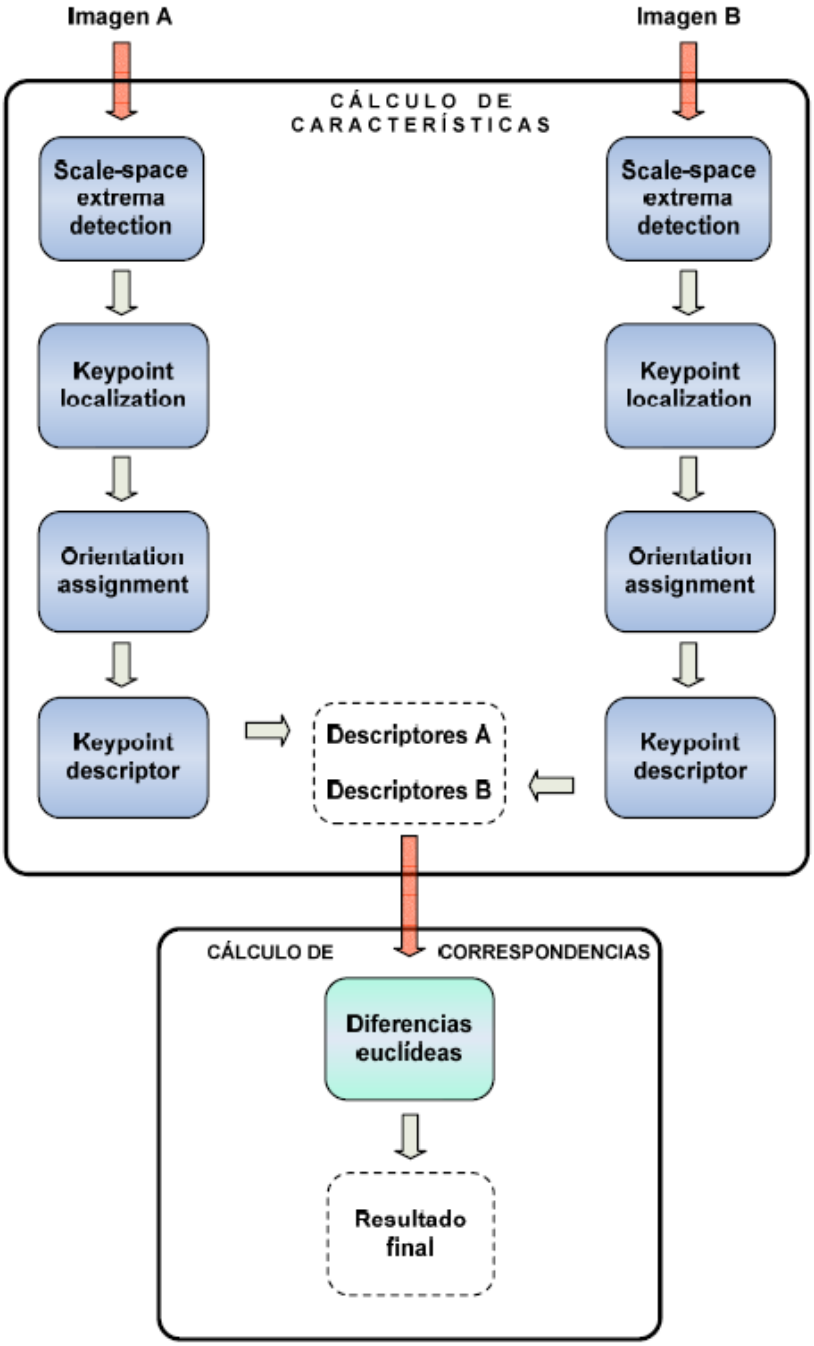

Figura 24 – Esquema a seguir para realizar nuestro experimento SURF

Creación de la base de datos a partir de los videos de entrenamiento

En este paso como bien hemos descrito en el apartado SURF de esta misma memoria, nos dedicaremos a encontrar los puntos de interés (keypoints) y sus respectivos descriptores para crear la base de datos que usaremos a lo largo de todo el proyecto para poder así comparar las imágenes de los diferentes videos.

El código a utilizar será el siguiente:

```
des=[];a=[];a2=[];a3=[];a4=[];a5=[];a6=[];a7=[];a8=[];a9=[];a10=[];
a11=[];a12=[];a13=[];a14=[];a15=[];a17=[];a18=[];a19=[];a20=[];a21=[];
a22=[];a23=[];a24=[];a25=[];a26=[];a16=[];
c=1;
descriptores=[];descriptores2=[];descriptores3=[];descriptores4=[];
descriptores5=[];descriptores6=[];descriptores7=[];descriptores8=[];
descriptores9=[];descriptores10=[];descriptores11=[];descriptores12=[]
;
descriptores13=[];descriptores14=[];descriptores15=[];
descriptores16=[];descriptores17=[];descriptores18=[];
descriptores19=[];descriptores20=[];descriptores21=[];
descriptores22=[];descriptores23=[];descriptores24=[];
descriptores25=[];descriptores26=[];obj = mmreader('usc-dec-1.avi');
for i=1:199
%del 1 al 199 pasillo1
fprintf('Leyendo el frame: %d\n',i);
fprintf('\n');
%                       vid = read(obj,i);
vid = read(obj,i);
f = vid;
rgb = single(rgb2gray(f)) ;

[puntos]=OpenSurf(rgb);
for k=1:length(puntos)
d=puntos(k).descriptor;
% % d=d';

descriptores=[descriptores d];
end
for j=1:length(descriptores)
a(j)=c;

end

%    hold on;
End
```

Vamos a explicar por partes el trozo de código expuesto anteriormente.

Primero creamos las variables para guardar las clases y los descriptores en diferentes matrices para al final juntarlos , ya que debido al coste computacional , resulta mas como trabajar con matrices pequeñas al principio. La siguiente linea, la que contiene la instrucción “c=1” lo que hacemos es declararnos la variable fija , que luego

usaremos para rellenar el vector de clases descrito posteriormente , llamado a , para poder así clasificar cada frame con que corresponde , si con el pasillo 1, la habitación 2, etc.

En la siguiente línea lo que hacemos es declararnos el vector vacio que contendrá a las clases de los frames.

Este vector acabaría teniendo la siguiente forma:

	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1

Figura 25 – Vector de clases

Con lo que el vector es un vector del tipo single de enteros, con el siguiente formato:

a= 1 x numero_de_descriptores

La línea que contiene la instrucción "descriptores=[];" por ejemplo es el vector que utilizaremos para almacenar los descriptores de cada keypoint , uno detrás de otro, esto es lo que denominaríamos nuestra base de datos cada cierto numero de frames.

	1	2	3	4	5	6	7
1	-0.0032	-0.0061	0.0438	0.0198	0.0120	0.0499	0.0049
2	0.0065	-0.0932	-0.0366	-0.0239	-0.0188	-0.0937	-0.0024
3	0.0170	0.0096	0.0958	0.0235	0.0137	0.0590	0.0069
4	0.0199	0.1021	0.0688	0.0319	0.0236	0.0984	0.0069
5	0.2872	-0.0056	-0.1417	0.0057	0.0072	0.0189	0.0041
6	0.0864	-0.1360	-0.0705	-0.0043	-0.0032	-0.0328	-0.0021
7	0.3652	0.0131	0.1899	0.0072	0.0074	0.0241	0.0079
8	0.0995	0.1549	0.0987	0.0086	0.0089	0.0379	0.0138
9	0.0283	-0.0081	-0.1878	0.0081	0.0063	0.0054	4.8993e-04
10	0.0022	-0.1219	0.0161	-0.0071	-0.0044	-0.0051	-0.0034
11	0.0950	0.0115	0.1975	0.0085	0.0076	0.0087	0.0159
12	0.0319	0.1367	0.0882	0.0140	0.0152	0.0118	0.0253

Figura 26 – Matriz descriptores

En los que podemos observar que el vector tendría la siguiente forma:

Descriptores= 128 x numero_de_keypoints

Descriptores= 64 x numero_de_keypoints

Donde los 128 son el numero de descriptores que saca nuestro algoritmo SURF para cada keypoint.

Como podemos ver , SURF utiliza doubles/float, con lo que en teoría aumenta fiabilidad del método frente al SIFT

La siguiente instrucción, la que contiene la linea "obj = mmreader('usc-dec-1.avi');" es una función de matlab que permite crear un objeto a partir del video y dejarlo en el espacio de trabajo de matlab para poder ser utilizado después.

Ya dentro del bucle de cada frame, podemos ver que lo que necesitamos es pasar el frame que obtenemos con la función "vid = read(obj,i);" a escala de grises con la instrucción "rgb = single(rgb2gray(f));", ya que el algoritmo SIFT trabaja imágenes en escala de grises.

Figura 28 – Imagen normal

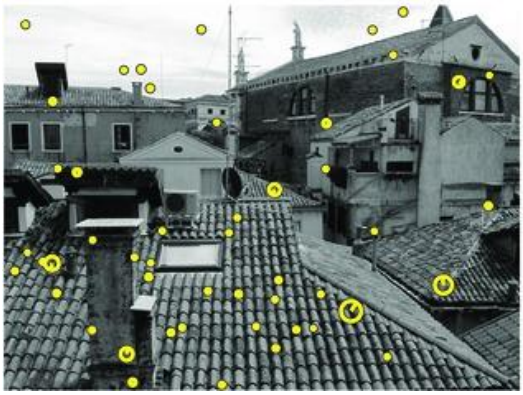

Figura 27 – Imagen en escala de grises con los keypoint en amarillo

Luego ya ejecutamos las instrucciones necesarias que utiliza el algoritmo SURF que hemos seleccionado que lo que hace es crear a partir de la imagen y los datos que extrae, un una matriz que contiene una matriz de structs de los diferentes puntos de el frame, que contienen la localización de los keypoints, la escala , la derivada de Laplace, la orientación, y los descriptores, y a nosotros lo que de verdad nos interesa es esta ultima matriz, que podemos sacar a partir del siguiente fragmento de código:

```
[puntos]=OpenSurf(rgb);
for k=1:length(puntos)
d=puntos(k).descriptor;
% % d=d';

descriptores=[descriptores d];
end
```

Ipts1(1,1) <1x1 struct>

Field	Value	Min	Max
x	230.1883	230.1883	230.1883
y	54.7183	54.7183	54.7183
scale	1.8869	1.8869	1.8869
laplacian	1	1	1
orientation	2.8326	2.8326	2.8326
descriptor	<64x1 double>	-0.3472	0.4287

Figura 29 – Estructura del keypoint en SURF

Las siguientes instrucciones servirían para almacenar la información en los vectores que hemos creado anteriormente , el denominado descriptores, que como su nombre indica guardaríamos todos los decriptores de los dos videos de entrenamiento, y el vector a, que contendría las clases de cada frame que contiene los videos de entrenamiento.

Para que nuestra base de datos quede lista, solo nos quedaría realizar unas ultimas instrucciones de MATLAB, las necesarias para almacenar los vectores en ficheros .mat, que son los que utilizaremos enla segunda parte de nuestro experimento para cargar las bases de datos creadas anteriormente.

El código sería el siguiente:

```
Atotal=[a a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17 a18
a19 a20 a21 a22 a23 a24 a25 a26];
Dtotal=[descriptores descriptores2 descriptores3 descriptores4
descriptores5 descriptores6 descriptores7 descriptores8 descriptores9
descriptores10 descriptores11 descriptores12 descriptores13
descriptores14 descriptores15 descriptores16 descriptores17
descriptores18 descriptores19 descriptores20 descriptores21
descriptores22 descriptores23 descriptores24 descriptores25
descriptores26 ];

savefile = 'BdatosCompletaSurf.mat';
save(savefile, 'Dtotal');
savefile2='ClasesCompletaSurf.mat';
save(savefile2,'Atotal');
```

Lectura de videos de test y clasificación a través del algoritmo NN.

Una vez tenemos creada nuestra base de datos con todos los descriptores y las clases de los dos videos de entrenamiento, ahora toca poner en practica los videos de test para poder comprobar el funcionamiento de la técnica SURF en el tema que se ha elegido para este proyecto, a través del algoritmo NN, o lo que es lo mismo, el vecino mas próximo. Esto nos proporcionará la obtención de resultados para saber cuanto es de rápido el algoritmo SURF, tanto de 64 como 128, y sobre todo , cuanto es de fiable , si tiene mejor porcentaje de acierto que el otro algoritmo que hemos descrito anteriormente, el SURF.

El código que usaremos para realizar nuestro experimento es el siguiente:

```
entrenamiento=load('BdatosCompletaSurf.mat','Dtotal');
entrenamiento=single(entrenamiento.Dtotal);
clas=load('ClasesCompletaSurf.mat','Atotal');
clas=single(clas.Atotal);
clasestest=zeros(1,1817);
descriptorestest=[];
creapruebaclases;
m=0;
Resul1=[];
obj = mmreader('usc-dec-3.avi');
for i=1:4
%1817 en total
descriptorestest=[];
pasillo1=0;
pasillo2=0;
pasillo3=0;
hab1=0;
hab2=0;
hab3=0;
hab4=0;
fprintf('empezamos');
vid = read(obj,i);
f = vid;
rgb = single(rgb2gray(f)) ;
% imshow(vid);
% f = getframe;
% [im,map] = frame2im(f);
% im = imresize(im, [240 320]);
% if isempty(map)
% rgb = im;
% else
% rgb = ind2rgb(im,map);
% end
% rgb = single(rgb2gray(rgb)) ;
Ipts1=OpenSurf(rgb);
  descriptorestest = single(reshape([Ipts1.descriptor],64,[]));
fprintf('Leyendo el frame: %d\n',i);
fprintf('\n');



s=NN(entrenamiento,clas,descriptorestest,1);

for j=1:length(s)
```

```
    if s(j)==1
        pasillo1=pasillo1 +1;
    end
      if s(j)==2
        pasillo2=pasillo2 +1;
      end
      if s(j)==3
        pasillo2=pasillo3 +1;
      end
      if s(j)==11
        hab1=hab1 +1;
      end
      if s(j)==12
        hab2=hab2 +1;
      end
      if s(j)==13
        hab3=hab3 +1;
      end
      if s(j)==14
        hab4=hab4 +1;
      end

end

if pasillo1>pasillo2  && pasillo1>pasillo3 && pasillo1>hab1 &&
pasillo1>hab2 && pasillo1>hab3 && pasillo1>hab4
    clasestest(i)=1;
end

if pasillo2>pasillo1  && pasillo2>pasillo3 && pasillo2>hab1 &&
pasillo2>hab2 && pasillo2>hab3 && pasillo2>hab4
    clasestest(i)=2;
end
if pasillo3>pasillo1  && pasillo3>pasillo2 && pasillo3>hab1 &&
pasillo3>hab2 && pasillo3>hab3 && pasillo3>hab4
    clasestest(i)=3;
end
if hab1>pasillo1  && hab1>pasillo2 && hab1>pasillo3 && hab1>hab2 &&
hab1>hab3 && hab1>hab4
    clasestest(i)=11;
end
if hab2>pasillo1  && hab2>pasillo2 && hab2>pasillo3 && hab2>hab1 &&
hab2>hab3 && hab2>hab4
    clasestest(i)=12;
end
if hab3>pasillo1  && hab3>pasillo2 && hab3>pasillo3 && hab3>hab1 &&
hab3>hab2 && hab3>hab4
    clasestest(i)=13;
end
if hab4>pasillo1  && hab4>pasillo2 && hab4>pasillo3 && hab4>hab1 &&
hab4>hab2 && hab4>hab3
    clasestest(i)=14;
end


if(clasestest(i)==clasesprueba1(i))
 m=m+1;
end
 Resul1(i)=[m/i];
 R=Resul1(i)*100;
```

```
 fprintf('Porcentaje de acierto: %d \n',R);
 fprintf('\n');
fprintf('\n');fprintf('\n');
fprintf('\n');
 savefile = 'Test1surf64.mat';
 save(savefile, 'clasestest');

 savefile= 'ResultadoTest1surf64.mat';
 save(savefile,'Resul1');


end
```

El cual vamos a explicar detenidamente a continuación, paso por paso , de una manera sencilla lo que hacemos para poder obtener los resultados de clasificación, donde la parte clave del código es el método NN, que acto seguido explicaremos como funciona.

Lo primero que necesitaremos hacer será crearnos las variables necesarias para poder utilizarlas luego en los métodos, donde las variables “entrenamiento” y “clas” son las variables que contendrán la base de datos creada anteriormente , con los descriptores y las clases respectivamente, ejecutaremos el método “creapruebaclases”, donde nos crearemos un vector que contendrá la clase de cada frame del video , generada por nosotros , que nos servirá para saber si lo que nosotros sabemos que es el pasillo numero 1, al ejecutar el test , nos da que ese mismo frame para el código es el pasillo 1 o es el pasillo 2, con lo que podremos obtener una tasa de acierto mas adelante.

Mas adelante lo que hacemos es leer el video como hacíamos anteriormente cuando creábamos la base de datos , y obtenemos un los descriptores con el método “opensurf”, cosa que ya explicamos como funcionaba en el apartado anterior. Al crear esta matriz de descriptores, es donde toma un papel importante el método NN, que dispone del siguiente código:

```
function test_targets = NN(train_patterns, train_targets, test_patterns, K)

% Classify using the Nearest neighbor algorithm
% Inputs:
%   train_patterns  - Train patterns (data)
%   train_targets   - Train targets  (classes)
%   test_patterns   - Test  patterns (classes)
%   K               - Number of nearest neighbors
%
% Outputs
%   test_targets    - Predicted targets (classes)

L  = length(train_targets);
Uc = unique(train_targets);

if (L < K),
   error('You specified more neighbors than there are points.')
end

N           = size(test_patterns, 2);
test_targets = zeros(1,N);
```

```
for i = 1:N,
   dist           = sum((train_patterns - test_patterns(:,i)*ones(1,L)).^2);
   [m, indices]    = sort(dist);
   n              = hist(train_targets(indices(1:K)), Uc);
   [m, best]       = max(n);
   test_targets(i) = Uc(best);
end
```

Al que se le pasa como argumentos la base de datos creada de descriptores, las clases de los frames obtenidas para la base de datos, las clases de los frames que hemos creado nosotros mismos para clasificar los frames de los test, y una constante K que es el numero de vecinos que vamos a encontrar, que por defecto será 1.

Resumiendo lo que hace el algoritmo es lo siguiente:

$$dif_i = \sqrt{(a_i - b_i)^2}$$

$$dif\_total = \sum_{1}^{128} dif_i$$

Donde *difi* es la diferencia euclídea entre el elemento *ai* de un descriptor de la imagen A, y *bi* su correspondiente de un descriptor de la imagen B. La variable *dif_total* es la suma de las diferencias euclídeas que hay entre los 128 elementos de ambos descriptores.

Realizando esta serie de operaciones entre cada descriptor de la imagen A con cada uno de la imagen B, podremos decidir cuales son iguales, eligiendo siempre el que haya dado una diferencia euclídea total menor.

Siguiendo con el código anterior lo que hemos hacemos para terminar es una vez que tenemos las clases de los keypoints y descriptores del test , analizamos cual es el patrón/clase que mas se repite, porque este será la clase del frame del test del video de test.

El método se realiza tantas veces como frames hayan en el video de test , y se realiza todo 2 veces , uno por cada video.

En el siguiente capítulo analizaremos los resultados obtenidos y los compararemos.

5. Resultados

En este capítulo se mostrarán los resultados obtenidos para cada uno de los métodos de estudio.

Primero lo que haremos será ver los resultados que nos ha proporcionado el algoritmo SIFT para clasificar ayudado del algoritmo del vecino mas próximo, y luego seguiremos con el algoritmo SURF, en sus dos variantes, la primera que utilizaremos para comprobar y comparar resultados con el SIFT será la de 128, ya que el numero de descriptores corresponde con el método anterior, y luego compararemos las dos técnicas SURF utilizadas, la de 128 con la de 64, suponiendo a priori que la de 128 debería ser mas exacta que la de 64 gracias al numero de descriptores, al igual que presuponemos que el algoritmo SURF tiene que ser mejor que el SIFT ya que es un método mejorado del segundo.

Hemos realizado pruebas similares con los dos algoritmos, con los mismos videos, cada uno con sus bases de datos de SIFT y SURF, por lo que las pruebas que hemos realizado podemos decir que son exactamente iguales, como hemos demostrado anteriormente en los códigos de matlab.

Vamos a empezar demostrando los resultados obtenidos en los test de SIFT y luego seguiremos con los de SURF.

SIFT

Vamos a empezar con el test1, o lo que es lo mismo, con el test que realiza la clasificación del video numero 3 de test (recordemos que los videos 1 y 2 eran los videos de entrenamiento, y los 3 y 4 son los que utilizábamos en los test).

Tras haber ejecutado el código, nos ha devuelto un vector con el número de acierto y las clases del video de test. El resultado que hemos obtenido de acierto en este test es de **87,5%**, lo que nos indica que tiene una alta fiabilidad respecto a la clasificación, ya que respecto a los 1817 frames que contiene el video 3, se han clasificado correctamente 1590.

El porcentaje que hemos sacado respecto al acierto del video 4 ha sido de **85,12%**, lo que nos indica que tiene una fiabilidad alta en los dos test realizados, aunque este test la tiene menor que el anterior, ya que de 1754 se han clasificado correctamente 1493.

Una vez hallados el porcentaje de acierto de cada test realizado con SIFT, tendríamos que sacar el promedio de acierto del algoritmo SIFT respecto al problema planteado.

Para saber el porcentaje medio total hay que multiplicar el porcentaje del primer test por el ratio sobre el total de frames (frames video 3 / (frames video 3 + frames video 4)) y sumarlo con el porcentaje del segundo

test por el ratio sobre el total de frames (frames video 4 / (frames video 3 + frames video 4)).

Esto quedaría de la siguiente manera:

87,5 x (1817/(1817+1754)) = 44,52 %

85,12 x (1754/(1817+1754)) = 41,81 %

Donde obtenemos un porcentaje medio de **86,13%**, algo que veremos si la implementación de SURF puede superar como se presupone, o no.

SURF

Vamos a empezar con el algoritmo SURF que nos devuelve 64 descriptores, ejecutando los dos test, y luego proseguiremos con el algoritmo de 128.

Tras haber ejecutado el código, nos ha devuelto un vector con el número de acierto y las clases del video de test. El resultado que hemos obtenido de acierto en este test es de **91,08%**, lo que nos indica que tiene una alta fiabilidad respecto a la clasificación, ya que respecto a los 1817 frames que contiene el video 3, se han clasificado correctamente 1653.

El porcentaje que hemos sacado respecto al acierto del video 4 ha sido de **89,97%**, lo que nos indica que tiene una fiabilidad alta en los dos test realizados, aunque este test la tiene menor que el anterior, ya que de 1754 se han clasificado correctamente 1578.

Una vez hallados el porcentaje de acierto de cada test realizado con SURF de 64 descriptores, tendríamos que sacar el promedio de acierto del algoritmo respecto al problema planteado.

Para saber el porcentaje medio total hay que multiplicar el porcentaje del primer test por el ratio sobre el total de frames (frames video 3 / (frames video 3 + frames video 4)) y sumarlo con el porcentaje del segundo test por el ratio sobre el total de frames (frames video 4 / (frames video 3 + frames video 4)).

Esto quedaría de la siguiente manera:

91,08 x (1817/(1817+1754)) = 46,34 %

89,97 x (1754/(1817+1754)) = 44,2 %

Donde obtenemos un porcentaje medio de **90,54%**

Ahora seguimos con los 128 descriptores

Tras haber ejecutado el código, nos ha devuelto un vector con el número de acierto y las clases del video de test. El resultado que hemos obtenido de acierto en este test es de **90,86%**, lo que nos indica que tiene una alta fiabilidad respecto a la clasificación, ya que respecto a los 1817 frames que contiene el video 3, se han clasificado correctamente 1651.

El porcentaje que hemos sacado respecto al acierto del video 4 ha sido de **89,05%**, lo que nos indica que tiene una fiabilidad alta en los dos test realizados, aunque este test la tiene menor que el anterior, ya que de 1754 se han clasificado correctamente 1562.

Una vez hallados el porcentaje de acierto de cada test realizado con SURF de 128 descriptores, tendríamos que sacar el promedio de acierto del algoritmo respecto al problema planteado.

Para saber el porcentaje medio total hay que multiplicar el porcentaje del primer test por el ratio sobre el total de frames (frames video 3 / (frames video 3 + frames video 4)) y sumarlo con el porcentaje del segundo test por el ratio sobre el total de frames (frames video 4 / (frames video 3 + frames video 4)).

Esto quedaría de la siguiente manera:

90,86 x (1817/(1817+1754)) = 46,23 %

89,05 x (1754/(1817+1754)) = 43,74 %

Donde obtenemos un porcentaje medio de **89,97%**

6. Conclusión

Cuando ya tenemos todos los resultados, solo nos quedaría comprobar cual ha tenido mas fiabilidad y cual ha tenido un mayor coste computacional para nosotros, aunque esto en nuestro proyecto es un poco secundario, lo que nos interesa es saber que algoritmo sería mas apropiado para la navegación robótica, para tener un mayor índice de acierto cuando nuestro robot vaya por los pasillos y habitaciones reconociendo cada sitio.

Tras haber obtenido los algoritmos, hay que decir que el coste de tiempo para obtener las bases de datos y concluir con los test, ha sido bastante menor con el algoritmo SIFT que con el SURF, pero la conclusión que sacamos con todo lo realizado en este proyecto es la siguiente:

Al realizar los tests con SIFT y SURF, notamos una amplia mejoría en el SURF respecto al SIFT. Su respuesta ante la rotación y el escalado es más satisfactoria que en el SIFT, y esto es lo que nos proporciona mayor numero de acierto en uno que en otro, en este caso el SURF, aunque sea por poco.

Además entre la versión de 64 descriptores y la versión de 128 descriptores de SURF, notamos una ligera superioridad en la versión de 64, pero esto puede que sea debido a que al tener menos descriptores, el fallo va a ser menor ya que tiene menos con que comparar, al ser una base de datos mas pequeña.

Entonces para concluir nuestro proyecto podemos decir que El algoritmo SURF, aunque temporalmente en ejecución sea mayor, es mas fiable que el algoritmo SIFT para ser usado para la navegación robotica.

7. APENDICE A : Matlab

HISTORIA

La primera versión de matlab data de los años 70, y fue diseñada como herramienta de apoyo para los cursos de Teoría de Matrices, Álgebra Lineal y Análisis Numérico. El nombre matlab es un acrónimo: “MATrix LABoratory”.

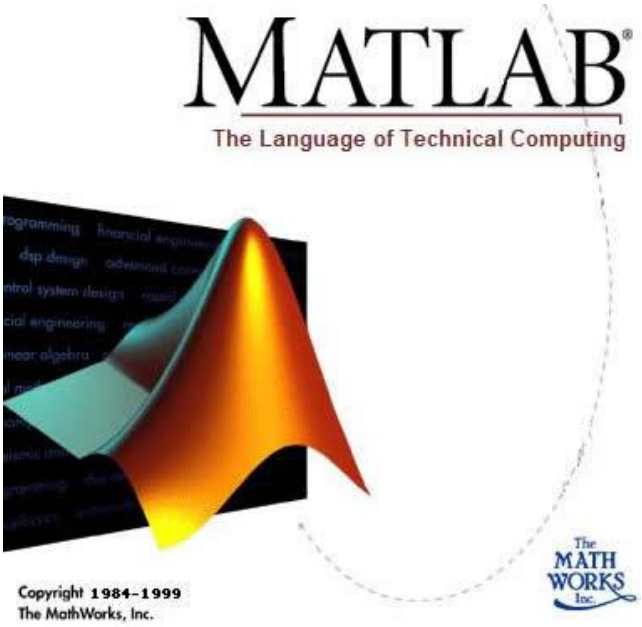

Figura 30 – Logotipo MATAB

DEFINICIÓN

El Lenguaje de Computación Técnica MATLAB es un ambiente de computación técnica integrada que combina computación numérica, gráficos y visualización avanzada y un lenguaje de programación de alto nivel.

MATLAB integra análisis numérico, cálculo matricial, proceso de señal y visualización gráfica en el mismo entorno.MATLAB dispone de una amplia gama de programas de apoyo especializados, denominados Toolbox. Estos cubren en la actualidad prácticamente casi todas las áreas principales en el mundo de la ingeniería y la simulación, destacando entre ellos el 'toolbox' de proceso de imágenes(que es el que usaremos nosotros, junto a implementaciones realizadas por entendidos en la materia), señal, control robusto, estadística, análisis financiero, matemáticas simbólicas, redes neurales,etc. es un entorno de cálculo técnico, que se ha convertido en estándar de la industria, con capacidades no superadas en computación y visualización numérica.

ENTORNO DE TRABAJO

El entorno operativo de Matlab se compone de una serie de ventanas en las cuales cada una tiene su función , en una tendremos por ejemplo el espacio de variables creadas y compartidas y en otra de ellas tendríamos la ventana para introducir comandos MATLAB.

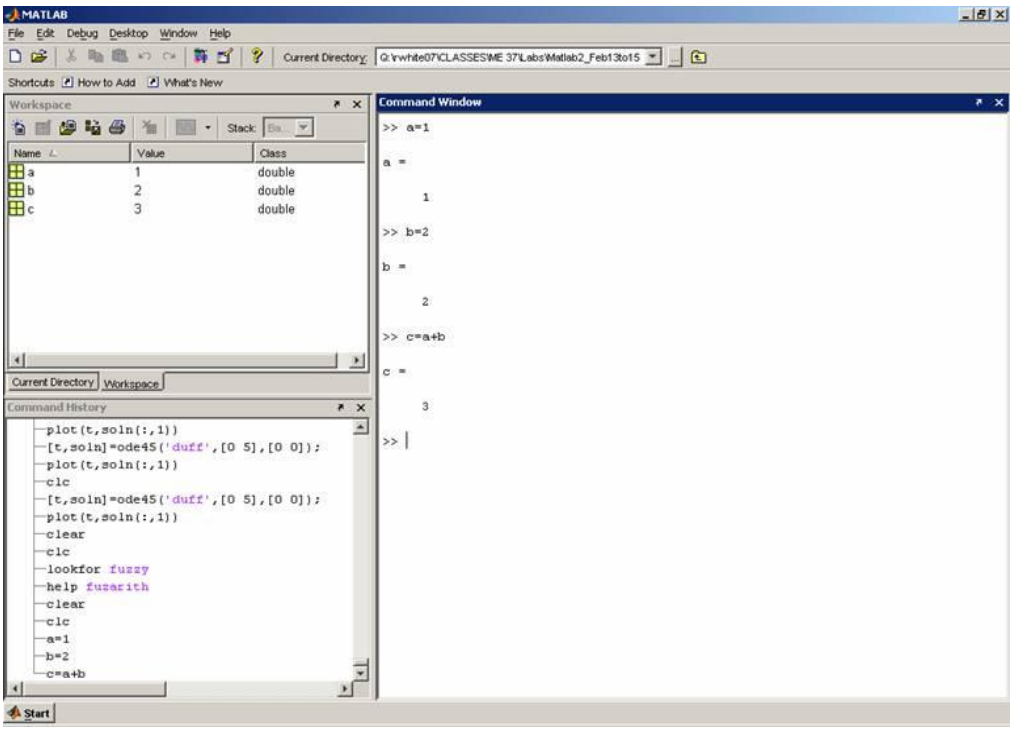

Figura 31 - ENTORNO DE TRABAJO MATLAB

- **Command Window**

Se utiliza para introducir órdenes directamente por el para ser ejecutadas en ese mismo instante. Los resultados de las órdenes introducidas se muestran en esta misma pantalla. Cuando se ejecuta un programa previamente escrito(código que es muy básico, muy parecido a java), que en Matlab recibe el nombre de M-file, los resultados también aparecen en esta ventana.

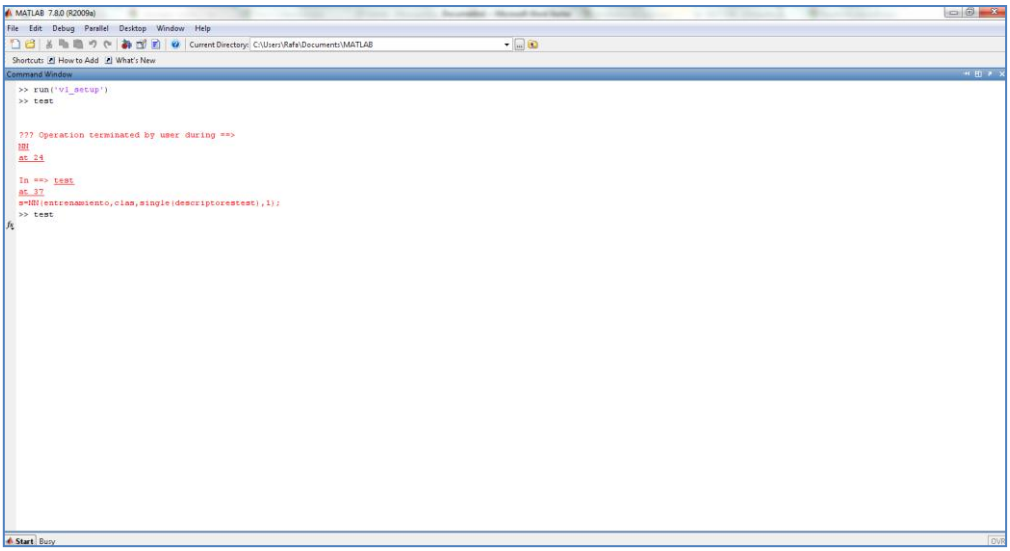

Figura 32 - COMMAND WINDOW MATLAB

- **Command history**

Las órdenes introducidas en la ventana command window quedan grabadas en esta ventana, de forma que, haciendo doble click sobre ellas, las podemos volver a ejecutar.

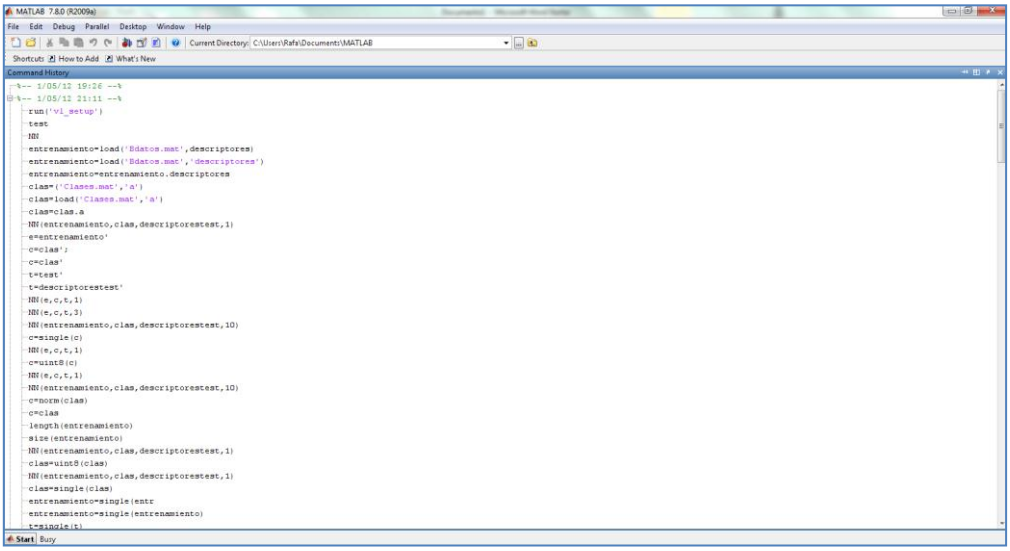

Figura 33 - COMMAND HISTORY MATLAB

- **Workspace**

Esta ventana contiene las variables (escalares, vectores, matrices, ...) creadas en la sesión de Matlab. La ventana workspace nos proporciona información sobre el nombre, dimensiones, tamaño y tipo de variable.

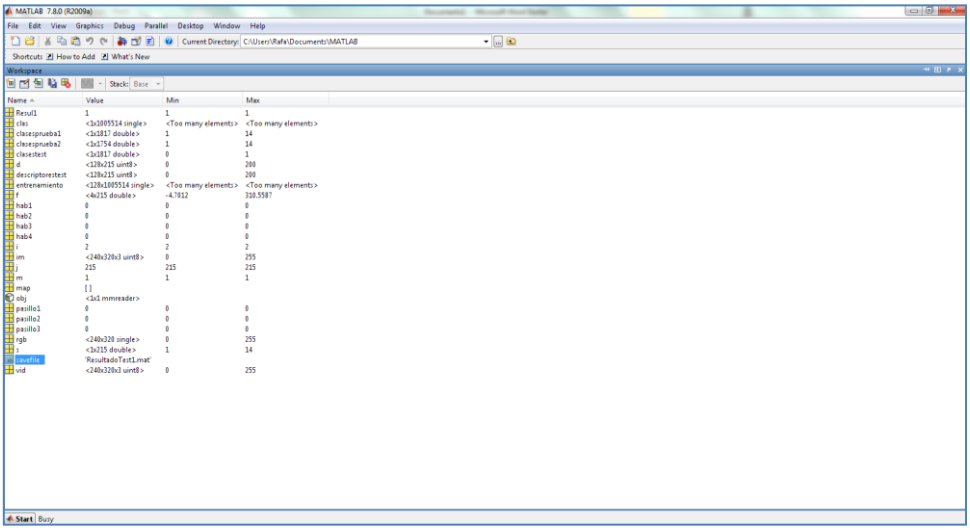

Figura 34 - WORKSPACE MATLAB

- **Current** directory

Las operaciones de Matlab utilizan el directorio seleccionado en current directory (a través del botón para explorar) como punto de referencia.

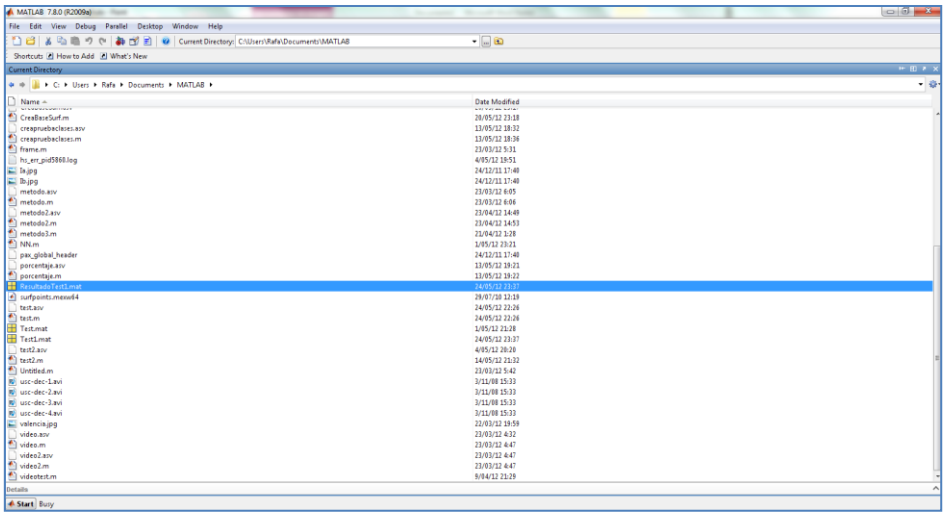

Figura 35 - CURRENT DIRECTORY MATLAB

En conclusión, usamos el Matlab porque es un programa matemático muy potente para realizar acciones matemáticas y de procesado de imagen que requieran de una gran potencia y de un entorno sencillo y manejable.

LENGUAJE DE PROGRAMACIÓN MATLAB

Matlab puede considerarse como un lenguaje de programación tal como C, Fortran, Java, etc. Algunas de las características de Matlab son:

- La programación es mucho mas sencilla.
- Hay continuidad entre valores enteros, reales y complejos.
- La amplitud del intervalo y la exactitud de los números es mayor.
- Cuenta con una biblioteca matemática amplia.
- Abundantes herramientas gráficas, incluidas funciones de interfaz gráfica con el usuario.
- Capacidad de vincularse con los lenguajes de programación tradicionales.
- Transportabilidad de los programas.

Algunas de sus desventajas son:

- Necesita de muchos recursos de sistema como son Memoria, tarjeta de videos, etc. para funcionar correctamente.
- El tiempo de ejecución es lento.
- No genera código ejecutable.
- Es caro.

Ejemplo Codigo Matlab:

```
... ...

for j=1:length(s)
   if s(j)==1
      pasillo1=pasillo1 +1;
   end
    if s(j)==2
      pasillo2=pasillo2 +1;
    end
    if s(j)==3
      pasillo2=pasillo3 +1;
    end
    if s(j)==11
      hab1=hab1 +1;
    end

... ...
```

```
if pasillo1>pasillo2  && pasillo1>pasillo3 && pasillo1>hab1 && pasillo1>hab2 &&
pasillo1>hab3 && pasillo1>hab4

    clasestest2(i)=1;

end


if pasillo2>pasillo1  && pasillo2>pasillo3 && pasillo2>hab1 && pasillo2>hab2 &&
pasillo2>hab3 && pasillo2>hab4

    clasestest2(i)=2;

end

if pasillo3>pasillo1  && pasillo3>pasillo2 && pasillo3>hab1 && pasillo3>hab2 &&
pasillo3>hab3 && pasillo3>hab4

    clasestest2(i)=3;

end

if hab1>pasillo1  && hab1>pasillo2 && hab1>pasillo3 && hab1>hab2 && hab1>hab3
&& hab1>hab4

    clasestest2(i)=11;

end


... ...


if(clasestest2(i)==clasesprueba2(i))

 m=m+1;

end

 Resul2(i)=[m/i];

 R=Resul2(i)*100;

 fprintf('Porcentaje de acierto: %d \n',R);

 fprintf('\n');

fprintf('\n');fprintf('\n');

fprintf('\n');

 savefile = 'Test2.mat';

 save(savefile, 'clasestest');


 savefile= 'ResultadoTest2.mat';
```

```
save(savefile,'Resul2');



end
```

8. Bibliografía

- http://arantxa.ii.uam.es/~jms/pfcsteleco/lecturas/20100915DavidOtero.pdf
- http://upcommons.upc.edu/pfc/bitstream/2099.1/8052/1/memoria.pdf
- http://epsilon.cnnet.upr.edu/mmarcano/cursos/ananumI/introduccion_matlab.pdf
- http://www.recercat.net/bitstream/handle/2072/92896/PFC_ArturoBonninLlofriu.pdf;jsessionid=18D88A0848DB3BA67F9AE59D6DF89076.recercat1?sequence=1
- http://www.mathworks.com/matlabcentral/fileexchange/28300-opensurf-including-image-warp
- http://www.vlfeat.org/overview/sift.html
- http://www.diphernet.com/web/cvision/dea_j/deaweb.html
- http://amitsarangi.wordpress.com/2010/05/24/surf-and-opensurf/
- http://en.wikipedia.org/wiki/SURF
- http://code.google.com/p/opensurf1/
- http://en.wikipedia.org/wiki/Scale-invariant_feature_transform
- Lowe, David G. (1999). "Object recognition from local scale-invariant features". Proceedings of the International Conference on Computer Vision. 2. pp. 1150–1157
- Lowe, D. G., “Distinctive Image Features from Scale-Invariant Keypoints”, International Journal of Computer Vision, 60, 2, pp. 91-110, 2004.