Examen práctico de Visión Artificial
12 de Enero, 2015

Ejercicio 1

Dado un conjunto de imágenes que han sido tomadas apuntando la cámara en direcciones ligeramente diferentes (Fig.1), imaginemos que tenemos que crear con ellas una imagen panorámica utilizando una aplicación que hemos encontrado en la web. Esta aplicación necesita como entrada la secuencia de imágenes a juntar en el orden correcto. Queremos automatizar este proceso para poder replicarlo con cualquier conjunto de imágenes.

Fig.1 *Imágenes captadas con cámara en movimiento.*

Para ello crearemos una función **ex1.m**, que utilizando el método *computeMatches*, que a partir de los descriptores *Sift*, calcula los puntos correspondientes entre cada par de imágenes. Para esto, cread las funciones necesarias que se llaman desde **ex1.m** con la siguiente funcionalidad:

a) **[1 punto]** La función *constructList(nombre, numeroImagenes, extension)* que construye la lista de nombres de las imágenes.

Por ejemplo:
>> imageList=constructList('llanes_',3,'jpg')
imageList =

'llanes_1.jpg' 'llanes_2.jpg' 'llanes_3.jpg'

Nota: Para construir el nombre de las imágenes, podéis utilizar el comando sprintf o strcat de Matlab, que os permitirá concatenar las partes fijas del nombre de fichero ('llanes_', '.jpg') y su número una vez convertido en string (int2str).
La lista de nombres debe ser de tipo cell (help: cell). Un vector/lista de tipo cell en Matlab es una lista de elementos que permite guardar una lista de cadenas de caracteres (p.e. los nombres de los ficheros). La inicialización es: lista={}, y la forma de indexar los elementos es la misma como en los vectores (lista(3)).

b) **[1 punto]** La función *readAndVisualize(imageList)*, que lea y visualiza una tras otra todas las imágenes de la lista de imágenes (imageList).

Nota: Si necesitáis convertir un elemento en string, podéis utilizar el comando char().

c) **[1.5 puntos]** La función *[distances]=calcularDiferencias(imageList, noImages)* calcula la disimilitud/diferencia de cada dos imágenes y retorna una matriz de disimilitudes (distancias), donde el elemento *distances(i,j)* de la matriz es la disimilitud/distancia entre las imágenes *i* e *j* de la lista *imageList*.

Nota 1: Puedes utilizar como criterio para calcular la disimilitud entre cada par de imágenes, la suma promedia de los 'scores' de las correspondencias que retorna el matching del Sift. Recuerda que el score es la distancia/diferencia entre los descriptores de los puntos Sift.

Nota 2: Para utilizar la librería del Sift (en particular, el método *computeMatches*), primero debes ejecutar el método *vl_setup* de la carpeta *toolbox* adjuntada al enunciado del examen.

Una posible solución del resultado de la función calcularDiferencias sería:

distances =

1.0e+07 *

1.0000	0.0016	0.0011
0.0016	1.0000	0.0011
0.0011	0.0011	1.0000

d) **[1.5 puntos]** La función *[neworden]=ordenarPorSimilitud(distances, imageList, noImages)* ordena por la similitud de las imágenes y las visualiza en el orden correspondiente. Es decir, empezando desde una imagen cualquiera (p.e. la primera), determina la siguiente según la mínima diferencia con la anterior y así sucesivamente (p.e. neworden = [2,3,1]).

Fig.2 *Imágenes visualizadas en el orden correcto.*

Ejercicio 2

Imaginemos que tenemos una imagen de profundidad, que es un tipo de imagen donde cada píxel guarda la distancia de un punto en el Mundo real hasta la cámara (ver imagen en la Fig. 3 (izquierda)). Queremos separar la región que está más cerca de la cámara.

Dada la imagen depth.png, crear la función **ex2.m** de la cual se llamarán las siguientes funciones:

a) **[1 punto]** *visualizaImHist(nombreImagen)* que lee y visualiza la imagen conjuntamente con su histograma. Para visualizar la imagen con pseudocolores (Fig.3 centro), se puede utilizar el método *imagesc* de Matlab.

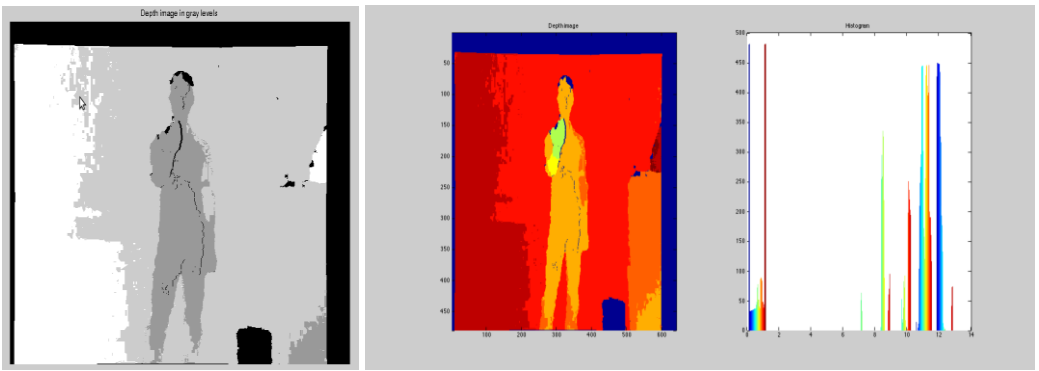

Fig. 3 *Imagen de profundidad (a la izquierda), su visualización con pseudocolores (centro) y su histograma (a la derecha).*

b) **[1 punto]** *[simplifiedImage,ctrs]=reduceNumReg(nombreImagen)* que reduce el número de regiones de la imagen utilizando el método *kmeans*, teniendo en cuenta que el objetivo es facilitar la segmentación de la zona más cercana (en este caso del antebrazo).

Nota: Recuerda que una imagen de profundidad, cada valor del píxel es la distancia a la cámara, por lo que valores pequeños indican puntos cercanos y valores grandes puntos alejados**.**

c) **[1 punto]** *[segmentedImage]=segmenta(nombreImagen,simplifiedImage,ctrs)* que segmenta la región del antebrazo tomando como input las salidas de la función anterior y visualiza en una misma figura la imagen original, el resultado de la segmentación con k-means y la segmentación de la zona más cercana.

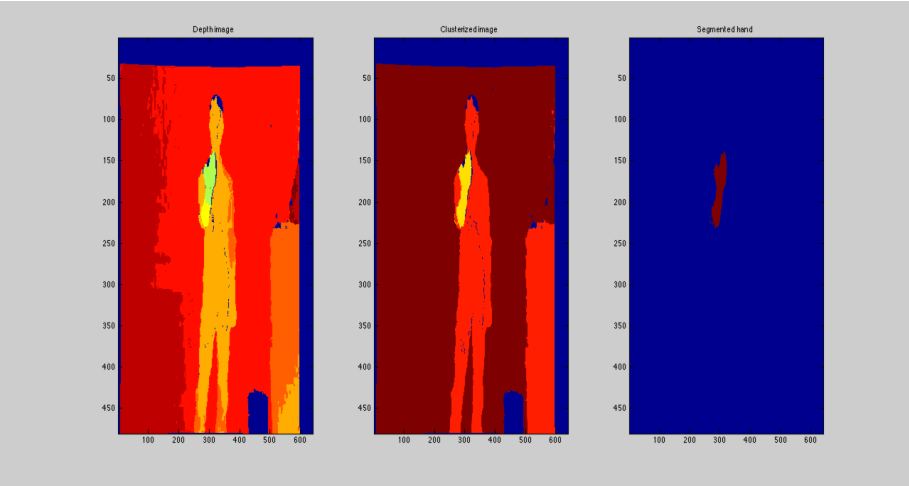

Fig.4 *Imagen original (izquierda), imagen resultante del k-means (centro) y región más cercana extraída automáticamente (derecha).*

Ejercicio 3

Queremos implementar un reconocedor de caras para desbloquear nuestro nuevo teléfono móvil. Para esta aplicación, necesitaremos desarrollar un detector de caras y un método de reconocimiento facial.

a) **[1 punto]** El detector de caras estará basado en características de Haar (ver Fig. 5). Una de las peculiaridades de las características de Haar es que se pueden evaluar de forma muy eficiente a partir de la imagen integral. Se pide que implementes una función ***[ii]=integralImage(img)***, que dada una imagen *img* (ya cargada), devuelva su imagen integral *ii*. Para asegurar que el método funciona correctamente, se debe comprobar que la imagen de entrada es en escala de grises (un sólo canal) y en caso que no lo sea, habrá que convertirla antes de calcular la imagen integral.

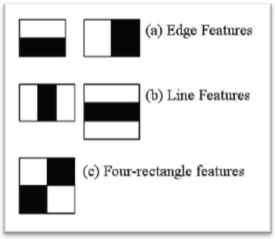

Fig 5: *Características de Haar*

b) **[1 punto]** Para el reconocimiento facial utilizaremos el método de los eigenfaces. Este método se basa en encontrar los vectores y valores propios a partir de la matriz de covarianza, permitiendo reducir la dimensión de los datos. Implementa la función ***[base]=selectVectors(vec, val)***, que

dada la matriz de vectores propios *vec* (cada fila es un vector propio) y la lista de valores propios *val*, devuelva en *base* el subconjunto de vectores propios seleccionados. Escribe como parte de la documentación del método el criterio que has utilizado para seleccionar los vectores.