Ejercicio 1: Escribir una función que reciba las 3 coordenadas XY de un triángulo origen y las correspondientes coordenadas XY' del triángulo destino y devuelva la matriz 3x3 de la correspondiente transformación afín P.

Las coordenadas XY deben pasarse como sendas matrices 3x2 de la forma $\begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{pmatrix}$

La matriz de transformación P de salida deber ser del tipo: $\begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}$

Partir del "template" get_P3.m para construir vuestra función:

```
function P=get_P3(xy,xyp)

x = xy(:,1); y =xy(:,2);
xp = xyp(:,1); yp=xyp(:,2);  % Extraigo datos origen (x,y) y destino (xp,yp)

P = [1 0 0; 0 1 0 ; 0 0 1];  % Ahora no hace nada (P = identidad)

return
```

Usar la información vista en clase para plantear y resolver los coeficientes (a,b,c) + (d,e,f) de la transformación afín. Organizarlos en una matriz P como se indica.

Para probar vuestra función llamarla con xy = 200 375 , xyp = 87 445
400 125 319 69
600 375 600 375

Debéis obtener la matriz P = 1.2825 0.0980 -206.2500
-0.1750 1.3640 -31.5000
0 0 1.0000

Adjuntar código de vuestra función

Ejercicio 2: La función interp2 de MATLAB recibe una matriz 2D (imagen) y unas coordenadas (posiblemente no enteras) y devuelve el valor interpolado de la imagen dada en esas coordenadas, usando diversos tipos de interpolación.

im2 = interp2(im,Xi,Yi,tipo)

La función interp2 asume que las coordenadas de la imagen original im son y=[1:N] y x=[1:M], con N y M las dimensiones (alto x ancho) de la imagen.

Los parámetros (Xi,Yi) son las coordenadas donde queremos interpolar. Pueden ser un punto, en cuyo caso el resultado es un único valor. También pueden ser vectores o matrices, en cuyo caso el resultado es un vector o matriz con todos los valores

interpolados. Si algunos de los valores de Xi o Yi caen fuera de la imagen de partida (y por lo tanto no podemos interpolar su valor) la función devuelve un NaN.

Vamos a escribir una función que reciba una imagen + matriz de transformación P y devuelva la imagen deformada de acuerdo a la transformación P especificada.

```
function im = interpola(im,P)
 [N,M,s]=size(im);
 im=double(im);  % Ahora deja las cosas como están
  ...
return
```

Para ello:

1) Crear unas matrices Xi,Yi del mismo tamaño N x M que la imagen dada.

2) Convertir a double la imagen im para poder hacer cuentas con ella.

3) Hacer un bucle barriendo las coordenadas de la imagen destino k=1:N,j=1:M. En cada paso del bucle:

   - Aplicar la **transformación P** a las coordenadas x=j, y=k. Para ello crear un vector 3x1 = (x, y, 1) usando coordenadas homogéneas.
   - Una vez aplicada P dividir las dos primeras componentes del vector obtenido por la tercera componente, para obtener así las coordenadas (X,Y) transformadas.
   - Guardar las coordenadas así obtenidas en las correspondientes casillas (k,j) de las matrices Xi,Yi.

4) Una vez terminado el bucle y rellenas las matrices Xi, Yi con las coordenadas a interpolar solo falta llamar a interp2 pasándole como parámetros im, Xi e Yi. Usar como tipo de interpolación 'bilinear'. Notad que si la imagen es en color tendréis que interpolar cada plano de color im(:,:,col) por separado porque interp2 solo trabaja con matrices 2D.

5) Guardar la imagen interpolada (salida de interp2) machacando la imagen de partida.

El resultado será la imagen deformada de acuerdo con la transformación P. Probar vuestro programa con la imagen de 'foto.jpg' y la transformación proyectiva P dada por la matriz:

```
P =    0.2997   -0.4120  122.4903
       0.0094   -0.0688   71.8727
       0.0000   -0.0020    1.0000
```

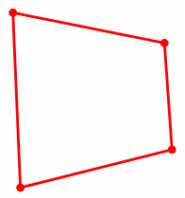
Tras visualizar la imagen: image(uint8(im)) debéis obtener algo similar a la imagen adjunta.

Ejercicio 3: Cuando os tengáis listas las funciones get_P3 + interpola podréis usar la aplicación demoP3 que por debajo llama a vuestras funciones. En la demo podéis escoger (arrastrando los vértices con el ratón) un triángulo origen y/o destino. A partir de esos datos el programa usa vuestra rutina get_P3 para hallar la matriz P de la transformación afín. Luego deforma la imagen con vuestra rutina interpola.

Mover los vértices de los triángulos de origen y/o destino y comprobad como tras la deformación los vértices del triángulo origen van siempre a parar a los vértices del triángulo destino. Adjuntar una captura del resultado.

¿Cómo tenéis que alterar los triángulos para girar 45º y hacer un zoom de la 2ª imagen sin "deformarla"? Hacerlo y adjuntar una captura del resultado.

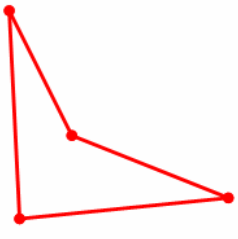
La aplicación demoP4 es muy similar pero en vez de triángulos trabajaremos con cuadriláteros y por lo tanto con matrices P proyectivas (homografías). A partir de los vértices de origen y destino, una rutina similar a la que habéis escrito vosotros (get_P4) obtiene la matriz proyectiva P que convierte un cuadrilátero en otro. Para deformar la imagen la demo vuelve a usar vuestra rutina interpola().



De nuevo el programa deforma la imagen de forma que el cuadrilátero de entrada se "mapea" dentro del cuadrilátero destino. Modificar el cuadrilátero destino de forma que se parezca al ejemplo de la izquierda y adjuntar el resultado.

Esto puede usarse para introducir una foto dentro de otra, por ejemplo colocar una foto vuestra en un panel de publicidad que aparezca en una segunda foto. Si se repite el proceso recursivamente podemos obtener imágenes como esta:

Deformar ahora el cuadrilátero destino de forma que quede como el de la figura adjunta. Adjuntar una captura del resultado.



Formalmente la figura que habéis creado es no-convexa (ya que es posible encontrar 2 puntos dentro de la figura cuya línea de unión se **salga** de la figura).

¿Es posible conseguir que un triángulo sea no convexo? Esta es una de las razones por las que cuando queremos p.e. dividir una superficie en una malla de puntos se prefiere una división por triángulos en vez de otro tipo de polígonos.

Proyecto de morphing (entrega por parejas)

Reproducir el proceso explicado en clase sobre cómo deformar 2 imágenes para así poder superponerlas antes de hallar su media.

Datos:

- Dos imágenes en color del mismo tamaño. El tamaño de las imágenes (320x270) es pequeño para que no tarden mucho los cálculos. Las imágenes son 'ant.jpg' y 'willis.jpg'. Cargar ambas imágenes en MATLAB.

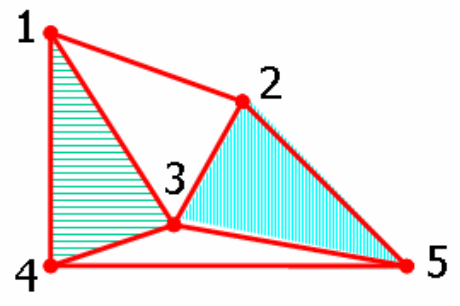
- Un fichero 'xy_control.mat' con las coordenadas x,y de varios (en este caso NP=30) puntos de control comunes en ambas fotos. Los datos (tras un load) se verán como 4 vectores columna NP x 1 llamados x1, y1, x2, y2. (x1,y1) son las coordenadas de los puntos de control en la 1ª imagen (ant) y (x2,y2) las coordenadas sobre la segunda imagen (willis)

Vuestro script debe:

* **Promediar las coordenadas** de los puntos en ambas imágenes, (x1,x2) e (y1,y2), para obtener los puntos medios xm, ym.

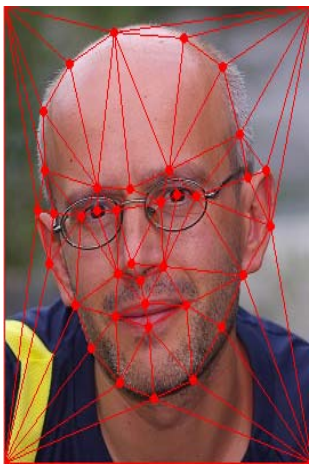
* **Hallar una triangulación** que recubra los puntos dados a base de triángulos. Para ello usaremos la función de MATLAB `tri=delaunay(xm,ym);`

Esta función devuelve una matriz NT x 3 donde NT es el número de triángulos que forman la triangulación. Cada fila de tri son los tres índices de los vértices de ese triángulo (referidos a la lista de datos originales xm e ym). Por ejemplo, para los 5 vértices del dibujo adjunto, la triangulación mostrada podría codificarse como:




```
tri = [ 1 2 3      (Vert 1, 2 ,3 = triángulo verde)
        2 3 5      (Vert 2, 3, 5 = triángulo azul)
        3 4 5      ...
        1 3 4 ]    ...
```

NT = 4 filas =4 triángulos



* Una vez que tenemos la triangulación la visualizaremos con el comando triplot(). Hacer primero un image() para ver la primera imagen seguido de un hold on y una llamada a triplot(tri,x1,y1). Deberíais ver algo como la figura adjunta.

Repetirlo para la 2ª imagen y adjuntar vuestro resultado.

*** Calcular las matrices de transformación** que convierten las coordenadas de la imagen destino (xm,ym) en las de la imagen 1 (x1,y1) PARA TODOS LOS TRIÁNGULOS HALLADOS.

Para ello hacemos un bucle desde k = 1 a NT (número de triángulos). En cada paso:

1) Determinamos los índices de los vértices asociados con el k-ésimo triángulo (idx = k-ésima fila de tri).

2) Usando idx extraemos de xm, ym las coordenadas de los tres vértices de origen Xorg,Yorg. Igualmente de x1,y1 sacamos las coordenadas de los tres vértices de destino Xdest, Ydest.

3) Como se trata de transformar un triángulo en otro podemos usar la función getP3() que ya tenemos escrita. Basta juntar las coordenadas de origen [Xorg Yorg] en una matriz 3x2, hacer lo mismo con las de destino [Xdest Ydest] y usarlas como argumento de getP3().

4) Ir guardando las matriz resultante en una array de celdas P1{k} = getP3();

Repetir para calcular la colección de matrices P2 que transforman los triángulos de la 2ª imagen a la imagen central. Podéis aprovechar el mismo bucle. Tras calcular P1{k}, extraer las nuevas coordenadas de destino de (x2,y2) y repetir la llamada a getP3(). Las coordenadas de origen son las mismas en ambos casos. Guardar la nueva matriz en un nuevo array de celdas P2{k}.

*** Escribir una rutina que deforme las imágenes** de acuerdo a P1 (si se trata de la 1ª imagen) o P2 (para la segunda). Esto es algo muy similar a la rutina del 2º ejercicio. La diferencia es que ahora P1 no es una única transformación, sino muchas, una para cada triángulo (lo mismo sucede con P2).

¿Cuál de las transformaciones usar? Dado un cierto punto debemos usar la transformación correspondiente al triángulo que contenga al punto en cuestión. Si un cierto punto pertenece al triángulo 17 usaríamos P1{17} si estamos deformando la 1ª imagen y P2{17} si queremos deformar la 2ª imagen.

La rutina determina_triang (ya suministrada, no hay que codificarla) se encarga de averiguar en qué triángulo cae cada píxel. La rutina se llama como:

```
clas=determina_triang(tri,[xm ym],N,M);
```

y devuelve una matriz 2D del mismo tamaño (N=alto x M=ancho) de las imágenes a fundir. El valor (k,j) del argumento de salida (clas(k,j) en este caso) nos indica el índice del triángulo (1,2,3…) en el que cae el píxel (k,j) de la imagen.

Con la información obtenida con determina_triang() ya podemos escribir la rutina pedida, de acuerdo al siguiente template:

```
function im=deforma_img(im,P,clas)
% im = imagen a deformar (im1 o im2)
% P{} = familia de matrices afines a utilizar (P1 o P2)
% clas = información sobre triángulo al que pertenece cada píxel, que
% usaré para determinar que P{k} usar en la transformación coordenadas
...
return
```

La función es casi idéntica a la función interpola() que ya tenéis escrita. La única diferencia es que el parámetro P en lugar de ser una única matriz, es una colección de matrices una para cada triángulo. Además la función recibe la matriz clas con la "clasificación" de todos los píxeles según el triángulo al que pertenezcan.

Copiar el código de vuestra función interpola(), ya que solo hace falta modificarlo ligeramente, dentro del bucle donde rellenábamos Xi e Yi.

- En el código original se barrían en un bucle todos los píxeles desde k=1 a N, y desde j=1 a M. Luego se aplicaba la matriz P (siempre la misma) a la coordenada correspondiente (x=j, y=k).

- Ahora, antes de aplicar la transformación debemos saber cuál de ellas aplicar. Para ello hacemos c = clas(k,j) y sabemos que el punto cae en el triángulo número c. Por lo tanto la matriz a aplicar para esa coordenada será P{c}.

Con ese único cambio (determinar el triángulo adecuado y usar la transformación correspondiente) la rutina deforma_img() está completa.

*** Usar la rutina deforma_img** para deformar ambas imágenes a su posición media. Adjuntar las imágenes resultantes (acordaros de pasar a uint8 antes de intentar visualizarlas).

*** Promediar ambas imágenes** para obtener la imagen final.
Adjuntar vuestro resultado.

Adjuntar código completo usado en el proyecto.

Extras: repetir el proceso pero haciéndolo con una foto vuestra y de vuestro compañero (o quien sea).

Las imágenes deben ser del mismo tamaño, que no debería ser muy grande. El código debería funcionar para cualquier tamaño, pero al no estar optimizado puede ser lento si las imágenes son grandes. Con IrfanView (ctrl-R) o vuestro software favorito podéis cambiar el tamaño de las imágenes antes del proceso.

Obviamente cuanto más parecidas sean la imágenes de partida mejor saldrá la mezcla: procurad que la escala y pose (orientación) de las caras en las fotos sea parecida, evitad también que una imagen sea muy clara y la otra muy oscura, etc.

El código ya lo tenéis, lo único que os falta son los puntos de control.

Podéis usar la aplicación get_puntos2 para seleccionar los puntos en ambas fotos. Recordad que tenéis que marcar los MISMOS PUNTOS y en el MISMO ORDEN en ambas imágenes.

El programa os pide que selecciones 2 jpg's, los muestra en una figura y va recogiendo vuestras pulsaciones del ratón (botón izquierdo) en una u otra imagen.

Lo más sencillo es que vayáis pinchando de forma alternada en una y otra imagen.

El programa sigue acumulando puntos hasta que pinchéis en el botón derecho. Al terminar guarda las coordenadas de los puntos de control (x1,y1,x2,y2) en el formato que usa vuestro programa de morphing. El nombre del fichero creado es:

xy_fich1_fich2.mat

Una vez que dispongáis del archivo de puntos de control + las nuevas imágenes a usar solo es cuestión de volver a correr vuestro código cambiando el nombre de los archivos de datos.

Adjuntar vuestros resultados para vuestras propias imágenes:

- Ambas imágenes con la triangulación superpuesta.
- Ambas imágenes deformadas a su posición "media"
- Promedio de ambas imágenes.