1. INTERPOLACIÓN en Transformaciones Geométricas ($R^2 \rightarrow R^2$)

En esta práctica EVITAR USAR Edit/Copy Figure para adjuntar figuras, ya que con las fotos el fichero final podría exceder el tamaño máximo permitido en la entrega. Es mejor hacer una captura de la ventana activa (Ctrl+Alt+ImprPant) y copiarla al .doc con Ctrl V.

Para leer una imagen usad el comando `im=imread('xxxx.jpg');` y para verla por pantalla `imshow(im);` Si `imshow()` no os funciona podéis usar `image(im);`

Introducción/Planteamiento

Consideremos el problema de deformar una imagen im(x,y). Primero definimos una transformación de coordenadas dada por **u(x,y)=f(x,y), v(x,y)=g(x,y).** Luego se crea una nueva imagen im'=im(u,v), donde los valores de la imagen no cambian, pero lo que antes aparecía en el pixel (x,y) ahora aparece en (u,v). Es como si la imagen estuviera impresa sobre una superficie elástica y deformaramos su soporte (sus coordenadas).

Una transformación usada en estas aplicaciones es la afín, que tiene 6 parámetros (a, b, c, d, e, f) libres:

$$u = ax + by + c$$
$$v = dx + ey + f$$

o, expresado matricialmente

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = P \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad (1)$$

Nuestro objetivo es determinar la transformación afín P especificando unas coordenadas de origen $(x,y)_k$ y de destino $(u,v)_k$. Se trata hallar (a, b,…,f) tal que al aplicar (1) a las coordenadas de origen (1ª columna) nos lleve a las coordenadas de destino (2ª columna).

Origen	Destino
(x1,y1)	(u1,v1)
(x2,y2)	(u2,v2)
(x3,y3)	(u3,v3)

Este es un problema típico de interpolación: hallar una función de un cierto tipo que aplicada a unos t_k (1ª columna de la tabla de interpolación) nos de los valores f_k (2ª columna). En vez de números $t_k \rightarrow f_k$ ahora trabajamos con parejas $(x_k,y_k) \rightarrow (u_k,v_k)$, esto es, en vez de funciones R → R estamos interpolando funciones $R^2 \rightarrow R^2$.

Determinar 6 coeficientes implicará tener 6 condiciones o ecuaciones. Por cada pareja de puntos (x,y) → (u,v) podemos plantear 2 ecuaciones:

$$u = ax + by + c$$
$$v = dx + ey + f$$

Por lo tanto se necesitan 3 parejas de coordenadas para obtener las 6 ecuaciones necesarias. O lo que es lo mismo, con tres coordenadas de origen y tres de destino queda definida una transformación afín. Como tres coordenadas definen los vértices de un triángulo, estamos hallando la transformación que convierte un triángulo de vértices (x1,y1),(x2,y2),(x3,y3) en otro de vértices (u1,v1),(u2,v2),(u3,v3).

La primera tarea es completar la función **`function P=get_afin(x,y,u,v)`** que recibe dos vectores (3x1) con las coordenadas (x,y) de origen y otros dos con las de destino (u,v). La función calculará los coeficientes (a,b,c+d,e,f) de la transformación afín y los devolverá en una matriz P (2x3) como la mostrada en la ecuación (1).

Para hallar los coeficientes de esta transformación debéis plantear las 6 ecuaciones y resolver el consiguiente sistema lineal. En este caso se puede separar el problema en dos sistemas lineales 3x3 independientes:

$$\begin{pmatrix} \\ \bar{v}_1 \\ \\ \end{pmatrix} = \begin{pmatrix} \\ H_1 \\ \\ \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} \qquad \begin{pmatrix} \\ \bar{v}_2 \\ \\ \end{pmatrix} = \begin{pmatrix} \\ H_2 \\ \\ \end{pmatrix} \begin{pmatrix} d \\ e \\ f \end{pmatrix}$$

Plantear las 6 ecuaciones a cumplir y separarlas en los 2 sistemas independientes, indicando cuáles serían los vectores v1, v2 y las matrices H1 y H2. Adjuntar una foto o captura con vuestra deducción.

Dentro de get_afin, construir las matrices H y los vectores v a partir de los datos de entrada. Procurar "apilar" vectores (comando [ ]) para evitar tener que acceder a las componentes individuales de los datos. Resolver los sistemas usando MATLAB y hallar los coeficientes (a, b, c) + (d, e, f) de la transformación. Construir con ellos la matriz P a devolver:

$$P = \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix}$$

Si probáis vuestra función con los puntos **`x=[200 400 600]'; y=[375 125 375]';`**
`u=[ 87 319 600]'; v=[445  69 375]';`

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

Adjuntad código de get_afin.m.

Volcad la matriz Q que obtenemos al intercambiar las coordenadas origen por las de destino en el ejemplo anterior.

Formar una matriz PP(3x3) añadiendo una tercera fila con [0 0 1] a la matriz P. Invertir dicha matriz para obtener la matriz QQ (cuya 3ª fila será también [0 0 1]). Volcad la matriz QQ obtenida. ¿A que se corresponden las 2 primeras filas de QQ?

Las matrices P y Q corresponderían a transformaciones afines inversas: usaremos P para pasar de coordenadas de origen (x,y) a coordenadas destino (u,v) y Q para deshacer el cambio recuperando (x,y) a partir de (u,v).

Deformación de la imagen

Escribiremos una función que aplique la transformación dada por la matriz P a una imagen en color (im), devolviendo la imagen deformada:

```
function im2 = aplica_P(im,P)
```

Al aplicar una deformación habrá que interpolar los valores de la imagen destino porque las coordenadas transformadas no caerán justo en los píxeles de la imagen original. Para hacer esta interpolación en MATLAB usaremos: `interp2(im,X,Y,tipo)`.

Esta función recibe una matriz 2D (im) y unas coordenadas (X,Y), posiblemente no enteras, devolviendo el valor interpolado en esas coordenadas. Por defecto interp2() asume que las coordenadas de la imagen original im van desde 1 hasta N (alto de imagen) para las Y's y desde 1 hasta M (ancho imagen) para las X's. El último parámetro es el tipo de interpolación usada (nosotros usaremos 'bicubic').

Si las coordenadas (X,Y) donde queremos interpolar son un único valor el resultado es el valor de im interpolado en ese punto. (X,Y) también pueden ser matrices, en cuyo caso el resultado es una matriz del mismo tamaño con los valores interpolados para todas las coordenadas dadas. A los valores de X,Y que caen fuera de la imagen inicial (por ejemplo si X<1 o X>M) la función interp2 les asigna un valor NaN.

Escribid el código de aplica_P siguiendo estos pasos:

1) Reescalar la imagen im a valores entre 0 y 1 con im=double(im)/255.

2) Determinar el alto (N), ancho (M) y los planos de color (NP) de la imagen con **`[N,M,NP]=size(im)`**. Cread dos matrices X e Y (2D) de tamaño N x M.

3) Calcular en Q la **transformación inversa de P** (de tamaño 3x2) tal como se explicó en el apartado anterior. Vamos a usar la transformación inversa Q porque se va a aplicar al espacio de llegada para llegar al de partida (x,y) que es donde tenemos los datos de la imagen.

4) Hacer un doble bucle desde v=1 a N (alto) y desde u=1 a M (ancho) de forma que pasemos por todas las coordenadas de la imagen destino. En cada paso:

   - Crear un vector columna uv (de tamaño 3x1) con componentes u, v y 1.
   - Aplicad (multiplicando) **la transformación inversa Q** al vector uv.
   - Guardar las coordenadas x resultante en la casilla correspondiente (v,u) de la matriz X. Repetir con la coordenada y (guardarla en matriz Y).

5) Una vez terminado el bucle y rellenas las matrices X, Y con las coordenadas a interpolar basta llamar a interp2 con los parámetros **`(im,X,Y,'bicubic')`**.
   Como interp2() solo trabaja con matrices 2D, tendréis que hacer un bucle desde k=1 a NP interpolando cada plano de color im(:,:,k) por separado, guardando los sucesivos resultados en im2(:,:,k).

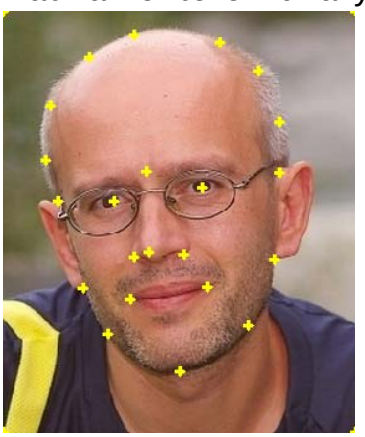
Probad la función aplicando la transformación `P =[ 0.8 -0.9 200`
`-0.2 1.0 20];`
a la imagen del fichero 'ant.jpg'. Cargarla con **`im=imread('ant.jpg')`**, aplicadle la transformación P y visualizar el resultado con **`imshow(im)`** o **`image(im)`.**
Adjuntad código de vuestra función aplica_P.m y la imagen resultante.

Morphing

En este apartado vamos a reproducir el proceso de *morphing* que vimos en clase al introducir el tema de interpolación. Lo haremos en varias etapas.

1) Marcar los puntos de control

Los datos de partida son dos imágenes en color ('willis.jpg' y 'ant.jpg') de pequeño tamaño (para hacer más rápidos) los cálculos a realizar. Lo primero es marcar una serie de puntos de control comunes con la aplicación marcar_puntos. El programa carga las dos imágenes, las muestra en una figura y recoge vuestras pulsaciones del ratón (botón izquierdo) en una u otra imagen. DEBÉIS MARCAR LOS PUNTOS EN EL MISMO ORDEN en las dos imágenes. Podéis pinchar alternativamente en una y otra imagen o marcar varios puntos en una de ella y luego marcar los mismos puntos en la otra. No intentéis marcar 20 puntos en una de ellas y luego pasar a la otra, ya que no seréis capaces de recordad su orden. Marcad del orden de 20 o 25 puntos para cada imagen (ojos, nariz, orejas, contorno de la cara, etc.) como se muestra en la figura adjunta. El programa acumula los puntos hasta que pinchéis en el botón derecho. Entonces añade automáticamente las coordenadas de las 4 esquinas a vuestros puntos y los guarda en el fichero puntos.mat.



Tras terminar, si hacéis **`>>load puntos`**, veréis las coordenadas de los NP puntos de control escogidos como cuatro vectores columna (x1, y1, x2, y2) de tamaño NP x 1. En (x1,y1) están las coordenadas de los puntos de control en la 1ª imagen (willis) y en (x2,y2) las coordenadas sobre la segunda imagen (ant).Cargad ambas imágenes y sobre cada una de ellas superponer sus correspondientes puntos de control con las opciones **`plot(...,...,'r+','MarkerSize',6,'LineWidth',2).`**

Adjuntad las figuras resultantes, indicando el número NP de puntos de control.

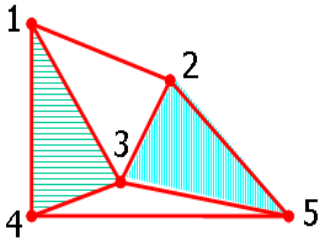
2) Dividir las imágenes en una malla triángular

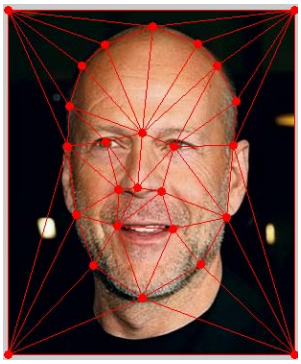
Calculad ahora la media de los puntos de control en ambas imágenes u=(x1+x2)/2;
v=(y1+y2)/2;

Las posiciones (u,v) son los puntos medios a donde queremos deformar tanto los puntos de la 1ª imagen (x1,y1) como los de la segunda (x2,y2).

Lo primero es hallar a partir de (u,v) una malla triangular que cubra las fotos, y cuyos vértices sean esos puntos. Con MATLAB podemos hacer **`T=delaunay(u,v)`** que implementa el algoritmo de triangulación de Delaunay. Recibe la lista de puntos en dos vectores (u,v) y devuelve una matriz T de tamaño NT x 3 que describe los NT triángulos de la triangulación. Cada fila de T son los tres índices de los vértices del correspondiente triángulo (referidos a la lista de vértices dada en u,v). Por ejemplo, para los 5 vértices del dibujo adjunto, los cuatro triángulos de la triangulación mostrada se codificarían como:

```
T = [ 1  3  4    % vertices 1, 3 ,4 = triángulo rallado
      2  3  5    % vertices 2, 3 ,5 = triángulo azul
      3  4  5    ...
      1  2  3]   ...
```





Una vez calculada una triangulación se puede visualizar usando triplot(T,x,y) y superponerla en la imagen de dónde sacamos los puntos de control. Usad para cada imagen sus propios puntos de control (x1,y1) y (x2,y2), pero recordad que la triangulación T es única (la obtenida a partir de los puntos medios u, v). Adjuntad vuestro resultado para ambas imágenes (que debe ser similar al adjunto). ¿Cuántos triángulos obtenéis en vuestra triangulación? Llamad NT a dicho número.

3) Calcular el conjunto de matrices de transformación P1 y P2

A continuación calcularemos las matrices P1 que convierten las coordenadas (x1,y1) en la 1ª imagen en las correspondientes coordenadas (u,v) de la imagen destino PARA TODOS LOS TRIÁNGULOS de la triangulación. Al tener NT triángulos habrá NT matrices de transformación (de tamaño 2x3) que guardaremos en un array de celdas creado con **`P1=cell(1,NT)`**. Un array de celdas es como una tabla normal pero en sus casillas, en lugar de números, podemos guardar cualquier tipo de objeto (en nuestro caso matrices 3x2). La única diferencia es que para indexarlo hay que usar llaves, P1{k}, en vez de paréntesis, P1(k), como se haría con arrays. Para calcular estas matrices haremos un bucle en k desde 1 a NT (número de triángulos) y en cada paso:

1) Extraeremos la k-ésima fila de la matriz T (triangulación) que contiene los índices de los 3 vértices del k-ésimo triángulo.

2) Con esos índices extraemos de x1, y1 las coordenadas de los tres vértices origen (X,Y). De la misma forma sacamos de u,v las coordenadas de los tres vértices destino (U,V).

3) Para transformar el triángulo dado por las coordenadas (X,Y) en otro de coordenadas (U,V), usamos la función get_afin(X,Y,U,V). La matriz 2x3 resultante la guardaremos en la k-ésima celda P1{k}.

Repetir para calcular las matrices P2 que transforman los triángulos de la 2ª imagen a la imagen central. Podéis aprovechar el mismo bucle: tras calcular P1, extraer (ahora de x2, y2) las nuevas coordenadas de origen (X,Y). Las coordenadas de destino (U,V) no cambian (son las mismas en ambos casos). Repetid la llamada a get_afin y guardar la nueva matriz en P2{k}. Adjuntad código de vuestro bucle.

4) Escribir una rutina que deforme una imagen aplicando la transformación a trozos, especificada por un conjunto de transformaciones P (array de celdas). Ahora necesitamos saber en qué triángulo cae cada punto de la imagen para saber qué transformación P{k} aplicarle. La rutina **determina_triang** (suministrada) calcula para una triangulación dada en qué triángulo cae cada píxel de la imagen destino. La rutina se llama (una sola vez) como:

```
zona=determina_triang(T,u,v,N,M);
```

Recibe los datos de la triangulación T + coordenadas (u,v) + el tamaño (N,M) de las imágenes y devuelve una matriz zona del mismo tamaño (N x M) que las imágenes. El valor de zona(i, j) nos da el índice del triángulo (1,2,3…,NT) al que pertenece el píxel (i, j) de la imagen destino. Con esta información ya podemos escribir la rutina pedida, de acuerdo al siguiente template:

```
function im=warp_img(im,P,zona)
% im = imagen a deformar (im), P = familia de matrices afines a utilizar
% zona = información sobre el triángulo al que pertenece cada píxel.
 ...
return
```

La función es muy parecida a aplica_P(). Las diferencias son:

- La función recibe un parámetro adicional (zona) con la "clasificación" de todos los píxeles de la imagen según el triángulo al que pertenezcan.
- El 2º parámetro P, en vez de una única matriz, es una colección de matrices (en un array de celdas), una por cada triángulo.

Para el código de la función podéis copiar dentro de la nueva función el viejo código de aplica_P() para ir modificándolo. Basta hacer un par de cambios:

1. El cálculo de la transformación inversa de P (Q) se sustituye por un bucle para calcular todas las inversas Q{k} para cada P{k}.
2. Dentro del bucle donde se rellenaban las matrices X, Y: antes de aplicar la inversa de la transformación al vector de coordenadas uv hay que determinar cuál de las transformaciones Q hay que usar. Podéis averiguar el triángulo al que pertenece el píxel con z=zona(v,u) y usar la transformación Q{z}.

Adjuntad el código de vuestra función. Usarla luego para deformar ambas imágenes a la posición media. En un caso tendréis que usar las transformaciones guardadas en P1 y en el otro las de P2. Adjuntar las dos imágenes resultantes.

Finalmente, promediar ambas imágenes. Adjuntad imagen con el resultado final.

El código anterior puede generalizarse para calcular la mezcla no sólo para el punto medio (t=0.5) sino para cualquier valor de t entre 0 y 1. Para ello recalculamos los puntos destino en función del valor de t como:

$$u=(1-t)\cdot x1+t\cdot x2;$$
$$v=(1-t)\cdot y1+t\cdot y2;$$

De esta forma los puntos de la malla destino (u,v) se moverán de forma gradual desde sus posiciones iniciales en la 1ª imagen (u=x1,v=y1, para t=0), hasta sus posiciones finales en la 2ª imagen (u=x2, v=y2, para t=1).

Para cada t, al cambiar (u,v), hay que recalcularse las transformaciones P1 y P2. La triangulación T no hace falta volver a calcularla (se usa siempre la encontrada antes para los puntos medios). Luego usaremos P1 y P2 para deformar las dos imágenes con warp_im() y la imagen final será el promedio de las 2 imágenes deformadas, pero ahora ponderadas según el valor de t:

```
mezcla=(1-t)·im1_def + t·im2_def
```

Adjuntad las imágenes mezcla obtenidas para los valores de t=0.3 y t=0.7.

5) Creación de un GIF ilustrando el proceso

Repetir el proceso pero ahora con una foto vuestra y de vuestro compañero de prácticas. Ambas fotos deben ser del mismo tamaño. Aunque el código funciona para cualquier tamaño, al no estar optimizado puede ser lento para imágenes muy grandes. Obviamente cuanto más parecidas sean las imágenes de partida mejor saldrá la mezcla: procurad que la escala y pose (orientación) de las caras en las fotos sea parecida, evitad que una imagen sea muy clara y otra muy oscura, etc. Lo mejor es haceros las fotos con el mismo móvil en el mismo lugar para que el fondo sea similar. Es preferible un fondo con pocos detalles (cielo, pared).

Adjuntad las 2 imágenes originales y las mezclas obtenidas para t=0.4 y t=0.6

Una vez que sabemos hacer la mezcla para cualquier t intermedio, cread un vector t con: `N=15; x=(-N:N)/N; t=0.5+tan(x)/pi; t=[t fliplr(t)];`

Este vector toma valores desde t 0 a 1 y de vuelta a 0, de forma que pasamos desde la 1ª imagen a la segunda y volvemos de nuevo a la primera.

Haced un bucle calculando la imagen mezcla para cada t(k) y guardarla de forma automática en un fichero (morph01.jpg, morph02.jpg, ...) con:

```
fich=sprintf('morph%02d.jpg',k); imwrite(mezcla,fich,'Quality',95);
```

Finalmente subid las imágenes a un sitio como http://gifmaker.me/ o similar para convertirlas a un GIF animado. Subid el GIF resultante a la entrega Moodle abierta para ello.

2. AJUSTE en Transformaciones Geométricas ($R^2 \rightarrow R^2$)

Introducción/Planteamiento

Vamos a aplicar el ajuste de datos a un problema cuyos fundamentos son muy similares al que hemos visto en la 1ª parte. Trabajaremos con un mapa topográfico en papel (zona de la Sierra de Gredos) escaneado y guardado como una imagen ('mapa.jpg'). El objetivo es superponer sobre este mapa digital un recorrido tomado con un GPS. El problema es que no conocemos la relación (calibración) entre los píxeles de la imagen y las correspondientes coordenadas geográficas.

Podríamos atacar este problema averiguando datos como la escala original del mapa (p.e. 1:25000), la densidad de puntos (DPIs) usados al escanear, el posible giro del mapa en el scanner, etc. En vez de tratar de averiguar o medir esos parámetros (lo que podría no ser tarea fácil) vamos a hallarlos como los parámetros de un ajuste de datos. Para el ajuste usaremos una serie de puntos de control, puntos para los cuales conocemos tanto su posición en píxeles sobre la imagen (X,Y) como sus coordenadas geográficas (E,N):

	Pixel X	**Píxel Y**	**E (UTM) mt.**	**N (UTM) mt.**
Cabeza Nevada	380	159	305090	4460829
Pto. Candeleda	2775	1706	309535	4457780
Plataforma (K12)	3127	48	310288	4460893
Refugio del Rey	2182	1457	308485	4458288
Peña Rayo	1416	477	307032	4460173
Almanzor	212	1764	304690	4457805
Casquerazo	855	2045	305890	4457240
La Galana	125	1163	304556	4458946

Cargar los datos de la tabla anterior con **`>>load ptos_control`**. Visualizar el mapa y superponer sobre él los puntos de control usando círculos azules. Haced zoom sobre alguno de los puntos de control y adjuntad la imagen resultante.

En un problema de ajuste es muy importante acertar con la relación existente entre nuestros datos (en este caso píxeles vs coordenadas geográficas). Una relación que cubre los problemas mencionados antes es una transformación de similaridad que consiste en un giro θ, un cambio de escala s y unos desplazamientos oE, oN:

$$\begin{pmatrix} E \\ N \end{pmatrix} = s \cdot \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} X \\ Y \end{pmatrix} + \begin{pmatrix} oE \\ oN \end{pmatrix} \quad \Rightarrow \quad \begin{aligned} E &= s\cos\theta \cdot X - s\sin\theta \cdot Y + oE \\ N &= s\sin\theta \cdot X + s\cos\theta \cdot Y + oN \end{aligned}$$

donde **(E,N)** son las coordenadas geográficas y **(X,Y)** píxeles sobre la imagen.

Las ecuaciones anteriores deben modificarse ligeramente, ya que mientras que los ejes E y X van en la misma dirección (incrementándose hacia la derecha), los ejes N e Y van en sentidos contrarios. Por convención, el píxel (1,1) de una imagen es su

esquina superior izquierda, por lo que la coordenada Y en píxeles se incrementa al bajar, mientras que la coordenada Norte de los mapas (también arbitrariamente) se incrementa hacia arriba. Para tener en cuenta esta discrepancia basta cambiar el signo a la coordenada Y en las ecuaciones anteriores:

$$E = s\cos\theta \cdot X + s\sin\theta \cdot Y + oE$$
$$N = s\sin\theta \cdot X - s\cos\theta \cdot Y + oN$$

Este modelo no es lineal en los parámetros (s,θ), pero eso se soluciona cambiando a unos nuevos parámetros (a,b) que combinan la información de escala y giro:

$$E = a \cdot X + b \cdot Y + oE$$
$$N = b \cdot X - a \cdot Y + oN$$

o matricialmente:

$$\begin{pmatrix} E \\ N \end{pmatrix} = \begin{pmatrix} a & b & oE \\ b & -a & oN \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} = P \cdot \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$

siendo a=s·cos(θ), b=s·sin(θ).

Con los nuevos parámetros el problema es muy similar al de la primera parte de la práctica, estimar los parámetros (en este caso a, b, oE, oN) que forman la matriz de transformación P para poder así convertir de píxeles (X,Y) a coordenadas (E,N).

La diferencia es que aquí tenemos más datos de los estrictamente necesarios (es un problema de ajuste). A partir de los puntos de control disponibles con píxeles (X,Y) y coordenadas (E,N) se trata de hallar los parámetros (a, b, oE, oN) que mejor cumplen que:

$$E \approx a \cdot X + b \cdot Y + oE$$
$$N \approx b \cdot X - a \cdot Y + oN$$

Lo primero es identificar el sistema sobredeterminado (H·c=v) cuyas incógnitas son los parámetros (a,b,oE,oN) del ajuste:

$$H \cdot \bar{c} \approx \bar{v} \quad \rightarrow \quad \begin{pmatrix} & & & \\ & H & & \\ & & & \end{pmatrix} \begin{pmatrix} a \\ b \\ oE \\ oN \end{pmatrix} = \begin{pmatrix} \\ v \\ \\ \end{pmatrix} = \begin{pmatrix} E \\ - \\ N \end{pmatrix}$$

Deducir la matriz H y vector v del sistema sobredeterminado. Se recomienda poner primero las ecuaciones de las coordenadas {E} y a continuación las de las {N}.
Adjuntar scanner/foto de vuestro deducción y resultado.

Se trata ahora de completar la función **`function [P,res]=get_similar(X,Y,E,N)`**. Su formato es muy similar a **`get_afin()`** que escribimos en el 1er apartado. Al igual que antes la función recibe dos vectores columna con coordenadas 2D origen (X,Y) y otros dos con coordenadas 2D de destino (E,N) y debe calcular la matriz P del mejor ajuste entre ambas.

La diferencia es que en el caso anterior ambas coordenadas 2D eran del mismo tipo, coordenadas (píxeles) sobre una imagen. Ahora las segundas coordenadas son coordenadas geográficas (E,N), pero el problema matemático subyacente es similar.

Dentro de get_similar construid la matriz H y el vector v a partir de (X,Y) y (E,N) según vuestra deducción. Procurar usar los comandos de "apilar" vectores sin tener que acceder a las componentes individuales de los datos de entrada. Luego resolver el sistema sobredeterminado con MATLAB para hallar los coeficientes (a, b, oE, oN) que usaréis para construir la matriz P que devuelve la función:

$$P = \begin{pmatrix} a & b & oE \\ b & -a & oN \end{pmatrix}$$

Como este es un caso de ajuste, además de la matriz P queremos que la función devuelva los residuos del ajuste (res = v - H·c).

Adjuntad código de vuestra función get_similar().

Volcad la matriz P obtenida usando **`fprintf('%6.3f %6.3f %9.1f\n',P');`**

Adjuntad una gráfica de los 8 residuos en la coordenada E de los puntos de control.

Lo que está pasando es que se cometió un error al introducir los datos de uno de los puntos de control. Detectar cuál es el punto problemático, eliminarlo y volver a hacer el ajuste usando los 7 puntos restantes. Volved a adjuntad la nueva matriz P obtenida y el volcado de los nuevos residuos. Ahora los residuos deben ser bajos, del orden de un par de metros como máximo. Si veis residuos mayores es que tenéis algún error en vuestra función get_similar().

Si deseo llegar al refugio Elola (al lado de la laguna grande que aparece en el centro del mapa), ¿qué coordenadas debo introducir en mi GPS?

¿Cuál es el ángulo de giro (en grados) entre los ejes (X,Y) de la imagen y los ejes (E,N) de coordenadas?

Visualización de un track sobre el mapa.

La transformación obtenida (matriz P) nos permite pasar de píxeles a coordenadas geográficas (X,Y --> E,N). Esto puede servirnos para planear una ruta, marcando un recorrido sobre la imagen del mapa y convirtiendo las coordenadas sobre la imagen en coordenadas geográficas, que podemos enviar a nuestro GPS.

Otras veces queremos hacer lo contrario, pasar de coordenadas geográficas (E,N) a píxeles (X,Y). Una aplicación sería solapar sobre la imagen del mapa un track con las coordenadas (obtenidos con el GPS) para ver por donde hemos estado. En este caso necesitamos la transformación inversa de la anterior, que pase de coordenadas (E,N) a píxeles (X,Y). Podemos obtenerla de dos formas distintas:

1) Invertir la matriz P para obtener la matriz Q1 (tamaño 2x3) de la transformación inversa como hicimos al principio de esta práctica.

2) Alternativamente podemos aprovechar que la función get_similar() no sabe nada de píxeles ni de coordenadas geográficas. Simplemente halla la transformación que aplicada a ciertas coordenadas 2D (1er y 2º argumentos) las convierte en lo más parecido posible a otras coordenadas 2D (3er y 4º argumentos). Podemos por lo tanto llamar a get_similar cambiando el orden de sus argumentos para obtener directamente la matriz inversa Q2.

Volcad los valores de Q1 o Q2 con el formato usado antes para las matrices P.

Restad Q1 y Q2 para ver si son exactamente iguales. Adjuntad el resultado de la resta. ¿Son exactamente iguales Q1 y Q2? ¿Qué puede estar pasando?

Conocida la matriz de la transformación inversa la usaremos para superponer un track sobre el mapa:

a) Haciendo `>>load track` veréis aparecer una variable **track** (matriz 2 x N). Cada columna corresponde a una posición (E,N) guardada por el GPS en un recorrido.

b) Aplicar la transformación Q1 (o Q2) al conjunto de posiciones de la ruta para obtener los píxeles X e Y correspondientes a las coordenadas (E,N) del track.

c) Superponer sobre la imagen del mapa el recorrido con un plot de las posiciones en píxeles obtenidas antes. Usar una línea discontinua de color azul ('b:') y grueso de línea 2:

```
plot( . . . , 'b','LineWidth',2)
```

Adjuntad la imagen del mapa con la ruta superpuesta.