Automatizar la creación de un mosaico de fotos (entrega por parejas para el 6 junio)

En este LAB vamos a ir escribiendo una serie de rutinas que nos permitirán el montaje automático de mosaicos de fotos.

Ejercicio 1: (Obtención de los keypoints usando algoritmo SIFT)

El punto de partida del proceso es un conjunto de 10 imágenes jpg en color (image01 a image10) que forman un mosaico.

Se trata de aplicar sobre ellas el algoritmo SIFT (Scale-Invariant Feature Transform) para obtener un conjunto de características o puntos relevantes (keypoints). Este algoritmo no vamos a implementarlo nosotros. Usaremos un ejecutable (solo para Windows) llamado siftWin32.exe. Su uso (desde la línea de comandos) es:

siftWin32 <img.pgm >data.txt (img.pgm = imagen data.txt = datos salida)

Alternativamente podéis ejecutarlo desde MATLAB usando el comando system de MATLAB que ejecuta una orden (cadena de texto) como si se hiciera desde una línea de comandos:

```
system('siftWin32 <img.pgm >data.txt');
```

El programa siftWin32 lee la imagen (sólo funciona con imágenes B/W y en formato pgm) y calcula sus "keypoints", escribiendo la info en un fichero de texto. Como nuestras imágenes son en color y en formato jpg deberemos pasarlas antes a niveles de gris y guardarlas en formato pgm:

```
im=imread('image01.jpg'); bw=fc_rgb2gray(im); imwrite(bw,'img.pgm');
```

Tras ejecutar SiftWin32, usaremos la función suministrada parse_keyfile('data.txt') que lee los datos del fichero data.txt y los organiza en una estructura con campos:

.xy = matriz de Np x 2 con las posiciones x e y de los Np keypoints de la imagen.

.sd = matriz de Np x 2 con la info de escalas/direcciones asociadas a cada punto.

.desc = matriz de Np x 128 con la lista de 128 descriptores (independientes de escala y orientación) que describen cada uno de los Np keypoints.

Esta función puede aceptar un 2º argumento: parse_keyfile('data.txt',[2 3])

En este caso la función devuelve solamente información de aquellos "keypoints" cuya escala característica esté en el rango especificado. El objetivo es simplemente reducir el número de puntos por cada foto y reducir el trabajo para las etapas posteriores.

Se trata de:

1) Definir name='image' y Nfotos=10 (nombre y número de imágenes).
2) Reservar un array de celdas de tamaño Nfotos: s=cell(1,Nfotos)
3) Hacer un bucle barriendo las 10 imágenes. En cada paso:

- Leer la imagen original, pasarla a bw y guardarla como img.pgm. Podéis usar `fich=sprintf('%s%02d.jpg',name,k);` para crear el nombre del archivo a leer y cargar la imagen con `im=imread(fich);`

- Ejecutar siftWin32 desde MATLAB usando system().

- Ejecutar parse_keyfile('data.txt',[2 3]) y guardar el resultado en s{k}

Al terminar tendréis un array de celdas donde cada elemento s{1},…,s{10} es una estructura con la información sobre los "keypoints" detectados para las sucesivas image01, …, image10, con los campos antes indicados.

Hemos filtrado los "keypoints" con un rango de escalas entre 2 y 3, lo que nos da un rango de entre 400 a 900 puntos por imagen.

Finalmente hacer un **save keypoints s**, guardando el array s con los resultados en el fichero keypoints.mat. Para el resto de la práctica ya no es necesario usar SiftWin32.exe ni trabajar en Windows. Podemos trabajar a partir de la información guardada en s{} haciendo **load keypoints**.

Hacer un volcado con el número de los "keypoints" guardados para cada imagen.
Hacer un gráfico de las 4 primeras imágenes (usar subplot(22x)) superponiendo sobre ellas la posición de sus keypoints como círculos rojos ('ro'). Adjuntar imagen.

El número de "keypoints" detectados en esas 4 primeras imágenes son: 390, 407, 486 y 431 si hemos seleccionado un rango de escalas entre 2 y 3.

Ejercicio 2: (hallar posibles emparejamientos entre dos imágenes)

Se trata ahora de encontrar posibles parejas entre los "keypoints" de las diferentes imágenes (que serán el punto de partida de la aplicación del algoritmo RANSAC que escribiremos luego). Para ello escribir una función:

```
function [xy1,xy2]=find_matches(s1,s2)
```

que recibe una par de estructuras como las descritas antes y encuentra los posibles emparejamientos entre "keypoints". La función debe devolver dos argumentos de salida, xy1 y xy2, con las posiciones de las posibles parejas encontradas.

El algoritmo que vamos a implementar será una simple búsqueda exhaustiva:

- Inicializar xy1=[], xy2=[]

- Por cada uno de los N1 "keypoints" de s1:

  Calcular la distancia entre los descriptores del keypoint de s1 con todos los N2 "keypoints" de s2. La distancia es simplemente la norma del vector resta de los descriptores correspondientes, vistos como vectores de 128 componentes.

  Guardar la distancia mínima Dmin hallada (y el índice de s2 donde la hemos encontrado), junto con el ratio entre las distancia de la 2ª opción y la de la 1ª, ratio=(d2/Dmin).

  Si el ratio es >= 5 (el 1er candidato está "cinco veces" más cerca que el segundo) se acepta la pareja. En ese caso añadir a xy1 la posición del punto de s1 que estamos considerando y a xy2 la posición del punto "ganador" en s2 (el que verificaba la distancia mínima).

Al terminar tendremos dos listas xy1, xy2 con las posiciones de las posibles parejas entre los puntos de las estructuras s1 y s2 (procedentes de sendas imágenes).

Adjuntar vuestro código de find_matches.m

Si aplicáis la búsqueda a los datos de las imágenes 1 (390 keypoints), 2 (407) y 3 (486) debéis obtener:

- 45 posibles emparejamientos entre la 1 y la 2
- 114 posibles parejas entre la 2 y la 3.

Ejercicio 3: Reescribir vuestra función get_P3 del laboratorio anterior para que resuelva el problema para el caso de recibir más de tres pares de coordenadas. En ese caso, como solo tenemos 6 grados de libertad (6 parámetros de la matriz P) y más de 6 ecuaciones (más de tres pares de puntos) tendremos un caso de ajuste.

Si codificasteis bien get_P3 (sin asumir que iban a ser sólo tres puntos, usando H\b en vez de inv(H)*b, …) es posible que no casi no tengáis que cambiar nada.

Hacer las modificaciones oportunas para que no get_P3 funcione correctamente para N>=3 pares de datos. Adjuntar código de vuestra función.

Para probar vuestra función llamarla con xy1 = 200 375 , xy2 = 87 445
400 125 319 69
600 375 600 375
300 200 200 185

Debéis obtener la matriz P = 1.2811 0.0955 -204.5617
-0.1721 1.3690 -34.8765
0 0 1.0000

Comprobad que en este caso al aplicar P a xy1 no obtenemos exactamente xy2.

Escribir una función error_ajuste: `function err=error_ajuste(xy1,xy2,P)`
que reciba las colecciones de puntos xy1, xy2 y la matriz de ajuste P obtenida con get_P3.

La rutina debe devolver un vector con los errores cometidos por el ajuste en cada punto. Calcular las posiciones obtenidas según el ajuste, xy2_ajuste, y restarlas a las deseadas xy2. Calcular la norma de cada vector error y ese será el vector de errores a devolver.

Adjuntar código de vuestra función.

Volcar el resultado de aplicarla a los datos del ejemplo.
¿Cuál es el error máximo cometido para los puntos anteriores? ¿En qué punto?

Ejercicio 4 (algoritmo RANSAC): Escribir una función que implemente el algoritmo RANSAC (RANdon Sample And Consensus) con el siguiente template:

```
function [Nmax,T]=ransac_fun(xy1,xy2)
```

A partir de una colección de Np parejas de puntos xy1 y xy2 se trata de hallar el mejor ajuste (usando una T afín) para pasar de xy1 a xy2. Si los puntos estuvieran elegidos con cuidado podríamos usar directamente T=get_P3(xy1,xy2) para encontrar el mejor ajuste. Sin embargo, si los datos incluyen muchos falsas parejas será necesario aplicar un algoritmo robusto tipo RANSAC para descartar "outliers".

La función debe devolver el número de puntos correctos Nmax y la transformación T que proviene de ajustar solamente esos "inliers". El algoritmo es el siguiente:

1) Determinar Np = número de parejas = número de filas de xy1 o xy2.

2) Inicializar Nmax=0; select=[]; correspondientes al número de puntos correctos encontrados ("inliers") y a la posición de dichos puntos en las tablas xy1,xy2 dadas.

3) Bucle de 500 iteraciones:

- Crear 3 índices de 1 a Np aleatorios usando rand.
- Usarlos para extraer 3 pares de coordenadas de xy1 y xy2.
- Llamar a get_P3 para esos tres parejas y obtener la correspondiente T.
- Usar la función error_ajuste para calcular el error de TODOS los puntos.
- Hallar cuantos de los puntos tienen un error menor que 0.5 píxeles. En MATLAB haríamos algo así como: ok = find(err<0.5)
- Si el número de píxeles bien ajustados (length(ok)) es mayor que Nmax, actualizar Nmax con el nuevo valor y guardar esos índices (ok) en select.

Deberíamos exigir que los tres índices fuesen ser distintos pero como vamos a hacer muchas pruebas podemos esperar que la mayoría de las veces lo sean. Eso si, añadir un 'warning off' al principio de la función para evitar "warnings" cuando se repitan puntos y nos salga una matriz singular en el proceso de ajuste.

4) Una vez terminado el bucle de "pruebas" en Nmax tenemos el máximo número de puntos que caían cerca de un ajuste. Podemos ahora calcular la transformación T usando SOLO los datos correspondientes a los índices guardados en select.

Volcar (fprintf) los puntos iniciales (Np) y los aceptados como "inliers" (Nmax).

Si cargáis los datos de data_ransac veréis un conjunto de 120 parejas de puntos (xy1,xy2). Tras correr el algoritmo debéis obtener unos 65-70 "inliers" y una matriz de T que será muy parecidas a:

```
 1.0040    0.0249  355.7627
-0.0270    0.9978 -277.5551
 0.0       0.0       1.0000
```

Ejercicio 5: (determinar conectividad entre las imágenes).

Ahora se trata de usar las funciones anteriores find_matches y ransac_fun para escribir una función que analice TODOS los keypoints de TODAS las imágenes y decida que imágenes están conectadas y a través de qué transformaciones.

```
function [Q P]=find_QP(s)
```

La función recibe como entrada el array de estructuras con los "keypoints" de TODAS las imágenes y devuelve dos matrices Q y P ambas cuadradas y de tamaño el número de imágenes:

- Q será una especie de matriz de conectividad, donde Q(i,j) guarda el número de parejas (confirmadas a través del algoritmo RANSAC) entre la imagen i y la j. La matriz Q es simétrica, con Q(i,j) = Q(j,i).

- P es un array de celdas donde **P{I,J}** guarda la matriz T de ajuste entre las **coordenadas de la imagen J respecto a las de la imagen I**. En P{J,I} guardaremos la correspondiente matriz inversa.

El proceso será el siguiente

- Inicializar la matrices Q=zeros(Nfotos) y P=cell(Nfotos)

- Hacer un doble bucle desde I=1:Nfotos y desde J=i+1 hasta Nfotos

  Dentro del bucle:

1) Usar la función de matching entre s{I} y s{J} para encontrar las posiciones de los posibles emparejamientos xy1, xy2

2) Si el número de posibles pares obtenidos es mayor de 5, usar los datos de xy1, xy2 para alimentar vuestro algoritmo RANSAC.

3) Si la función ransac_fun devuelve a menos 5 parejas aceptadas, usar la salida de ransac_fun() (Nmax y T) para rellenar Q(I,J) y P{I,J}.

Recordad que P{I,J} es la transformación de imagen J a la I. Aprovechar las propiedades de P y Q para rellenar Q(J,I) y P{J,I}.

4) Para cada par de imágenes volcar por pantalla cuántos puntos han sido emparejados por find_matches() y cuántos han sido aceptados por RANSAC.

Adjuntar código de find_QP()

Volcar la matriz Q de "conectividad" obtenida.

¿Cómo podríamos detectar en MATLAB que una de las imágenes de partida no está conectada a las demás.

Ejercicio 6: Usaremos la conectividad guardada en Q y las transformaciones de P para obtener un array de celdas T{} con las transformaciones a aplicar a cada imagen para llevar cada una de ellas a un sistema de referencia COMÚN.

```
function T = ordena(Q,P)
```

Lo normal es escoger una imagen y referir TODAS las transformaciones respecto de ella. Es lo que se llama la imagen "ancla" en un mosaico. A veces la imagen ancla es elegida por el usuario, pero se puede determinar automáticamente. Un posible criterio es escoger aquella imagen con el mayor número de puntos de contacto con las otras imágenes o la que está conectada al mayor número de imágenes.

Usando el criterio de conectividad al mayor número de imágenes ¿cómo podemos determinar la imagen ancla a partir de la matriz Q? Adjuntar vuestro código y el índice (1-10) de la imagen para usar como ancla.

Ayuda Matlab: Recordad que sum(A) devuelve la suma de las columnas de una matriz A y que la función [M i]=max(x) devuelve el máximo valor (M) de un vector x y TAMBIÉN el índice (i) de la posición donde se encuentra dicho máximo.

Un posible algoritmo para obtener la lista de transformaciones T{} podría ser:

- Marcar todas las imágenes como no procesadsa. Podéis usar una variable lógica en MATLAB como done = false(1,Nfotos).

- Hallar la imagen ancla y marcarla como procesada con done(ancla)=true; Como la referencia es la propia imagen ancla, T{ancla}= identidad (3x3)

- Mientras no estén todas las imágenes colocadas:

  1) Usando indexado lógico de MATLAB extraer una submatriz de Q cuyas filas sean las de las imágenes ya colocadas (done) y cuyas columnas correspondan a las no imágenes colocadas (~done)

  2) Buscar la posición del máximo dentro de la submatriz que corresponderá a la imagen no colocada (columna del máximo) mejor conectada con una de las ya colocadas (fila del máximo). Para hallar la posición (i,j) del máximo de una matriz usar:

  `[M,index]=max(A(:)); [i,j]=ind2sub(size(A),index);`

  3) Determinar índice de nueva imagen (new) y aquella en la que se apoya (ref) referidos a la lista completa de imágenes.

  4) Construir T{new} combinando la transformación entre las coordenadas de new a las coordenadas de ref (en la matriz P) con la transformación de ref a ancla (ya calculada en T{ref} al estar la imagen ref en la lista de procesadas)

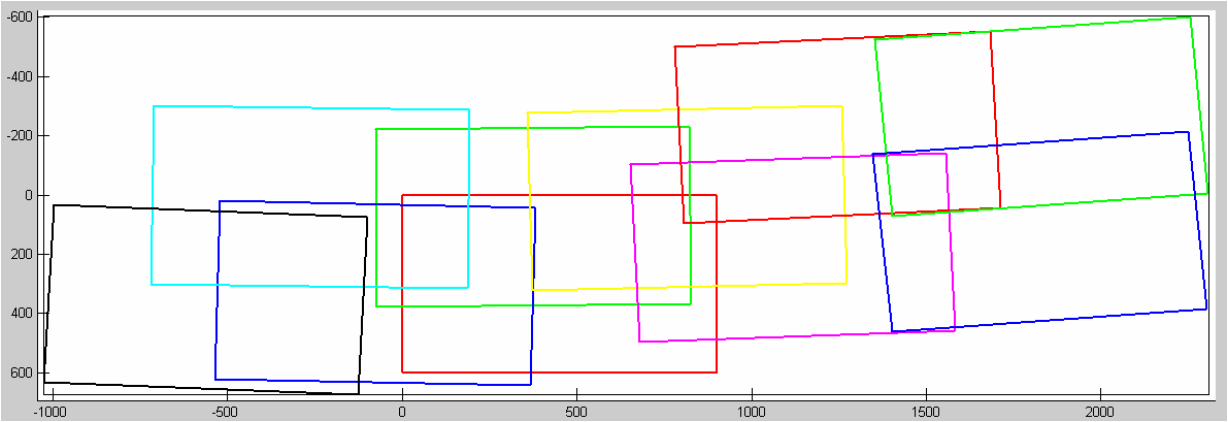
  5) Marcar la imagen new como colocada en done.

Al terminar (si todas las imágenes estaban conectadas) tendremos una lista T con las transformaciones entre todas las imágenes y la imagen ancla. Adjuntar código

El volcado siguiente muestra el progreso del algoritmo de colocación

```
Image  7 es el ANCLA
Image  3 enlaza con  7. COLOCADAS:  3  7
Image  2 enlaza con  3. COLOCADAS:  2  3  7
Image  8 enlaza con  2. COLOCADAS:  2  3  7  8
Image  9 enlaza con  8. COLOCADAS:  2  3  7  8  9
Image 10 enlaza con  9. COLOCADAS:  2  3  7  8  9 10
Image  1 enlaza con  9. COLOCADAS:  1  2  3  7  8  9 10
Image  4 enlaza con  7. COLOCADAS:  1  2  3  4  7  8  9 10
Image  5 enlaza con  4. COLOCADAS:  1  2  3  4  5  7  8  9 10
Image  6 enlaza con  5. COLOCADAS:  1  2  3  4  5  6  7  8  9 10
```

Ejercicio 7: Ilustración de la posición de las diferentes fotos en el mosaico.



Para conseguir el gráfico adjunto partiremos de las coordenadas de un rectángulo del tamaño de las imágenes:

Rx = [1 900 900 1], Ry = [1 1 600 600]

y haremos un bucle barriendo las 10 transformaciones obtenidas en el ejercicio anterior. En cada paso aplicamos la transformación T{k} a las coordenadas (Rx,Ry) para obtener las coordenadas x,y del rectángulo modificado (transladado/rotado).

Para pintar los diferentes rectángulos podéis usar la función patch de MATLAB:

```
p=patch(x,y,'w'); set(p,'FaceColor','none','EdgeColor','r','LineWidth',2);
```

Para que la imagen final quede correcta hacer un `set(gca,'Ydir','reverse');` que le da la vuelta al eje de las Y (ya que en imágenes el eje Y está invertido).

Adjuntar vuestra propia imagen obtenida mostrando el solapamiento y la posición de las imágenes en el mosaico.

El ejercicio anterior, además de mostrarnos gráficamente la disposición de las imágenes en el mosaico nos ilustra alguna pega adicional cuando tratemos de juntar las imágenes en un mosaico final.

Fijaros como hay un triángulo rojo centrado que ocupa las coordenadas originales (600x900). Corresponde a la imagen ancla cuya transformación es la identidad y por lo tanto no se mueve. Fotos a la izquierda ocuparán coordenadas X negativas y fotos por encima coordenadas Y negativas.

Esto no es bueno cuando vamos a componer la imagen final, ya que en una imagen las coordenadas deben ser siempre positivas. La solución es sencilla: sumar unos desplazamientos DX,DY suficientemente grandes a las coordenadas X,Y para asegurarnos de que todas caen en coordenadas positivas.

También debemos conocer el tamaño total del mosaico final para reservar espacio antes de ir pegando sobre ella las diferentes imágenes.

Podemos aprovechar el bucle anterior para comprobar estos dos aspectos:

- Inicializar Xmin,Xmax,Ymin,Ymax a los valores que toma la imagen ancla.

- En cada paso comparar con las coordenadas del rectángulo transformado y guardar los correspondientes valores máximos y mínimos.

- Al final podemos conocer las dimensiones del mosaico restando los valores máximos y mínimos en X (ancho) e Y (alto). Añadid por ejemplo M=10 o 20 píxeles adicionales en cada dimensión para tener un poco de "margen".

- Igualmente los desplazamientos DX,DY serían las inversas de Xmin,Ymin. Notad que si p.e. Xmin=1, DX debería dar 0 (no hay que desplazar nada). Si habéis añadido un "margen" M, sumar M/2 a DX,DY para dejar margen por los dos lados.

Dar los valores Xmin,Xmax,Ymin,Ymax obtenidos para vuestro mosaico y las correspondientes dimensiones (ancho/alto) y desplazamientos (DX,DY) a usar. Indicad si habéis usado margen y cuál.

Una vez decididos a aplicar un desplazamiento (DX,DY) a TODAS las imágenes podéis crear la matriz 3x3 de translación correspondiente a esos desplazamientos:

$$T_{xy} = \begin{pmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{pmatrix}$$

y aplicarla a todas las transformaciones de la lista T{} (recordad que este desplazamiento de coordenadas es el último paso y tenerlo en cuenta a la hora de decidir el orden en la multiplicación de matrices).

Ejercicio 8: Finalmente aplicaremos las transformaciones T{} a las imágenes de partida y las iremos componiendo sobre el mosaico final.

Para que el proceso sea razonablemente rápido modificaremos la función interpola del laboratorio anterior. En la versión anterior la función era:

```
function im = interpola(im,P)
```

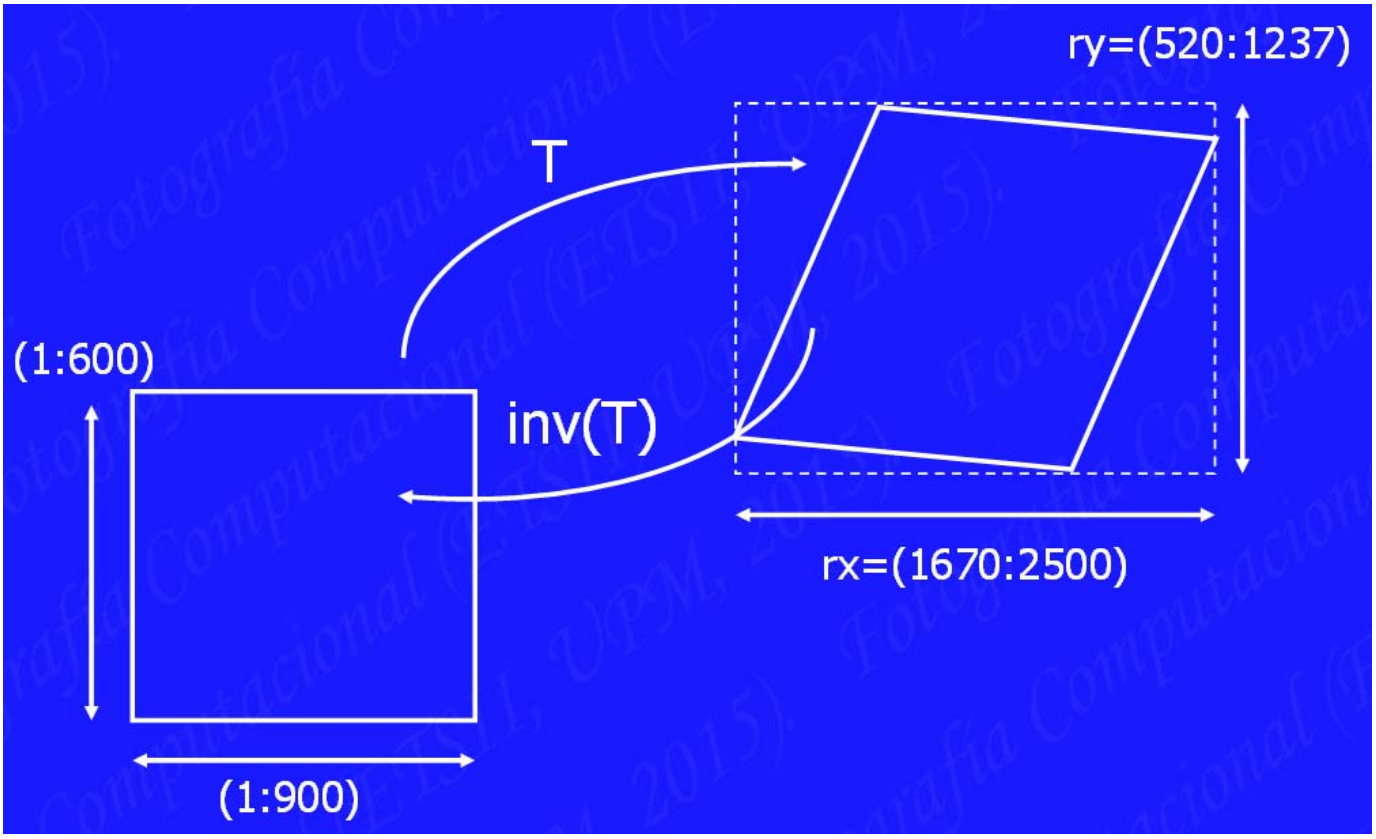
Dentro de la función se aplicaba P a las coordenadas X desde 1:M (ancho de la imagen) y coordenadas Y desde 1:N (alto de la imagen). Esto es, se interpolaba sobre las coordenadas originales de la imagen. Esto estaba bien para imágenes que no se movían mucho, pero con las transformaciones actuales si hago esto la imagen transformada puede salirse de las coordenadas (1:N) x (1:M) y no voy a ver nada.

Añadiremos un par de argumentos de entrada adicionales rx,ry: unos vectores con las coordenadas donde se quiere interpolar. P.e. rx=(1000:1700) y ry=(600:1200):

```
function im = interpola(im,P,rx,ry)
```

Dentro de la función la única modificación es que en los bucles, en vez de usar los valores de i y j directamente para las coordenadas Y, X , usaremos Y=ry(i), X=rx(j). Notad que en este caso el resultado no tiene por que tener el tamaño de la imagen original sino las dimensiones de los vectores ry (alto) y rx (ancho).

Para optimizar el proceso, cuando vayamos a transformar una imagen, primero se aplica la transformación T{} adecuada al rectángulo (1 600) x (1 900) para ver en que rango ry x rx de coordenadas va a parar la imagen transformada (ver figura adjunta):



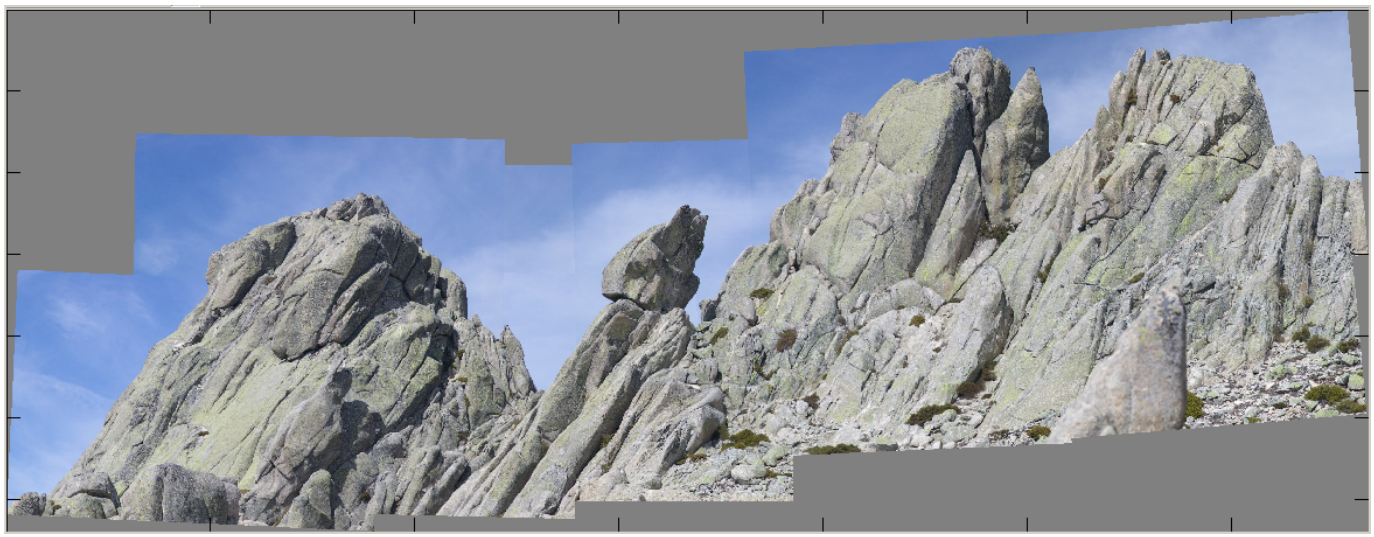
Una vez determinados (ry,rx) los usaremos para interpolar la imagen usando la transformación inversa. Así sólo barremos el área donde va a aparecer la imagen transformada, ahorrando trabajo.

Repetir en un bucle para cada imagen de partida, hallar el rango de coordenadas destino ry x rx y llamar a interpola para hallar imagen transformada.

Usar el resultado de interpola para llenar el cuadrado de coordenadas ry x rx en el mosaico final.

La función interp2 (usada en interpola) devuelve NaN para aquellos puntos fuera de la imagen de partida. Aprovechad esto para usar sólo los valores correctos de la interpolación cuando estamos rellenando la imagen mosaico.

Usando image() ir mostrando la imagen mosaico según la vais construyendo. En cada paso del bucle se irá añadiendo una imagen extra, hasta terminar con una imagen como esta:



Además de incluir el código en el fichero de respuestas adjuntar todo el **código necesario para que funcione el programa. Entregadlo en un .rar o similar junto con la hoja de respuestas.**

EXTRAS:

Si lo habéis hecho todo correctamente debería funcionar automáticamente para cualquier otro conjunto de imágenes con mínimos cambios (cambiando el nombre y número de imágenes, su tamaño, etc.). Procurar que todos estos parámetros estén bien etiquetados en vuestro programa para que sea sencillo cambiar su valor.

Hacer un mosaico de la facultad (perdón, escuela) o de alguno de los edificios del campus. Algunas recomendaciones:

- Tomar un mínimo de 6 fotos, usando la focal lo más larga (tele) posible. No las hagáis desde muy cerca.

- Reducirlas a tamaño 600 x 900 (formato 3/2) o 600 x 800 (formato 4/3) para que sea todo más rápido (usar IrfanView o similar). La función imresize de MATLAB también puede valer. Acordaros de que hay partes de vuestro código que dependen del tamaño de las imágenes empleadas.

- Usar SiftWin32 para obtener lista de keypoints (ejercicio 3). Recordad:

  Pasar la imagen a niveles de gris, guardar como pgm, aplicar Sift32Win.
  Parsear fichero de keypoints para leer datos.
  Reducir el número de puntos seleccionando un rango de escalas dado.

- Guardar los datos en un array de celdas s{1}, s{2}, etc.

- Aplicar el programa tal como lo tenéis.

Adjuntar FOTOS + programa completo + Fichero .mat que guarde los keypoints para poder obviar la fase de la extracción de parámetros.

Si tenéis los datos en una array s{} basta hacer >> save datos_puntos s

Otras mejoras:

- Prever el caso de que 1 o más imágenes no pertenezcan al panorama. Tras construir la matriz Q el programa debería avisarnos de que imágenes se quedan fuera. Podríamos marcarlas con un NaN en la T{k} y usar isnan() para detectarlas luego e ignorarlas en las fases posteriores.

- Usar transformaciones proyectivas en vez de afines. En las transparencias del tema de transformadas de dominio tenéis indicaciones sobre como determinar los parámetros de una transformación proyectiva.