Proyecto de Transformaciones Geométricas

1) Deformaciones lineales

Obtención de la matriz P de una transformación proyectiva

Completar la función **P=get_proy(xy,uv)**. Los parámetros de entrada son los cuatro vértices (xy) del cuadrilátero origen y los correspondientes cuatro vértices (uv) del cuadrilátero destino. En los dos casos, las coordenadas se dan como matrices 2x4:

$$xy = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \end{pmatrix} \qquad uv = \begin{pmatrix} u_1 & u_2 & u_3 & u_4 \\ v_1 & v_2 & v_3 & v_4 \end{pmatrix}$$

El resultado P será la matriz de la transformación proyectiva entre ellas $\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{pmatrix}$.

Seguid las indicaciones de las transparencias para hallar la matriz P a partir de las coordenadas origen xy y destino uv. Una vez completada la función, probadla con las coordenadas:

```
xy = [  0 600 465 215;   0  65 680 585];
uv = [275 655 365  25;  30 285 755 340];
```

Volcad la matriz P obtenida (usad la función dump_mat(P) que os doy)
Adjuntad código de get_proy.m.

Para verificar que P es correcta escribiremos una función **uv=convierte(xy,P).** Esta función recibe las coordenadas xy (2xN) en el espacio de partida y les aplica la transformación indicada por la matriz P devolviendo las coordenadas de dichos puntos (uv, también 2xN) en el espacio de destino. Para ello:

- Añadir a la matriz xy una 3ª fila de 1's para formar la matriz de coordenadas homogéneas 3xN (1ª fila = x's, 2ª fila = y's, 3ª fila = 1's).
- Multiplicad P por esta matriz de coordenadas homogéneas, obteniendo una nueva matriz también de tamaño 3xN.
- Las coordenadas de salida uv (2 x N) serán las dos primeras filas de la matriz resultante, DIVIDIDAS PUNTO a PUNTO por la tercera fila.

Comprobad vuestra función y matriz P haciendo uv2=convierte(xy,P). La matriz resultante uv2 debe ser prácticamente idéntica a las coordenadas uv iniciales, ya que P la hemos construido para pasar de las coordenadas (x,y) a (u,v). Adjuntad los valores de las diferencias entre uv y uv2. ¿A qué coordenadas u,v iría a parar un punto con coordenadas (x=200,y=100) al aplicarle la transformación dada por P?

Si calculáis la inversa de P, inv(P), y la usáis con convierte() pero ahora aplicada a las coordenadas (uv), debéis obtener las coordenadas xy. Esto nos muestra que con transformaciones lineales podemos obtener la matriz de la transformación inversa

haciendo la inversa de la matriz de la transformación original. Verificarlo calculando con get_proy() la matriz iP que pasa de uv a xy. Compararla con la inversa P^{-1} que habéis calculado antes. ¿Son iguales? Justificad este resultado con las propiedades que vimos de las matrices proyectivas.

Adjuntad código de la función convierte()

Deformación de imagen usando la transformación dada por una matriz P

Vamos a escribir una función warp_img() que aplique una transformación lineal a una imagen, devolviendo la imagen deformada:

```
function im2=warp_img(im,iP)
```

La función recibe una imagen im (tipo double con valores entre 0 y 1) y la matriz iP (3x3) de la transformación inversa a aplicar. Usamos la inversa porque (como vimos en clase) para hallar la imagen deformada (im2) haremos un "warping" inverso, barriendo las coordenadas de destino (en im2) y aplicando P^{-1} para ver donde caen sobre la imagen original. Adicionalmente tendremos que hacer una interpolación porque estas coordenadas transformadas no caerán exactamente en los píxeles enteros de la imagen original.

Para interpolar usaremos la función interp2 cuyo uso es **`A_XY=interp2(A,X,Y,tipo)`**. La función recibe una matriz A (2D) y coordenadas (X,Y) posiblemente no enteras, devolviendo el valor interpolado (A_XY) en la posición (X,Y). El último parámetro nos permite elegir el tipo (**`'nearest', 'bilinear'`**, …) de interpolación a usar.

Por defecto, interp2 asume que las coordenadas de la matriz original A van de 1 a N (alto de imagen) para las Y's y de 1 hasta M (ancho imagen) para las X's. Si las coordenadas (X,Y) son un único valor el resultado será el valor de A interpolado en ese punto. Los argumentos (X, Y) también pueden ser un par de matrices, en cuyo caso el resultado es una matriz con todos los valores interpolados para todas los pares de coordenadas dadas.

Dentro de la función warp_img() debéis:

1) Determinar el alto N, ancho M y número de planos de color de la imagen im
2) Con la función zeros() reservar espacio para la imagen de salida im2 del mismo tamaño que la original. Cread también sendas matrices X e Y (2D) de tamaño NxM para guardar las coordenadas donde realizaremos la interpolación.
3) Para rellenar las coordenadas a interpolar X e Y recorreremos las coordenadas destino, aplicándo P^{-1} para ver donde caen en la imagen de partida. Para que sea más eficiente transformaremos una fila entera cada vez. Para ello, preparar una matriz uv (de tamaño 2xM), siendo M la dimensión de una fila de la imagen. En su 1ª fila poned los valores de 1 a M (coordenadas u de la fila). En su 2ª fila iremos poniendo la coordenada v de la correspondiente fila a procesar. Para ello barred las filas con un bucle desde k=1 a N (alto de la imagen) y en cada paso:

a) Poned todos los elementos de la 2ª fila de uv al valor de k (la coordenada v común a toda la fila).
b) Usando **`convierte()`** y aplicando la transformación inversa a las coordenadas uv obtener las coordenadas xy (en espacio de origen) correspondientes a la k-ésima fila de la imagen destino.
c) Extraer las coordenadas x e y del resultado y guardarlas en la k-ésima fila de la matriz X e Y respectivamente.

4) Completado el bucle y rellenas las matrices X, Y con las coordenadas a interpolar solo queda usar interp2 para que MATLAB haga la interpolación en esos puntos (usando `'bilinear'` como tipo de interpolación). **Como la función interp2 trabaja con matrices 2D habrá que hacer un bucle interpolando cada plano de color de im por separado y guardando el resultado en el correspondiente plano de im2.**

Adjuntad código de warp_img().

Cargad la imagen del fichero "foto.jpg" y convertirla a double con valores entre 0 y 1. Vamos a deformar esta imagen aplicándole la transformación P del apartado anterior (recordad que para aplicar la deformación dada por P hay que pasarle a la función warp_img la matriz inversa P^{-1}). Adjuntad la imagen resultante.

Las zonas en negro del exterior corresponden a puntos que caen fuera del soporte de la imagen original y que por lo tanto no pueden ser interpolados a partir de ella. ¿Qué valor devuelve MATLAB en esos puntos? También se ve que partes de la imagen original no aparecen en la imagen deformada. Eso es porque solo se barre un rectángulo con el mismo "tamaño" que la imagen original. Dependiendo de la transformación usada, partes de la imagen deformada pueden salirse de esa zona.

Una vez completadas estas funciones os funcionará el programa demoP4. En él se muestran la imagen origen (izquierda) y destino (derecha) con los dos cuadriláteros superpuestos que definen la deformación. Pinchando con el ratón y arrastrando los vértices podéis mover las esquinas de los ambos cuadriláteros. Usando los vértices de los cuadriláteros el programa (con vuestra función get_proy) calcula la matriz P y la usa para deformar la imagen (con warp_img) que muestra a la derecha.

Llamándole con demoP4(im) podéis usar vuestra propia foto en vez de la mía. Usando una foto vuestra, tratad de obtener para la imagen deformada un resultado similar al que se muestra en la figura adjunta. Partid de una fotografía vuestra en modo retrato (cámara vertical) con una resolución de unos 900x600 o similar (podéis usar las funciones imcrop e imresize de MATLAB para ajustar la foto de partida a estas dimensiones). Adjuntad captura de demoP4 donde se muestre la imagen original y la deformada.

Volcad también las coordenadas de los vértices de ambos cuadriláteros para vuestro resultado y la matriz P correspondiente a la deformación usada (dump_mat).

Transformaciones recursivas

Una transformación proyectiva permite transformar cualquier cuadrilátero en otro, lo que se puede aprovechar para insertar una fototografía (un rectángulo) dentro de cualquier otra superficie plana en una 2ª foto (que por efecto de la perspectiva aparecerá como otro cuadrilátero). En este caso las coordenadas de partida serían las esquinas de la 1ª foto y las de llegada, las del cuadrilátero destino en la 2ª foto:

Usando la misma imagen como origen y destino podemos crear fotos "recursivas", insertando sucesivas copias de una imagen dentro de sí misma. Es típico usar una foto en la que aparece un cuadro o similar, dentro del cual insertaremos la imagen original (donde a su vez aparece de nuevo el cuadro con una nueva imagen, etc.).

Se trata de que uséis las funciones desarrolladas durante esta práctica para crear una imagen recursiva a partir de la foto de "billboard.jpg". Algunos comentarios:

1) Nuestro objetivo es usar una transformación proyectiva que, aplicada sobre la imagen completa la proyecte dentro del espacio del cartel. Adjuntad la matriz P de la transformación a usar.

2) Usad la función warp_img() para deformar la imagen usando la P anterior. Antes de deformarla es conveniente filtrar la imagen original con un filtro gaussiano (LAB anterior) con un ancho $\sigma \sim 1$ o 1.5. La razón es que la transformación va a reducir la resolución de la imagen y pueden aparecer efectos de aliasing si no eliminamos antes algo del detalle de la imagen original (que ya no puede ser representado adecuadamente con la resolución menor). Adjuntad la imagen resultante de aplicar la transformación.

3) Para fundir ambas imágenes usad el hecho de que la imagen deformada tendrá valores NaN en las zonas de fuera del cartel. De esta forma se puede saber en qué puntos de la imagen final se usan datos de la imagen original y en cuáles los datos de la imagen deformada. Se valorará hacerlo sin bucles. Adjuntad una captura de la imagen final resultante.

4) En la nueva imagen aparecerá un nuevo cartel publicitario dentro del espacio del antiguo. Indicad cómo podríais repetir el proceso para insertar copias adicionales de la foto dentro del nuevo cartel SIN TENER QUE VOLVER A MEDIR más coordenadas.

Adjuntad código usado en este apartado.

2) Ejemplo de transformación no lineal

Aunque el uso de transformaciones lineales (definidas a través de una matriz 3x3) es muy conveniente, también es posible usar cualquier otro tipo de función ($R^2 \rightarrow R^2$) para transformar las coordenadas de la imagen. En este apartado exploraremos una sencilla transformación no lineal, dada por:

$$u = A + B \cdot x + C \cdot y + D \cdot x \cdot y$$
$$v = a + b \cdot x + c \cdot y + d \cdot x \cdot y$$

Observaréis que es similar a una transformación afín, con la diferencia del cuarto término (coeficientes d y D) donde introducimos una no-linearidad con el producto (x·y). La transformación tiene 8 parámetros, por lo que necesitaremos en principio 4 puntos para fijarla (cada punto aporta 2 ecuaciones), los mismos que en la transformación proyectiva.

Lo primero es escribir una función **`function P=get_nolineal(xy,uv)`** (equivalente a get_proy) que reciba cuatro coordenadas origen (xy) y de destino (uv) en el mismo formato de antes (matrices 2X4). Tras calcular los coeficientes de la transformación se devuelven como una matriz P, ahora de tamaño 2x4:

$$P = \begin{pmatrix} A & B & C & D \\ a & b & c & d \end{pmatrix}$$

En las transparencias podéis ver cómo se plantea el sistema linear para obtener los coeficientes de la transformación. La única diferencia con el caso afín es que la matriz H del sistema a resolver tendrá una columna adicional con el producto de las x's y las y's. Una vez construida la matriz H resolver para las coordenadas u (obteniendo A, B, C, D) y para u (a, b, c, d). Agrupar estos coeficientes como las dos filas de la matriz P. Adjuntad código de vuestra función.

Usadla para obtener la matriz P de coeficientes que transforma las coordenadas del origen **`xy = [401 643 299 99; 201 296 646 268];`** en las coordenadas del espacio destino **`uv = [269 545 276 26; 350 371 744 412];`**

Usando dump_mat() volcar la matriz P con los coeficientes de la transformación directa (de xy a uv).

Notad que si ahora queremos calcular los coeficientes de la transformación inversa podemos hacerlo como antes invirtiendo el orden de xy, uv al llamar a la función. Sin embargo, al contrario que en el caso lineal, ya no existe una relación sencilla entre los coeficientes de la transformación inversa y los de la directa. Adjuntad el volcado de la matriz de coeficientes de la transformación inversa (de uv a xy).

A continuación vamos a modificar la función **`uv=convierte(xy,P)`** para que maneje también el caso no lineal. Ahora P será una matriz 2x4 con los coeficientes (A, B,..., a, b,...) de la transformación no-lineal en vez de la matriz 3x3 del caso lineal.

Se podría usar un parámetro adicional para indicar si tenemos una transformación lineal o no, pero es más elegante hacerlo automáticamente comprobando el tamaño de P. Para ello usaremos la función numel() que devuelve el nº de elementos de una matriz:

- Si detectamos que P tiene 9 elementos (matriz 3x3) estamos en el caso lineal y se ejecutaría el código que ya tenemos, donde se construía una matriz 3xN cuya 1ª fila eran 1's, la 2ª las x's y la 3ª las y's.

- En el caso no lineal (P es 2x4) construiremos una matriz (4xN) con una fila de 1's, una fila con las x's, otra con las y's y una 4ª fila adicional con el producto de las x's y las y's. Luego se hace el producto de P (2x4) por dicha matriz (4xN), obteniendo así una matriz 2xN con las coordenadas transformadas u (en su 1ª fila) y v (2ª fila).

Adjuntad código de vuestra función convierte() modificada.

Como en el caso lineal, si habéis calculado correctamente la matriz P, el resultado de hacer convierte(xy,P), debe ser muy parecido a las coordenadas uv destino. Adjuntad matriz de las diferencias entre uv y las coordenadas obtenidas.

Igualmente, aplicando la función convierte() a las coordenadas destino uv (usando los coeficientes de la transformación inversa) debéis recuperar las coordenadas originales xy.

Deformación de una imagen con esta transformación no lineal

Sin cambiar nada, la función **`warp_img(im,iP)`** también debería funcionar ahora para el caso no lineal. Como la función llama a convierte() para calcular las coordenadas de origen, si detectamos que las dimensiones de P son 2x4 se aplicará de forma automática la transformación no lineal.

Hacer un warping no lineal de la imagen 'foto.jpg' usando la transformación P que se halló en el ejemplo anterior. Al igual que en el caso lineal habrá que usar la matriz de la transformación inversa. Adjuntad la imagen resultante. Observaréis una imagen "doble" al aplicar la transformación. Eso quiere decir que el mismo punto del espacio de partida puede ir a varios puntos en el de destino (esto nunca pasaría con transformaciones lineales). ¿Qué imagen obtendríamos si usamos la transformación P directa (de xy a uv) al llamar a warp_img? Adjuntad captura del nuevo resultado.

Tras completar estas funciones podréis usar también la opción no-lineal de demoP4 (seleccionando la opción no-lineal). De nuevo, usando vuestra propia foto tratad de reproducir una imagen similar a la que habéis obtenido aquí, donde se vea una "doble" imagen en la imagen deformada.

Adjuntad captura de demoP4 con la imagen original y la deformada. Volcad como en el caso lineal las coordenadas de los vértices de ambos cuadriláteros y la matriz P de coeficientes de la transformación usada.

3) Seam Carving

En los apartados previos, tanto en el caso lineal 1) como en el ejemplo no lineal 2) las deformaciones siguen el mismo enfoque: aplicar una función T a las coordenadas (x,y) de partida para obtener las coordenadas (u,v) de llegada (o viceversa).

En este apartado veremos un tipo de deformación distinta, donde no existe una relación fija entre las coordenadas de origen y destino, sino que es dependiente del contenido de la propia imagen.

Este apartado está basado en el artículo "Seam Carving for Content-Aware Image Resizing" de S. Avidan y A. Shamir (https://faculty.idc.ac.il/arik/SCWeb/imret/). Es una método para cambiar la relación de aspecto de una imagen a base de eliminar o añadir píxeles a lo largo de las llamadas "costuras" de energía mínima.

Cargar la imagen img.jpg. Para el resto del ejercicio convertirla a double y escalarla entre 0 y 1 usando im2double(). Indicad sus dimensiones y su relación de aspecto.

Lo primero que necesitamos es definir una función asociada a la posición de cada píxel que nos indique el nivel de detalle del píxel (más alta cuanto más detalle). La implementaremos con el template **`function E=energia(im)`**. Dentro de la función suavizaremos la imagen con un filtro gaussiano de soporte 7x7 y σ=1.5, usando la opción 'sym' para los bordes. Luego (como vimos en el LAB anterior) calcularemos el valor absoluto de la diferencia con la imagen original. Esta diferencia es también una imagen con tres planos de color. Sumad los tres planos de color para obtener una imagen detalle (con valores siempre positivos) con un único plano. A partir del detalle, definiremos la energía E a devolver por la función como 10·log2(1+detalle). E debe ser una matriz del mismo ancho y alto que la imagen original (sin planos de color). Adjuntad código de vuestra función, el valor de E en el píxel (10,10) y el valor medio (mean2) de E.

Visualizar el resultado de E con **`imagesc();`** usando la paleta de colores predefinida "jet" con **`colormap("jet")`**. Poned una barra vertical al lado de la imagen mostrando la relación entre lvalores de E y los colores usados. Adjuntad la imagen resultante.

Eliminación de píxeles con menor energía

Queremos convertir la imagen dada al formato 3/2, típico en fotografía tradicional. Si mantenemos la altura de la imagen, ¿cuál debería ser ahora su ancho para tener una relación de aspecto 3:2? ¿Número n de columnas a eliminar de la foto original?
Una opción (óptima en cuanto a maximizar la "energía" o detalle de la imagen final) sería eliminar de cada fila el píxel de menor energía, obteniendo así una imagen con una columna menos. Tras re-calcular la energía de la nueva imagen se repetiría el proceso n veces. En cada paso la imagen pierde una columna para terminar con el tamaño deseado. Al terminar calcular la media de la energía de la imagen final.
Obviamente debe ser mayor que la calculada antes, ya que los píxeles eliminados han sido los de menor energía.

Adjuntad código usado y la imagen resultante. Vemos que el resultado final no es aceptable. La razón es que al no haber exigido continuidad entre los píxeles que se eliminan de cada fila la imagen se distorsiona por completo.

Eliminación de costuras verticales

Para evitar este problema exigiremos una cierta conectividad entre los puntos a eliminar. Se define una costura (vertical) como una lista de coordenadas (i, j) donde el índice i barre todas las filas (i=1,2,3,..., ALTO) y las coordenadas {j's} deben verificar que |j(i) - j(i-1)|<1. Esto es, si en la 1ª fila decidimos eliminar el píxel de la columna 37, en la 2ª fila solo podríamos eliminar los píxeles de las columnas 36, 37 o 38. Así se garantiza una continuidad en la línea de los puntos eliminados y de esta forma esperamos preservar mejor la integridad de la imagen.

El objetivo, partiendo del mapa de energía E calculado para una imagen, es hallar la costura vertical de puntos "adyacentes" (según el criterio anterior) cuya suma de energías sea menor que la de cualquier otro posible recorrido vertical con nuestras condiciones. En una costura vertical las coordenadas {i's} son siempre (1:ALTO) por lo que no hace falta guardarlas. En cuanto a las coordenadas {j} las guardaremos en un vector columna s (tamaño ALTO x 1). Nuestra restricción de la conectividad implica que |s(j) - s(j-1)| debe ser siempre ≤1.

El primer paso es escribir la función **`function M=calcula_M(E)`** que calcule la energía mínima acumulada M siguiendo todas las posibles costuras que llegan al píxel (i,j). El proceso es el siguiente:

- Reservar una matriz M del mismo tamaño que E, con sus valores inicializados a infinito (Inf), con **`M=Inf(size(E));`**
- Llenamos la última fila de M, desde j=2 hasta ancho-1, con los valores de E: M(end,j) = E(end,j)
- Luego retrocedemos el resto de las filas, desde la penúltima hasta la primera, haciendo en cada caso:

$$M(i,j) = E(i,j) + \min\{ M(i+1,j-1:j+1) \}$$

En cada paso se suma a la energía del píxel E(i,j) la mínima energía acumulada de las 3 posibles opciones que tenemos en la fila siguiente. Para evitar los problemas en los extremos de la fila (donde preguntaríamos por valores fuera de la imagen) calcularemos sólo los valores de M desde la 2ª columna a la penúltima, dejando la 1ª y última columna de M como Inf.

Adjuntad código de vuestra función y los 3x4 valores de M en la esquina superior derecha (las 4 últimas columnas de las 3 primeras filas de M).
Visualizar la M obtenida para la imagen original como antes hicimos con E, pero ahora usando la paleta de color "hot". Adjuntad captura del resultado.

Una vez calculada la matriz M, escribiremos una función **`function s=find_seam(M)`** para encontrar la costura de menor energía presente en M, y que nos devuelva sus coordenadas en un vector columna de tamaño ALTOx1.

Por definición, la primera fila de M contiene la energía acumulada mínima de todas las posibles costuras que empiezan en cada píxel a lo largo del ancho de la imagen. Adjuntad un plot de los valores de la primera fila de la matriz M. La posición del mínimo de esta primera fila de M nos da la columna j donde empieza la costura más "económica", por lo que tendremos que s(1)=j. ¿Valor y posición de dicho mínimo?

Desde ese punto final se trata de avanzar fila por fila (2→ALTO) buscando el camino de menor energía seguido por esa costura. Como conocemos s(k-1), para el valor de s(k) solo hay tres posibilidades: s(k-1)-1, s(k-1) o s(k-1)+1, ya que el índice solo puede cambiar en ±1. Para decidir entre ellas se consulta el valor de la k-ésima fila de M en esos tres índices y se escoge para s(k) el índice que tenga un menor valor de M. Una vez llegados a la última fila se ha completado la costura y tenemos el vector s con las coordenadas {j's} a eliminar en cada fila.

Adjuntad código de la función y mostrar la costura mínima encontrada superpuesta (en verde) sobre la imagen original. Volcad las coordenadas de los 3 últimos puntos de la costura. Sumando los valores de E a lo largo del trayecto debéis verificar que la suma coincide con el valor de la primera fila de M en el inicio de la costura.

Ya solo hay que eliminar esa costura de la imagen. Para ello barremos con un bucle todas las filas de la imagen eliminando en la i-ésima fila el píxel s(i) dado por la costura. Tras repetir el proceso n veces la imagen tendrá el ancho deseado. Eso sí, en cada paso habrá que re-calcular la energía E y la energía acumulada M, ya que al eliminar píxeles los vecinos cambian y con ellos la energía E y por consiguiente la energía acumulada M. Al terminar el proceso re-calcular la M de la imagen final y adjuntad el plot de su primera fila. Debéis ver valores más altos que en el creado con la imagen original, ya que los caminos más fáciles han sido ya "eliminados".

Adjuntad el código usado para eliminar columnas en la imagen.

Para presentar los resultados formad ahora una imagen poniendo una encima de la otra las siguientes imágenes (las tres deben tener las mismas dimensiones)

- La imagen original reescalada a las nuevas dimensiones usando imresize().
- La imagen original eliminando las primeras n columnas.
- La imagen que habéis obtenido con el algoritmo de seam-carving.

Adjuntad captura del resultado. Calculad como antes la energía media (mean2) de la imagen final. ¿Es mayor o menor que la obtenida en la imagen final del método anterior? ¿Por qué?

Inserción de costuras verticales

Hemos visto que una costura es una trayectoria a lo largo de la cual los píxeles vecinos difieren poco por lo que se nota menos al quitarla. Estas costuras también nos valen si, en vez de quitar, queremos añadir píxeles. Cuando se añade un píxel hay que inventarse su valor y eso es más fácil en zonas donde la imagen no cambia mucho ya que los valores de los píxeles son más previsibles.

El proceso es muy similar al anterior. Para aumentar el nº de columnas se calcula la energía de la imagen, su energía acumulada M y se encuentra la costura (s) que recorre la imagen de arriba abajo minimizando la suma de las energías del camino. Una vez conocida s se recorren todas las filas, con la diferencia de que ahora, en vez de eliminarlo, se duplica el píxel de la columna s(k), insertando un píxel nuevo. Así terminamos con una imagen con una columna más en vez de con una columna menos. Imaginad que ahora queremos añadir columnas a la imagen original hasta conseguir una relación de aspecto de 2:1. ¿Cuántas columnas habría que añadir? Adjuntad imagen resultante.

El problema es que al añadir un píxel en vez de quitarlo, al repetir el proceso el algoritmo siempre tiende a escoger la misma costura (o una mezcla con la que hemos añadido) con el resultado de que los mismos píxeles son replicados una y otra vez. Para evitar esto, calcularemos previamente las n costuras de energía más baja antes de empezar el proceso de ampliar la imagen:

1. Calcular la energía de la imagen original y reservar una matriz S (ALTO x n) para guardar las coordenadas de las n costuras encontradas.

2. Hacer un bucle de k=1 a n (el número de costuras a añadir). En cada paso:

    - Calcular energía acumulada M a partir de E (con la función ya escrita).

    - A partir de M, encontrad la costura de energía mínima s (usar la función ya escrita) y guardarla en la k-ésima columna de la matriz S.

    - Para evitar que en el siguiente paso se escoja la misma costura (o una con muchos pasos en común) penalizaremos a todos los píxeles de la última costura escogida multiplicando su energía E por un factor 1.5.

Mostrar la imagen original y superponer sobre ella gráficamente (en color verde) las n costuras verticales elegidas.

Ahora ya solo queda ampliar la imagen a lo largo de las N costuras. Para crear la imagen ampliada a partir de la información en S podríamos hacerlo de varias formas. Yo os describo una posibilidad pero podéis usar cualquier otra si lo preferís.

Crear una 2ª imagen im2 (llena de ceros) del tamaño necesario para la imagen ampliada. Barrer todas las filas de la imagen. Para cada fila k, en S(k,:) tenemos la lista de los píxeles originales a duplicar en esa fila. El objetivo es crear un vector idx

de tamaño NN (nº de columnas finales) cuyos valores vayan desde 1 a N (nº de columnas iniciales). En dicho vector todos los valores presentes en S(k,:) deben estar duplicados (o triplicados, etc. si aparecen más de una vez). Por ejemplo si partimos de una fila de 10 columnas que vamos a ampliar a 15 columnas y los píxeles a duplicar son S(k,:)={2, 5, 5, 7, 9} el vector idx resultante debería ser:

idx = [1 2 2 3 4 5 5 5 6 7 7 8 9 9 10].

Usando este vector de índices podemos rellenar la k-ésima fila de la nueva imagen ampliada con los correspondientes píxeles de la imagen original. Adjuntad el código usado para ampliar la imagen.

Tras terminar, como en el ejercicio anterior, presentad juntas (montándolas una encima de la otra) las tres imágenes siguientes (todas deben tener las mismas dimensiones):

- La imagen original reescalada a las nuevas dimensiones usando imresize().
- La imagen original añadiendo dos tiras negras de n/2 columnas a cada lado.
- La imagen obtenida con este método.

Adjuntad captura del resultado. Comentad las diferencias.

Hay que tener en cuenta que estos resultados pueden ser muy dependientes de las imágenes usadas. Para la imagen de prueba se obtienen buenos resultados (por eso la he escogido) porque el fondo está desenfocado respecto al primer plano, lo que hace que las costuras tiendan a preservar la cara (la zona con más detalle).

Partid de una foto de uno de vosotros con la misma relación de aspecto que la imagen que hemos usado en este apartado (podéis usar las funciones imcrop e imresize de Matlab para ajustar la imagen). Procurad que tenga un fondo con bastante detalle para ver las diferencias con respecto la foto del ejemplo. Procesadla como hemos hecho antes para:

- Quitar columnas hasta quedaros con un formato 3:2
- Añadir columnas para "estirarla" a un formato 2:1

Adjuntad la foto original y las dos imágenes resultado, indicando las dimensiones de las tres imágenes.

Posiblemente los resultados no serán tan buenos como los de la imagen de prueba.