Computational Photography **2014/2015** **P2**

Imágenes del Imperio Ruso

Material:

[1] Dataset: http://www.loc.gov/pictures/search/?sp=1&co=prok&st=grid

Background: Sergei Mikhailovich Prokudin-Gorskii (1863-1944) [Сергей Михайлович Прокудин-Горский, to his Russian friends] was a man well ahead of his time. Convinced, as early as 1907, that color photography was the wave of the future, he won Tzar's special permission to travel across the vast Russian Empire and take color photographs of everything he saw. And he really photographed everything: people, buildings, landscapes, railroads, bridges... thousands of color pictures! His idea was simple: record three exposures of every scene onto a glass plate using a red, a green, and a blue filter. Never mind that there was no way to print color photographs until much later -- he envisioned special projectors to be installed in "multimedia" classrooms all across Russia where the children would be able to learn about their vast country. Alas, his plans never materialized: he left Russia in 1918, right after the revolution, never to return again. Luckily, his RGB glass plate negatives, capturing the last years of the Russian Empire, survived and were purchased in 1948 by the Library of Congress. The LoC has recently digitized the negatives and made them available on-line.

En esta práctica vamos a reconstruir imágenes RGB a partir de tomas separadas de los tres canales R, G, B. Las imágenes pertenecen a la base de datos de la *Library of Congress* y se pueden bajar a través del enlace [1]. En la web hay ficheros en formato .jpg y ficheros en formato .TIF. Nosotros vamos a trabajar con los TIFs, ya que ofrecen una mejor calidad de la imagen.

Los pasos principales que se tienen que implementar en esta práctica son 3:

1) **separación** de la tres imágenes R, G, B y **alineación** de los tres canales, para reconstruir la imagen en color;

2) implementación de la alineación de imágenes utilizando la **descomposición piramidal** de la imagen;

3) una vez tengamos la imagen RGB, mejorar su aspecto trabajando sobre el brillo, el balance del blanco, el contrasto etc.

Proponemos repartir estas tres tareas en tres semanas de trabajo, aunque cada grupo puede organizarse como quiera, ya que la entrega es única y al final de la tercera semana.

1. Separación y alineación de los canales R, G, B

Las ficheros de la base de datos LoC [1] contienen los tres canales organizados en una única imagen, como en Figura 1.

Figura 1. Ejemplo de imágenes B, G, R, de la base de datos LoC.

Atención: el orden de las imágenes no es **R,G,B,** si no que **B,G,R**!!!

1.1 Reconstrucción

Para reconstruir la imagen RGB, tenemos que separar las tres imágenes, todas con el mismo tamaño, y juntarlas en una única imagen RGB. En Figura 2 hay un ejemplo de resultado de reconstrucción.

Figura 2. Ejemplo de reconstrucción de imagen RGB

Como se puede ver en la foto, la reconstrucción no es muy precisa, ya que las imágenes no están alineadas.

Podeis modificar el siguiente codigo para cargar y crear la imagen RGB

```
import matplotlib.pyplot as plt
from PIL import Image, ImageDraw
import numpy as np
import math
from scipy import ndimage
from scipy.misc import imresize

#LOAD IMAGES
img = Image.open("00029u.png")

#WARNING SIZE AND SHAPE INDICES ARE UPSIDE DOWN
size1_size=int(round(img.size[1]/10))
size2_size=int(round(img.size[0]/10))
img = imresize(img, ( size1_size,size2_size),interp='bilinear').astype('float')
img_np = np.array(img)


x1 = 20; y1 = 20;
x2 = 25; y2 = 333;
x3 = 24; y3 = 647;
w = 336; h = 302;

im1 = img_np[y1-2:y1+h-2,x1-1:x1+w-1];
im2 = img_np[y2-2:y2+h-2,x2-1:x2+w-1];
im3 = img_np[y3-2:y3+h-2,x3-1:x3+w-1];
I1 = 256*im1.astype('double')/im1.max();
I2 = 256*im2.astype('double')/im2.max();
I3 = 256*im3.astype('double')/im3.max();

RGB=np.zeros([h,w,3],dtype=type(img_np[0,0]))
RGB[:,:,0]=I3
RGB[:,:,1]=I2
RGB[:,:,2]=I1
```

1.2 Alineación

Para poder alinear las tres imágenes, tenemos que implementar un método de alineación automática, basado en búsqueda y matching de regiones de imágenes.

Para ello, podemos escoger una región, o más regiones de una imagen, y buscar la

correspondiente en las otras imágenes; por ejemplo una región cuadrada (**block to match**) de se puede buscar en un área más grande (**template**) de la otra imagen. En Figura 3 hay un ejemplo de búsqueda de unos bloques.

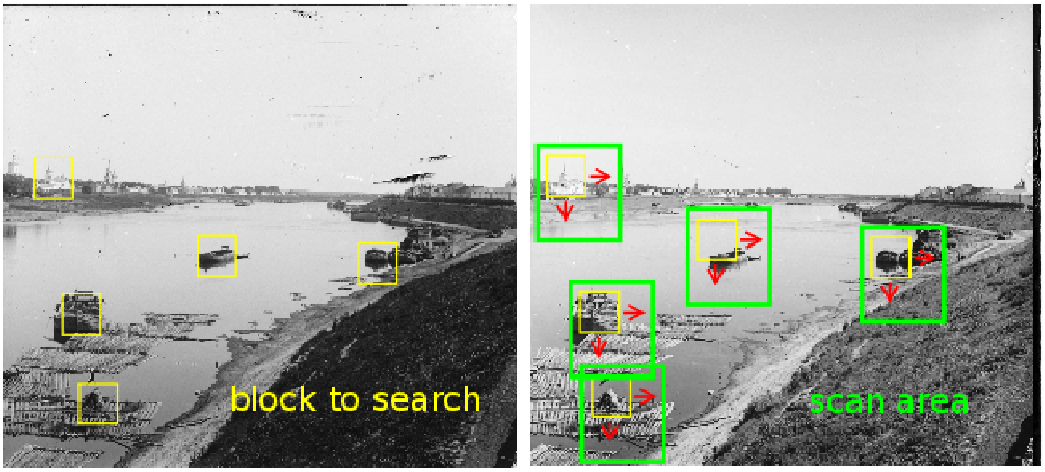


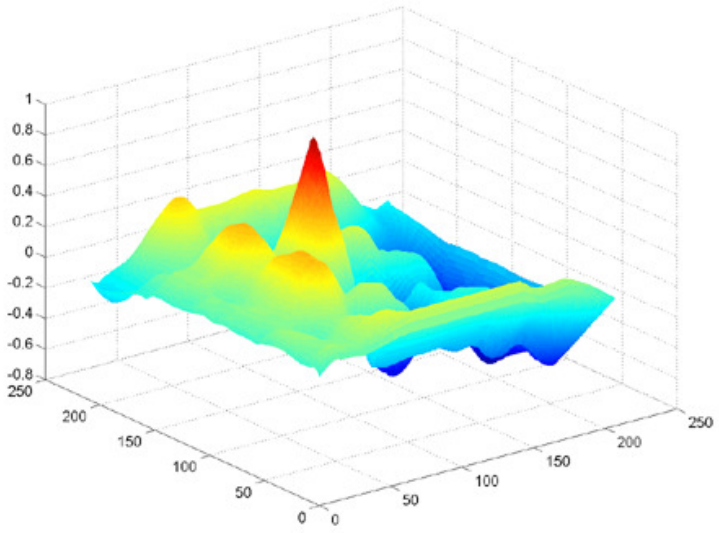
Figura 3. Ejemplo de block matching entre dos imágenes. El bloque que se tiene que buscar y el área de búsqueda están indicados.

Existe una implementación de template matching en la librería skimage, basada en el cálculo de la función de "cross-correlation".
http://scikit-image.org/docs/dev/auto_examples/plot_template.html
Sin embargo en esta práctica implementaremos manualmente la función de Cross Correlation, sin utilizar la librería función "np.unravel_index"
Para comparar bloques, usaremos la técnica del cross correlation. La documentación relativa a la función correlate2d en esta página:

http://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.correlate2d.html

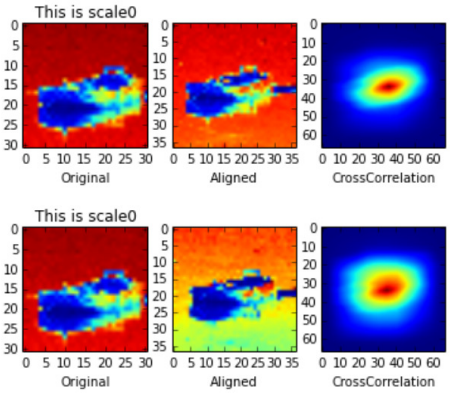
un ejemplo de uso es el siguiente:

NCC=correlate2d(block.astype('float'), template.astype('float'), mode='full', boundary='fill', fillvalue=0)



Como primer paso, deseamos alinear los canales R y G, y R con B de la imagen. Implementad una función que calcule la cross correlation de dos bloques centrado en las coordenadas X1 = 140; Y1 = 165; usando como parámetros 15 px (tamaño del bloque) y 3 (tamaño de la zona de búsqueda);
ATENCION, la imagen parametrica de Cross Correlation es mas grande del bloque inicial. Gestionar oportunamente los indices.
Una vez encontrado el vector (dx,dy) de desplazamiento entre las imágenes, imprimirlo a pantalla, y visualizar en *subplots* el bloque que estábamos buscando y su correspondiente encontrado según el vector de desplazamiento.

el resultado será el siguiente



2. Alineación con decomposición piramidal

Para conseguir una alineación de las imágenes más rápida, se puede implementar una búsqueda piramidal. En este caso, varias réplicas de la imagen original se calculan aplicando:

1) filtraje de suavización de la imagen, proporcionado a la escala considerada

2) sub-sampling de la imagen; entre dos escalas consecutivas hay un factor 2

En Figura 4 hay un ejemplo de decomposición piramidal.

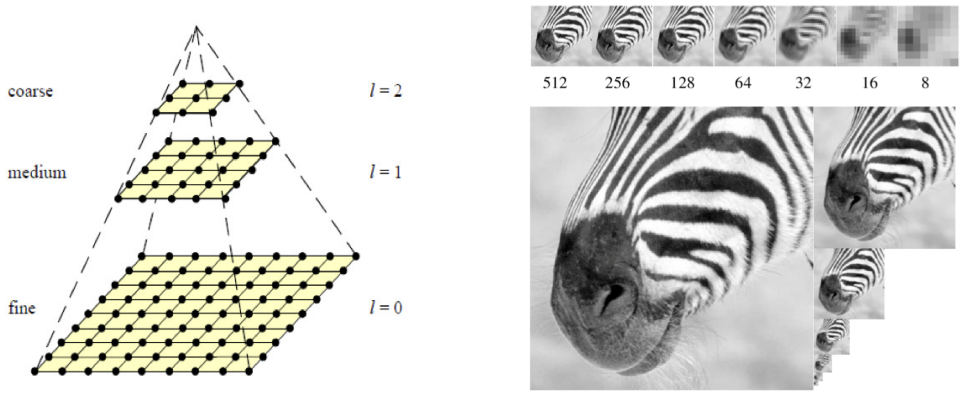


Figura 4. Ejemplo de decomposición piramidal de una imagen.

Para implementar la búsqueda a través de la decomposición piramidal, simplemente se puede re-utilizar el código implementado para la búsqueda a una escala (apartado 1) y adaptarlo para el cambio de escala. La implementación piramidal permite obtener los mismos resultados en menos tiempo, ya que nos permite utilizar áreas de búsqueda mucho más pequeñas!

Resultados esperados:

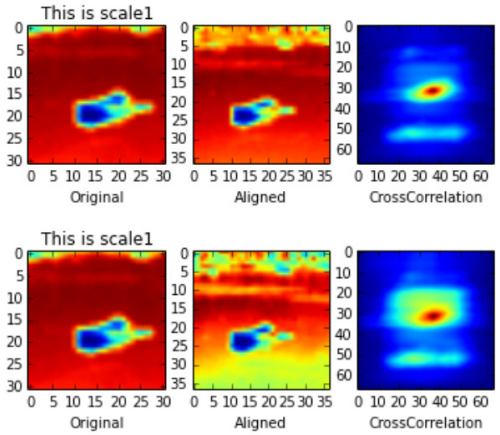


Figura 1 alineación de imagen del canal R con G (primera línea) , y del canal G con B (segunda línea) a la escala 1. La tercera columna indica la imagen de "normalized cross correlation" entre las dos imagenes.

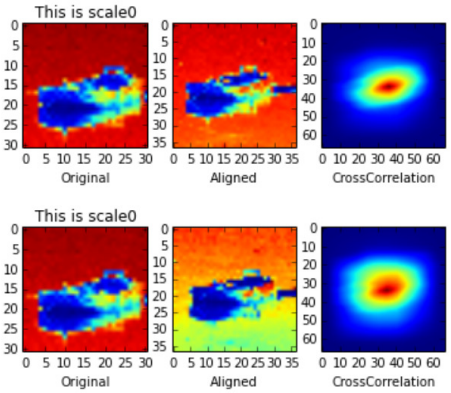


Figura 2 alineación de imagen del canal R con G (primera línea) , y del canal G con B (segunda línea) a la escala 0. La tercera columna indica la imagen de "normalized cross correlation" entre las dos imagenes.

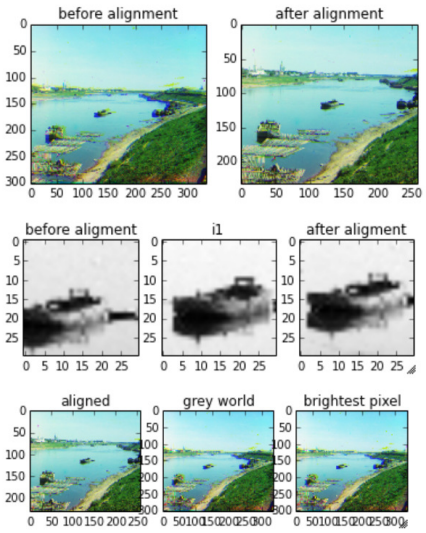


Figura 3 alineación de imagen completa (primera linea). detalle de imagen final (segunda linea), normalizacion de la escala de colores.

Presentar los resultados

Entregar todo el código de la practica, y un PDF (o el ipython commentado) que explique los resultados en el Campus Virtual **antes de la fecha límite.**

Para entregar y presentar vuestros resultados, es necesario:

1. Subir en una carpeta **TODO el código python y las imágenes utilizadas al Campus Virtual** de la UB antes de la fecha límite establecida con el profesor de prácticas.

2. Explicar posibles problemas y soluciones. La idea del texto es simplemente añadir informaciones útiles al lector para entender e interpretar el resultado conseguido.