Elaborazione di Segnali Multimediali

Elaborazione di immagini a colori
Soluzioni

1 Gli spazi di colore

1. *Lo spazio CMY e CMYK.*

```
def rgb2cmy(x):
   return 1.0 - x

x = io.imread('fragole.jpg')
x = np.float64(x)/255
z = rgb2cmy(x)

c = z[:,:,0]  # Ciano
m = z[:,:,1]  # Magenta
y = z[:,:,2]  # Giallo

plt.figure();
plt.subplot(1,3,1); plt.imshow(c,clim=[0,1],cmap='gray'); plt.title('Ciano');
plt.subplot(1,3,2); plt.imshow(m,clim=[0,1],cmap='gray'); plt.title('Magenta');
plt.subplot(1,3,3); plt.imshow(y,clim=[0,1],cmap='gray'); plt.title('Giallo');
```

Notate che in questo modo le componenti C, M e Y sono normalizzate. E' anche possibile visualizzare le componenti in scala di colore, modificando opportunamente la palette:

```
from matplotlib.colors import ListedColormap
L = np.arange(255,-1,-1)/255                  # 256 valori tra da 1 ad 0
B = np.ones(256)                              # 256 valori unitary
cmap_c = ListedColormap(np.stack((L,B,B),-1)) # palette per il Ciano
cmap_m = ListedColormap(np.stack((B,L,B),-1)) # palette per il Magenta
cmap_y = ListedColormap(np.stack((B,B,L),-1)) # palette per il Giallo

plt.figure();
plt.subplot(1,3,1); plt.imshow(c,clim=[0,1],cmap=cmap_c); plt.title('Ciano');
plt.subplot(1,3,2); plt.imshow(m,clim=[0,1],cmap=cmap_m); plt.title('Magenta');
plt.subplot(1,3,3); plt.imshow(y,clim=[0,1],cmap=cmap_y); plt.title('Giallo');
```

Per quanto riguarda invece lo spazio CMYK:

```
def rgb2cmyk(x):
   z = 1.0 - x       # da rgb a cmy
   k = np.min(z,-1)  # stima del Nero
   c = z[:,:,0] - k  # rimozione del Nero al Ciano
   m = z[:,:,1] - k  # rimozione del Nero al Magenta
   y = z[:,:,2] - k  # rimozione del Nero al Giallo
   return np.stack((c,m,y,k),-1)


z = rgb2cmyk(x)
c = z[:,:,0]  # Ciano
m = z[:,:,1]  # Magenta
y = z[:,:,2]  # Giallo
k = z[:,:,2]  # Nero

plt.figure();
L = np.arange(255,-1,-1)/255
cmap_k = ListedColormap(np.stack((L,L,L),-1)) # palette per il Nero
plt.subplot(1,4,1); plt.imshow(c,clim=[0,1],cmap=cmap_c); plt.title('Ciano');
plt.subplot(1,4,2); plt.imshow(m,clim=[0,1],cmap=cmap_m); plt.title('Magenta');
plt.subplot(1,4,3); plt.imshow(y,clim=[0,1],cmap=cmap_y); plt.title('Giallo');
plt.subplot(1,4,4); plt.imshow(k,clim=[0,1],cmap=cmap_k); plt.title('Nero');
```

2 Tecniche per l'elaborazione

1. *Negativo*.

```
x = io.imread('fragole.jpg')
x = np.float64(x)/255

y = 1.0 - x              # negativo

plt.figure();
plt.subplot(1,2,1); plt.imshow(x); plt.title('Immagine Originale');
plt.subplot(1,2,2); plt.imshow(y); plt.title('Immagine Nagate');
```

2. *Correzioni di toni e colori.*

```
x = io.imread('colori.jpg')
x = np.float64(x)/255
plt.figure(1); plt.imshow(x);
plt.title('Immagine originale');
gamma = 0.6;

# Elaborazione in RGB
y = x ** gamma
plt.figure(2); plt.imshow(y);
plt.title('Elaborazione in RGB');

# Elaborazione in HSI
from color_convertion import rgb2hsi, hsi2rgb
w = rgb2hsi(x)
w[:,:,2] = w[:,:,2] ** gamma
z = hsi2rgb(w)
plt.figure(3); plt.imshow(z);
plt.title('Elaborazione in HSI');
```

3. *Equalizzazione.*

```
x = io.imread('volto.tiff')
x = np.float64(x)/255
plt.figure(1); plt.imshow(x);
plt.title('Immagine originale');
from  skimage.exposure import equalize_hist

# Elaborazione in RGB
r = equalize_hist(x[:,:,0])
g = equalize_hist(x[:,:,1])
b = equalize_hist(x[:,:,2])
y = np.stack((r,g,b), -1)
plt.figure(2); plt.imshow(y);
plt.title('Equalizzazione in RGB');

# Elaborazione in HSI
from color_convertion import rgb2hsi, hsi2rgb
w = rgb2hsi(x)
w[:,:,2] = equalize_hist(w[:,:,2])
z = hsi2rgb(w)
plt.figure(3); plt.imshow(z);
plt.title('Equalizzazione in HSI');
```

4. *Color balancing*. Nello spazio CMY:

```
x = io.imread('foto.jpg')
x = np.float64(x)/255


c = 1 - x[:,:,0]
c = c ** 1.8
y = np.stack((1-c, x[:,:,1],x[:,:,2]), -1)

z = io.imread('foto_originale.tif')
z = np.float64(z)/255

plt.figure(1)
plt.subplot(1,3,1); plt.imshow(x); plt.title('troppo ciano')
plt.subplot(1,3,2); plt.imshow(y); plt.title('immagine elaborata');
plt.subplot(1,3,3); plt.imshow(z); plt.title('immagine originale');
```

E' anche possibile lavorare nello spazio RGB: per diminuire la quantita' di ciano bisogna aumentare il rosso.

5. *Filtraggio spaziale*.

```
x = io.imread('lenac.jpg')
x = np.float64(x)/255
plt.figure(1); plt.imshow(x);
plt.title('Immagine originale');
import scipy.ndimage as ndi

# filtraggio nello spazio RGB
y = ndi.uniform_filter(x, (5,5,1))
plt.figure(2); plt.imshow(y);
plt.title('Filtraggio in RGB');

# filtraggio nello spazio YUV
from skimage.color import rgb2yuv, yuv2rgb
w = rgb2yuv(x)
w[:,:,0] = ndi.uniform_filter(w[:,:,0], (5,5))
z = yuv2rgb(w)
plt.figure(3); plt.imshow(z);
plt.title('Filtraggio in YUV');
```

Per quanto riguarda l'enhancement mediante laplaciano, anziché determinare il laplaciano per poi sottrarlo all'immagine applichiamo direttamente la maschera che realizza entrambe queste operazioni.

```
x = io.imread('fiori.jpg')
x = np.float64(x)/255
plt.figure(1); plt.imshow(x);
plt.title('Immagine originale');
import scipy.ndimage as ndi

#filtro di Enhancement
h = np.array([[0,-1,0],[-1,5,-1],[0,-1,0]], dtype=np.float64)

# Enhancement nello spazio RGB
h_3d = np.expand_dims(h, -1)  # maschera 3D
y = ndi.correlate(x, h_3d)
plt.figure(2); plt.imshow(y);
plt.title('Enhancement in RGB');

# Enhancement nello spazio HSI
from color_convertion import rgb2hsi, hsi2rgb
w = rgb2hsi(x)
w[:,:,2] = ndi.correlate(w[:,:,2], h)
z = hsi2rgb(w)
plt.figure(3); plt.imshow(z);
plt.title('Enhancement in HSI');
```

2.1 Esercizi proposti

1. *Variazione del colore.*

```
x = io.imread('Azzurro.jpg')
x = np.float64(x)/255
plt.figure(1); plt.imshow(x);
plt.title('Immagine originale');

from color_convertion import rgb2hsi, hsi2rgb
y = rgb2hsi(x)
H = y[:,:,0]; S = y[:,:,1]; I = y[:,:,2];
plt.figure(2);
plt.subplot(1,3,1); plt.imshow(H,clim=[0,1],cmap='gray');
plt.title('Tinta');
plt.subplot(1,3,2); plt.imshow(S,clim=[0,1],cmap='gray');
plt.title('Satureazione');
plt.subplot(1,3,3); plt.imshow(I,clim=[0,1],cmap='gray');
plt.title('IntensitÃ ');

mask = (0.35<H) & (H<0.66) & (0.20<S) & (S<0.80) & (I>0.35)
H = H + 0.38*mask
H = H % 1.0 # funzione modulo per riportare H nel range [0,1]
y[:,:,0] = H
z = hsi2rgb(y)

plt.figure(3);
plt.subplot(1,3,1);
plt.imshow(x); plt.title('Immagine originale');
plt.subplot(1,3,2);
plt.imshow(mask,clim=[0,1],cmap='gray'); plt.title('Selezione');
plt.subplot(1,3,3);
plt.imshow(z); plt.title('Risultato');
```

2. *Filtraggio in frequenza.*

```
x = io.imread('foto_originale.tif')
x = np.float64(x)/255
plt.figure(1); plt.imshow(x);
plt.title('Immagine originale');
```

```
# filtraggio nello spazio HSI
w = rgb2hsi(x)
I = w[:,:,2]

X = np.fft.fftshift(np.fft.fft2(I))
m = np.fft.fftshift(np.fft.fftfreq(X.shape[0]))
n = np.fft.fftshift(np.fft.fftfreq(X.shape[1]))
l,k = np.meshgrid(n,m)

H = (np.abs(l)<=0.10) & (np.abs(k)<=0.25);
Y = H * X
y = np.real(np.fft.ifft2(np.fft.ifftshift(Y)))
w[:,:,2] = y
z = hsi2rgb(w)

plt.figure(2); plt.imshow(z);
plt.title('Immagine filtrata');
```

3 Tecniche class-based

1. *Thresholding locale.*

```
x = np.float64(io.imread('yeast.tif'))
plt.figure(1); plt.imshow(x,clim=[0,255],cmap='gray');
plt.title('immagine originale');

# thresholding adattativo
def thresholding_locale(x):
    m_g = np.mean(x)
    s_l = ndi.generic_filter(x, np.std, (3,3))
    mask = (x>30*s_l) & (x>1.5*m_g)
    return mask

mapp = thresholding_locale(x)
plt.figure(2); plt.imshow(mapp,clim=[0,1],cmap='gray');
plt.title('thresholding adattativo');
```