

ProgettoFinale

December 18, 2022

0.1 Titolo: Un mondo monocromatico

0.2 Autore: Andrea Stucchi

0.3 Collaborazioni: Riccardo Isola, solamente parzialmente per il filtro 1

0.4 Obiettivi:

- Replicare il filtro che, data un'immagine a colori e un range di colore, mette in risalto i colori all'interno di questo range, trasformando in scala di grigi tutto il resto;
- Realizzare un filtro simile a quello descritto sopra, ma che, dato un range di colore, trasforma i colori all'interno di questo range in scala di grigi e lascia invariato tutto il resto;
- Realizzare un filtro che, data un'immagine a colori e un range di colore, trasforma i colori all'interno di questo range in scala di grigi e ri-mappa all'interno del range selezionato tutti gli altri.

0.5 Filtro 1

Data un'immagine a colori e un range di colore, vengono messi in risalto i colori all'interno di questo range, trasformando in scala di grigi tutto il resto.

0.5.1 Descrizione metodi

gray_Scale(hsv): Funzione che trasforma un'immagine hsv in scala di grigi

Input

- **hsv:** immagine in formato hsv

Funzionamento

1. : Viene azzerata la saturazione dell'immagine

Output

L'immagine in scala di grigi

createMaskInRange(hsv, lowerBound, upperBound): Funzione che crea una maschera di selezione dei colori di un'immagine hsv all'interno di un range

Input

- **hsv:** immagine in formato hsv

- **lowerBound**: limite inferiore del range di colori target
- **upperBound**: limite superiore del range di colori target

Funzionamento

1. : La maschera viene realizzata facendo l'And tra i valori dell'immagine $\geq \text{lower_bound}$ e $\leq \text{upper_bound}$.
2. : Al risultato dell'operazione sopra viene applicata la funzione `np.all()`, sul terzo asse, per ottenere una matrice di booleani che indica quali pixel sono all'interno del range.

Output

La maschera

createInvMask(maskToInv): Funzione che inverte una maschera

Input

- **maskToInv**: maschera da invertire

Funzionamento

1. : La maschera inversa viene creata facendo il Not di `maskToInv`

Output

La maschera inversa

applyMask(img, mask): Funzione che applica una maschera booleana in 2 dimensioni ad un'immagine

Input

- **img**: immagine su cui applicare la maschera
- **mask**: maschera da applicare

Funzionamento

1. : Siccome la maschera è booleana ma l'immagine è di interi, viene convertita in interi ad 8 bit senza segno
2. : Siccome la maschera è bidimensionale ma l'immagine è a 3 dimensioni, tramite la funzione `np.repeat()` la maschera viene ripetuta 3 volte sul secondo asse.
3. : La maschera viene resa tridimensionale, tramite la funzione `np.reshape()`
4. : L'immagine viene finalmente moltiplicata per la maschera

Output

L'immagine con la maschera applicata

applyFilterToImg(img, lowerBound, upperBound): Funzione che applica il filtro ad un'immagine rgb

Input

- **img**: immagine a cui applicare il filtro

- **lowerBound**: limite inferiore del range di colori target
- **upperBound**: limite superiore del range di colori target

Funzionamento

1. : L'immagine viene convertita in hsv
2. : Viene realizzata la corrispondente immagine in scala di grigi
3. : Viene creata la maschera di selezione dei colori all'interno dei due bounds
4. : Viene creata la maschera inversa
5. : Viene applicata la maschera all'immagine in hsv (foreground)
6. : Viene applicata la maschera inversa all'immagine in scala di grigi (background)
7. : Viene fatta la somma tra il background e il foreground, convertendo il risultato in rgb.

Output

L'immagine in rgb col filtro applicato

Funzioni utilizzate per la visualizzazione delle immagini

convertTo255(img): Funzione che converte un'immagine di float compresi tra 0 e 1 in un'immagine di interi compresi tra 0 e 255. Viene utilizzata per mostrare le immagini filtrate.

Input

- **img**: immagine da convertire

Funzionamento

1. : L'immagine viene moltiplicata per 255
2. : L'immagine viene convertita in interi senza segno a 8 bit

Output

L'immagine convertita

showImgs(imgs, labels): Funzione che mostra all'interno dello stesso plot le immagini passate come parametro con le relative etichette

Input

- **imgs**: array di immagini da mostrare
- **labels**: lista di etichette da mostrare

Funzionamento

1. : Viene creato il plot con le immagini
2. : Vengono aggiunte le etichette
3. : Viene mostrato il plot

Output

Nessuno

0.5.2 Codice

```
[ ]: # import delle librerie
import numpy as np
import plotly.express as px
import skimage as ski
from skimage.color import rgb2hsv, hsv2rgb

# Funzione che converte un'immagine hsv in scala di grigi
def grayScale(hsv):
    grayScaleHsv = np.copy(hsv)

    # La saturazione viene messa a 0 per ottenere la scala di grigi
    grayScaleHsv[:, :, 1] = 0
    return grayScaleHsv

# Funzione che crea una maschera sull'immagine hsv per i colori compresi tra i due bounds
def createMaskInRange(hsv, lowerBound, upperBound):
    return np.all((hsv >= lowerBound) & (hsv <= upperBound), axis=2)

# Funzione che crea la maschera inversa della maschera passata come parametro
def createInvMask(maskToInv):
    return np.logical_not(maskToInv)

# Funzione che applica la maschera all'immagine
def applyMask(img, mask):
    # La maschera viene convertita in unsigned int a 8 bit
    mask = mask.astype('uint8')

    # La maschera viene ripetuta 3 volte e trasformata in 3 dimensioni, in modo da poterla applicare ai 3 canali dell'immagine
    mask = np.repeat(mask, 3, axis=1)
    mask = np.reshape(mask, img.shape)

    # La maschera viene applicata all'immagine e ritornata
    return img * mask

# Funzione che applica il filtro, basato sui bounds, all'immagine
def applyFilterToImg(img, lowerBound, upperBound):
    # L'immagine viene convertita da rgb in hsv
    hsvImg = rgb2hsv(img)

    # Viene creata la scala di grigi dell'immagine
    grayScaleImg = grayScale(hsvImg)
```

```

# Viene creata una maschera sull'immagine hsv per i colori compresi tra i due bounds
mask = createMaskInRange(hsvImg, lowerBound, upperBound)

# Viene creata la maschera inversa
invMask = createInvMask(mask)

# Viene creato il foreground applicando la maschera all'immagine originale
foreground = applyMask(hsvImg, mask)

# Viene creato il background applicando la maschera inversa alla scala di grigi
background = applyMask(grayScaleImg, invMask)

# L'immagine finale viene realizzata sommando il background al foreground
return hsv2rgb(background + foreground)

# Funzione che converte un'immagine di float compresi tra 0 e 1 in un'immagine di int compresi tra 0 e 255
def convertTo255(img):
    img = img * 255
    img = img.astype('uint8')
    return img

# Funzione che mostra all'interno dello stesso plot le immagini passare come parametro con le relative etichette
def showImgs(imgs, labels):
    fig = px.imshow(imgs, facet_col=0, binary_string=True)

    for i, label in enumerate(labels):
        fig.layout.annotations[i]['text'] = label

    fig.show()

```

0.5.3 Risultati

```

[ ]: import plotly.io as pio
pio.renderers.default = "notebook+pdf"

# Vengono lette le immagini sulle quali verranno applicati i filtri
rocket = ski.data.rocket()
jackets = ski.io.imread("jackets.jpeg")
parrot = ski.io.imread("parrot.jpg")
flowers = ski.io.imread("flowers.jpg")

# Vengono definiti i bounds per i vari colori
lowerBound_red = np.array([0.94, 0.39, 0.2])

```

```

upperBound_red = np.array([1,1,1])

lowerBound_blue = np.array([0.49,0,0])
upperBound_blue = np.array([0.71,1,1])

lowerBound_green = np.array([0.16,0,0])
upperBound_green = np.array([0.27,1,1])

lowerBound_orange = np.array([0.1,0,0])
upperBound_orange = np.array([0.2,1,1])

# Viene applicato il filtro alle immagini
finalJackets = applyFilterToImg(jackets, lowerBound_red, upperBound_red)
finalFlowers = applyFilterToImg(flowers, lowerBound_orange, upperBound_orange)
finalParrot = applyFilterToImg(parrot, lowerBound_green, upperBound_green)
finalRocket = applyFilterToImg(rocket, lowerBound_blue, upperBound_blue)

# Vengono mostrate le immagini
# Notare che le immagini ottenute con l'applicazione del filtro sono state
↳ convertite in interi senza segno ad 8 bit in range 0-255
# così da essere dello stesso tipo dell'immagine originale e poterle quindi
↳ mostrare con px.imshow
labels = ['Immagine originale', 'Immagine filtrata']
showImgs(np.array([jackets, convertTo255(finalJackets)]), labels)
showImgs(np.array([rocket, convertTo255(finalRocket)]), labels)
showImgs(np.array([flowers, convertTo255(finalFlowers)]), labels)
showImgs(np.array([parrot, convertTo255(finalParrot)]), labels)

```



Immagine originale



Immagine filtrata



Immagine originale



Immagine filtrata



0.5.4 Commenti ai risultati

Soglie colori Le soglie dei colori sono ovviamente state scelte in base al colore che si voleva evidenziare; prendendo soglie differenti, i colori catturati dalla maschera sarebbero stati chiaramente diversi, e quindi il risultato ottenuto non sarebbe stato come quelli sopra. Notare però che con soglie differenti il risultato ottenuto avrebbe comunque rispettato le specifiche del filtro, ma l'immagine finale non sarebbe stata piacevole alla vista come quelle mostrate sopra.

Motivazione di alcune scelte implementative

- Siccome le immagini in hsv risultano essere array a 3 dimensioni con valori compresi tra 0 e 1, le soglie utilizzate sono anch'esse comprese tra 0 e 1. Inizialmente questa non è stata la decisione presa, ma erano state invece utilizzate soglie con valori compresi tra 0 e 255: si rese quindi necessario convertire subito l'immagine hsv in interi senza segno compresi appunto tra 0 e 255 di modo da poterle utilizzare. Infine, foreground e background venivano convertiti in rgb, ma prima della somma per ottenere l'immagine finale, fu necessario convertire in scala 0-255 il background, poiché altrimenti l'immagine risultante dalla somma sarebbe stata caratterizzata da uno sfondo completamente nero; questo avveniva poichè il background è in scala di grigi, quando veniva convertito in rgb risultava in un'immagine a 3 canali, ma con valori compresi tra 0 e 1, mentre il foreground a colori aveva valori compresi tra 0 e 255: i valori del background venivano infatti interpretati come nero rgb.

Di seguito il risultato:

```
[ ]: # Funzione che applica il filtro, basato sui bounds, all'immagine
def applyFilterToImg(img, lowerBound, upperBound):
    # L'immagine viene convertita da rgb in hsv
    hsvImg = rgb2hsv(img)

    # L'immagine hsv viene convertita in interi compresi tra 0 e 255
    hsvImg = convertTo255(hsvImg)

    # Viene creata la scala di grigi dell'immagine
    grayScaleImg = hsv2rgb(grayScale(hsvImg))

    # Viene creata una maschera sull'immagine hsv per i colori compresi tra i due bounds
    mask = createMaskInRange(hsvImg, lowerBound, upperBound)

    # Viene creata la maschera inversa
    invMask = createInvMask(mask)

    # Viene creato il foreground applicando la maschera all'immagine originale
    foreground = applyMask(img, mask)

    # Viene creato il background applicando la maschera inversa alla scala di grigi
    background = applyMask(grayScaleImg, invMask)

    # L'immagine finale viene realizzata sommando il background al foreground
    return background + foreground

# Vengono definiti i bounds per i vari colori
lowerBound_red = np.array([240,100,50])
upperBound_red = np.array([255,255,255])

lowerBound_blue = np.array([125,0,0])
upperBound_blue = np.array([180,255,255])

lowerBound_green = np.array([40,0,0])
upperBound_green = np.array([75,255,255])

lowerBound_orange = np.array([25,0,0])
upperBound_orange = np.array([50,255,255])

# Viene applicato il filtro alle immagini
finalJackets = applyFilterToImg(jackets, lowerBound_red, upperBound_red)
finalFlowers = applyFilterToImg(flowers, lowerBound_orange, upperBound_orange)
finalParrot = applyFilterToImg(parrot, lowerBound_green, upperBound_green)
finalRocket = applyFilterToImg(rocket, lowerBound_blue, upperBound_blue)
```

```
# Vengono mostrate le immagini
labels = ['Immagine originale', 'Immagine filtrata']
showImgs(np.array([jackets, finalJackets]), labels)
showImgs(np.array([rocket, finalRocket]), labels)
showImgs(np.array([flowers, finalFlowers]), labels)
showImgs(np.array([parrot, finalParrot]), labels)
```



Immagine originale

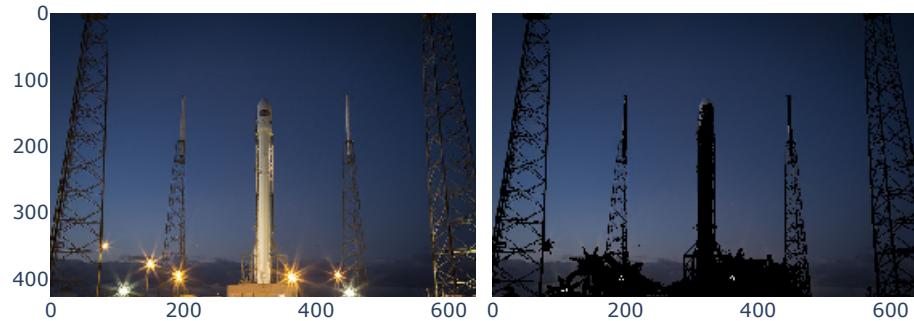
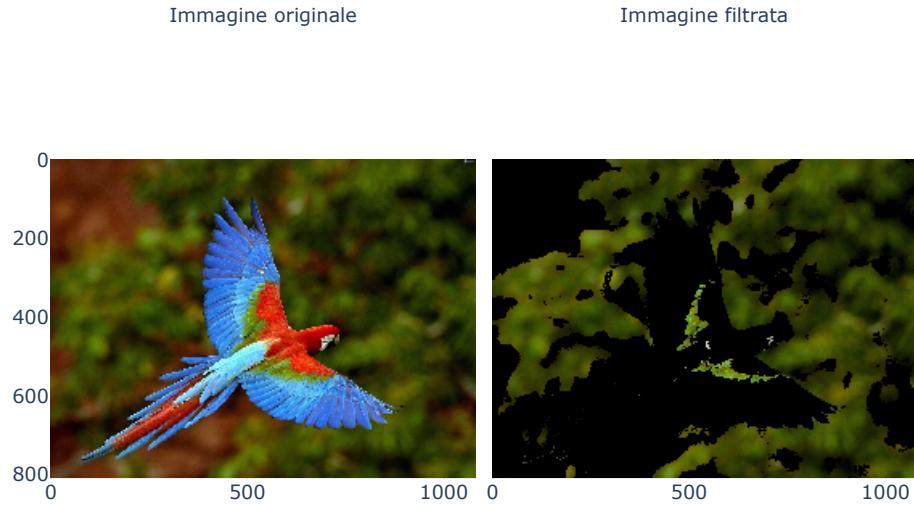


Immagine filtrata





0.6 Filtro 2:

Data un’immagine e un range di colore, trasforma i colori all’interno di questo range in scala di grigi e lascia invariato tutto il resto;

0.6.1 Descrizione metodi

gray_Scale(hsv): Funzione che trasforma un’immagine hsv in scala di grigi

Input

- **hsv:** immagine in formato hsv

Funzionamento

1. : Viene azzerata la saturazione dell’immagine

Output

L’immagine in scala di grigi

createMaskInRange(hsv, lowerBound, upperBound): Funzione che crea una maschera di selezione dei colori di un’immagine hsv all’interno di un range

Input

- **hsv**: immagine in formato hsv
- **lowerBound**: limite inferiore del range di colori target
- **upperBound**: limite superiore del range di colori target

Funzionamento

1. : La maschera viene realizzata facendo l'And tra i valori dell'immagine $\geq \text{lower_bound}$ e $\leq \text{upper_bound}$.
2. : Al risultato dell'operazione sopra viene applicata la funzione `np.all()`, sul terzo asse, per ottenere una matrice di booleani che indica quali pixel sono all'interno del range.

Output

La maschera

createInvMask(maskToInv): Funzione che inverte una maschera

Input

- **maskToInv**: maschera da invertire

Funzionamento

1. : La maschera inversa viene creata facendo il Not di `maskToInv`

Output

La maschera inversa

applyMask(img, mask): Funzione che applica una maschera booleana in 2 dimensioni ad un'immagine

Input

- **img**: immagine su cui applicare la maschera
- **mask**: maschera da applicare

Funzionamento

1. : Siccome la maschera è booleana ma l'immagine è di interi, viene convertita in interi ad 8 bit senza segno
2. : Siccome la maschera è bidimensionale ma l'immagine è a 3 dimensioni, tramite la funzione `np.repeat()` la maschera viene ripetuta 3 volte sul secondo asse.
3. : La maschera viene resa tridimensionale, tramite la funzione `np.reshape()`
4. : L'immagine viene finalmente moltiplicata per la maschera

Output

L'immagine con la maschera applicata

applyFilterToImg(img, lowerBound, upperBound): Funzione che applica il filtro ad un’immagine rgb

Input

- **img:** immagine a cui applicare il filtro
- **lowerBound:** limite inferiore del range di colori target
- **upperBound:** limite superiore del range di colori target

Funzionamento

1. : L’immagine viene convertita in hsv
2. : Viene realizzata la corrispondente immagine in scala di grigi
3. : Viene creata la maschera di selezione dei colori all’interno dei due bounds
4. : Viene creata la maschera inversa
5. : Viene applicata la maschera all’immagine in scala di grigi (foreground)
6. : Viene applicata la maschera inversa all’immagine in hsv (background)
7. : Viene fatta la somma tra il background e il foreground, convertendo il risultato in rgb.

Output

L’immagine in rgb col filtro applicato

convertTo255(img): Funzione che converte un’immagine di float compresi tra 0 e 1 in un’immagine di interi compresi tra 0 e 255. Viene utilizzata per mostrare le immagini filtrate.

Input

- **img:** immagine da convertire

Funzionamento

1. : L’immagine viene moltiplicata per 255
2. : L’immagine viene convertita in interi senza segno a 8 bit

Output

L’immagine convertita

Funzioni utilizzate per la visualizzazione delle immagini

convertTo255(img): Funzione che converte un’immagine di float compresi tra 0 e 1 in un’immagine di interi compresi tra 0 e 255. Viene utilizzata per mostrare le immagini filtrate.

Input

- **img:** immagine da convertire

Funzionamento

1. : L’immagine viene moltiplicata per 255
2. : L’immagine viene convertita in interi senza segno a 8 bit

Output L'immagine convertita

showImgs(imgs, labels): Funzione che mostra all'interno dello stesso plot le immagini passate come parametro con le relative etichette

Input

- **imgs:** array di immagini da mostrare
- **labels:** lista di etichette da mostrare

Funzionamento

1. : Viene creato il plot con le immagini
2. : Vengono aggiunte le etichette
3. : Viene mostrato il plot

Output Nessuno

0.6.2 Codice

```
[ ]: # import delle librerie
import numpy as np
import plotly.express as px
import skimage as ski
from skimage.color import rgb2hsv, hsv2rgb

# Funzione che converte un'immagine hsv in scala di grigi
def grayScale(hsv):
    grayScaleHsv = np.copy(hsv)

    # La saturazione viene messa a 0 per ottenere la scala di grigi
    grayScaleHsv[:, :, 1] = 0
    return grayScaleHsv

# Funzione che crea una maschera sull'immagine hsv per i colori compresi tra i due bounds
def createMaskInRange(hsv, lowerBound, upperBound):
    return np.all((hsv >= lowerBound) & (hsv <= upperBound), axis=2)

# Funzione che crea la maschera inversa della maschera passata come parametro
def createInvMask(maskToInv):
    return np.logical_not(maskToInv)

# Funzione che applica la maschera all'immagine
def applyMask(img, mask):
    # La maschera viene convertita in unsigned int a 8 bit
    mask = mask.astype('uint8')
```

```

# La maschera viene ripetuta 3 volte e trasformata in 3 dimensioni, in modo da poterla applicare ai 3 canali dell'immagine
mask = np.repeat(mask, 3, axis=1)
mask = np.reshape(mask, img.shape)

# La maschera viene applicata all'immagine e ritornata
return img * mask

# Funzione che applica il filtro, basato sui bounds, all'immagine
def applyFilterToImg(img, lowerBound, upperBound):
    # L'immagine viene convertita da rgb in hsv
    hsvImg = rgb2hsv(img)

    # Viene creata la scala di grigi dell'immagine
    grayScaleImg = grayScale(hsvImg)

    # Viene creata una maschera sull'immagine hsv per i colori compresi tra i due bounds
    mask = createMaskInRange(hsvImg, lowerBound, upperBound)

    # Viene creata la maschera inversa
    invMask = createInvMask(mask)

    # Viene creato il foreground applicando la maschera all'immagine originale
    foreground = applyMask(grayScaleImg, mask)

    # Viene creato il background applicando la maschera inversa alla scala di grigi
    background = applyMask(hsvImg, invMask)

    # L'immagine finale viene realizzata sommando il background al foreground
    return hsv2rgb(background + foreground)

# Funzione che converte un'immagine di float compresi tra 0 e 1 in un'immagine di int compresi tra 0 e 255
def convertTo255(img):
    img = img * 255
    img = img.astype('uint8')
    return img

# Funzione che mostra all'interno dello stesso plot le immagini passare come parametro con le relative etichette
def showImgs(imgs, labels):
    fig = px.imshow(imgs, facet_col=0, binary_string=True)

    for i, label in enumerate(labels):

```

```

fig.layout.annotations[i]['text'] = label

fig.show()

```

0.6.3 Risultati

```

[ ]: # Vengono lette le immagini sulle quali verranno applicati i filtri
rocket = ski.data.rocket()
jackets = ski.io.imread("jackets.jpeg")
parrot = ski.io.imread("parrot.jpg")
flowers = ski.io.imread("flowers.jpg")

# Vengono definiti i bounds per i vari colori
lowerBound_red = np.array([0.94,0.39,0.2])
upperBound_red = np.array([1,1,1])

lowerBound_blue = np.array([0.49,0,0])
upperBound_blue = np.array([0.71,1,1])

lowerBound_green = np.array([0.16,0,0])
upperBound_green = np.array([0.27,1,1])

lowerBound_orange = np.array([0.1,0,0])
upperBound_orange = np.array([0.2,1,1])

# Viene applicato il filtro alle immagini
finalJackets = applyFilterToImg(jackets, lowerBound_red, upperBound_red)
finalFlowers = applyFilterToImg(flowers, lowerBound_orange, upperBound_orange)
finalParrot = applyFilterToImg(parrot, lowerBound_green, upperBound_green)
finalRocket = applyFilterToImg(rocket, lowerBound_blue, upperBound_blue)

# Vengono mostrate le immagini
# Notare che le immagini ottenute con l'applicazione del filtro sono state ↴ convertite in interi senza segno ad 8 bit in range 0-255
# così da essere dello stesso tipo dell'immagine originale e poterle quindi ↴ mostrare con px.imshow
labels = ['Immagine originale', 'Immagine filtrata']
showImgs(np.array([jackets, convertTo255(finalJackets)]), labels)
showImgs(np.array([rocket, convertTo255(finalRocket)]), labels)
showImgs(np.array([flowers, convertTo255(finalFlowers)]), labels)
showImgs(np.array([parrot, convertTo255(finalParrot)]), labels)

```

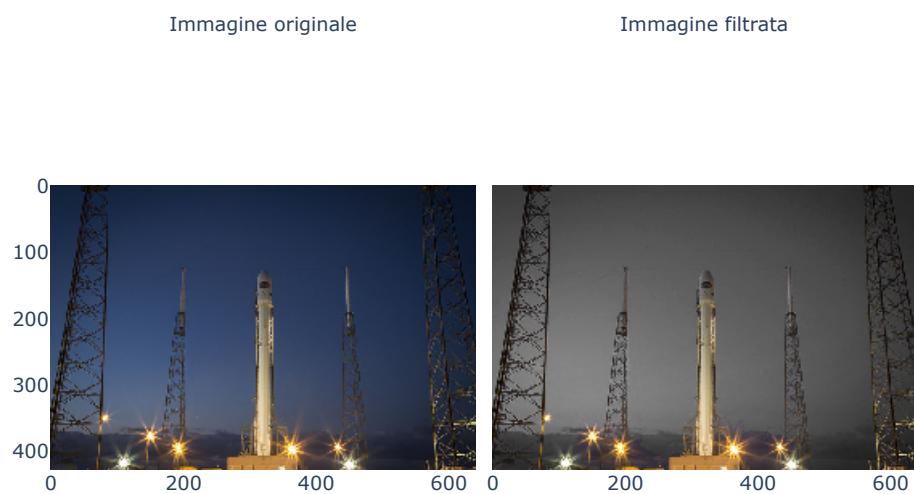
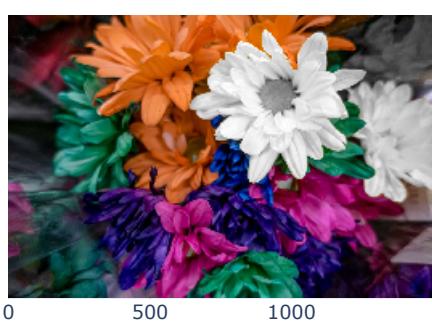
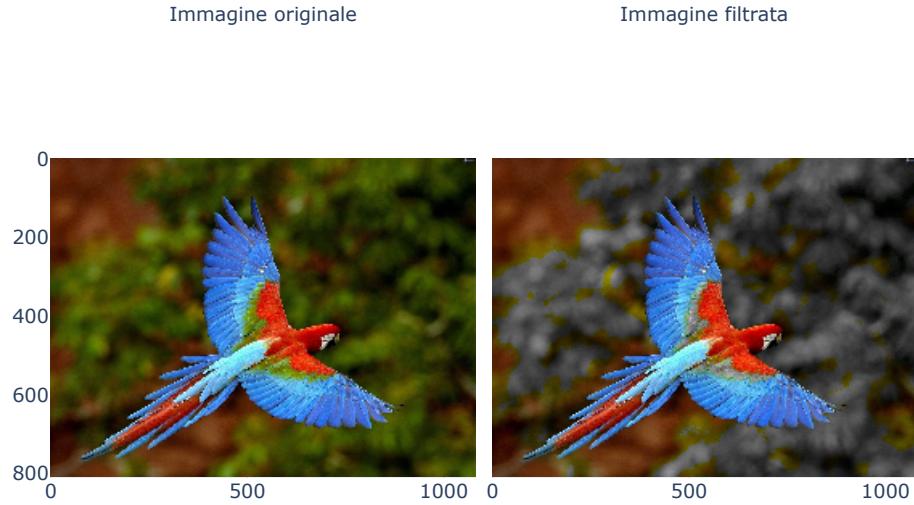


Immagine originale



Immagine filtrata





0.6.4 Commenti ai risultati

Soglie colori Le soglie dei colori sono ovviamente state scelte in base al colore che si voleva evidenziare; prendendo soglie differenti, i colori catturati dalla maschera sarebbero stati chiaramente diversi, e quindi il risultato ottenuto non sarebbe stato come quelli sopra. Ovviamente con soglie diverse il risultato ottenuto avrebbe comunque rispettato le specifiche del filtro, ma l'immagine finale non sarebbe stata piacevole alla vista come quelle ottenute.

Motivazione di alcune scelte implementative

- Siccome le immagini in hsv risultano essere array a 3 dimensioni con valori compresi tra 0 e 1, le soglie utilizzate sono anch'esse comprese tra 0 e 1. Inizialmente questa non è stata la decisione presa, ma erano state invece utilizzate soglie con valori compresi tra 0 e 255: si rese quindi necessario convertire subito l'immagine hsv in interi senza segno compresi appunto tra 0 e 255 di modo da poterle utilizzare. Infine, foreground e background venivano convertiti in rgb, ma prima della somma per ottenere l'immagine finale, fu necessario convertire in scala 0-255 il foreground, poiché altrimenti l'immagine risultante dalla somma sarebbe stata caratterizzata da zone completamente nere invece che in scala di grigi; questo avveniva poiché il foreground è in scala di grigi, quando veniva convertito in rgb risultava in un'immagine a 3 canali, ma con valori compresi tra 0 e 1, mentre il background a colori aveva valori compresi tra 0 e 255: i valori del foreground venivano infatti interpretati come nero rgb.

Di seguito il risultato:

```
[ ]: # Funzione che applica il filtro, basato sui bounds, all'immagine
def applyFilterToImg(img, lowerBound, upperBound):
    # L'immagine viene convertita da rgb in hsv
    hsvImg = rgb2hsv(img)

    # L'immagine hsv viene convertita in interi compresi tra 0 e 255
    hsvImg = convertTo255(hsvImg)

    # Viene creata la scala di grigi dell'immagine
    grayScaleImg = hsv2rgb(grayScale(hsvImg))

    # Viene creata una maschera sull'immagine hsv per i colori compresi tra i due bounds
    mask = createMaskInRange(hsvImg, lowerBound, upperBound)

    # Viene creata la maschera inversa
    invMask = createInvMask(mask)

    # Viene creato il foreground applicando la maschera all'immagine originale
    foreground = applyMask(grayScaleImg, mask)

    # Viene creato il background applicando la maschera inversa alla scala di grigi
    background = applyMask(img, invMask)

    # L'immagine finale viene realizzata sommando il background al foreground
    return background + foreground

# Vengono definiti i bounds per i vari colori
lowerBound_red = np.array([240,100,50])
upperBound_red = np.array([255,255,255])

lowerBound_blue = np.array([125,0,0])
upperBound_blue = np.array([180,255,255])

lowerBound_green = np.array([40,0,0])
upperBound_green = np.array([75,255,255])

lowerBound_orange = np.array([25,0,0])
upperBound_orange = np.array([50,255,255])

# Viene applicato il filtro alle immagini
finalJackets = applyFilterToImg(jackets, lowerBound_red, upperBound_red)
finalFlowers = applyFilterToImg(flowers, lowerBound_orange, upperBound_orange)
finalParrot = applyFilterToImg(parrot, lowerBound_green, upperBound_green)
finalRocket = applyFilterToImg(rocket, lowerBound_blue, upperBound_blue)
```

```
# Vengono mostrate le immagini  
labels = ['Immagine originale', 'Immagine filtrata']  
showImgs(np.array([jackets, finalJackets]), labels)  
showImgs(np.array([rocket, finalRocket]), labels)  
showImgs(np.array([flowers, finalFlowers]), labels)  
showImgs(np.array([parrot, finalParrot]), labels)
```



Immagine originale

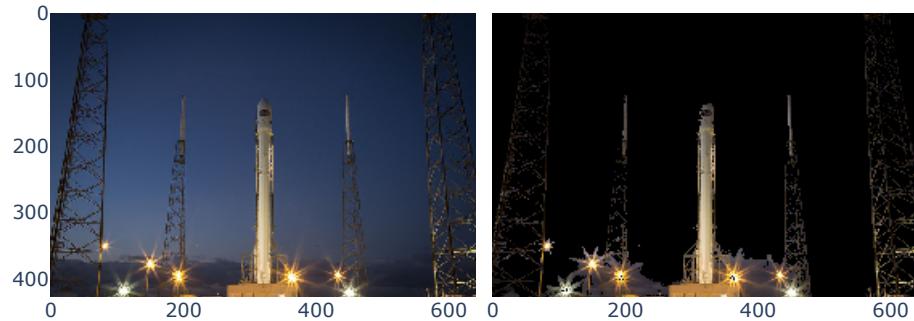


Immagine filtrata

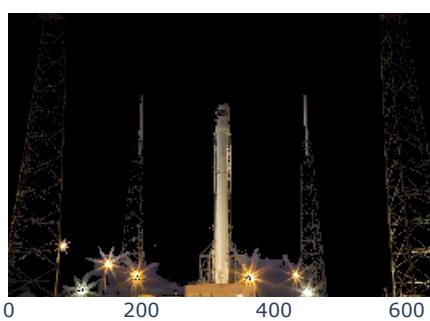


Immagine originale

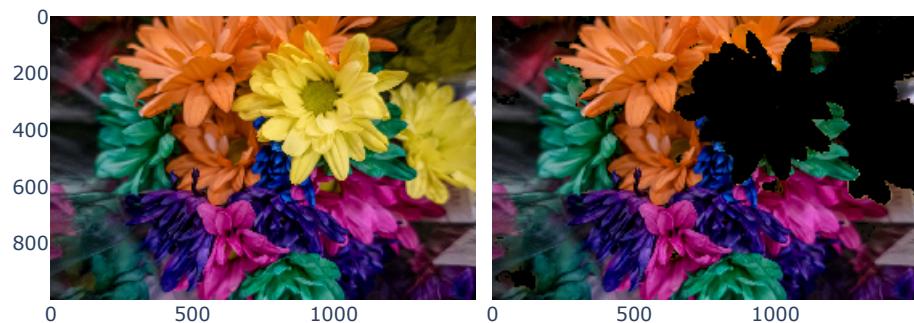
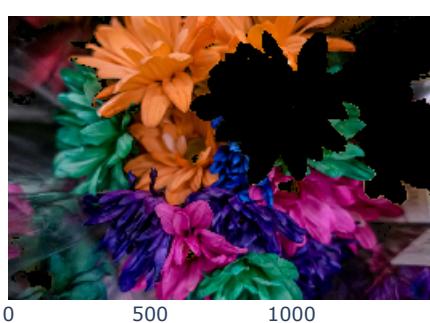
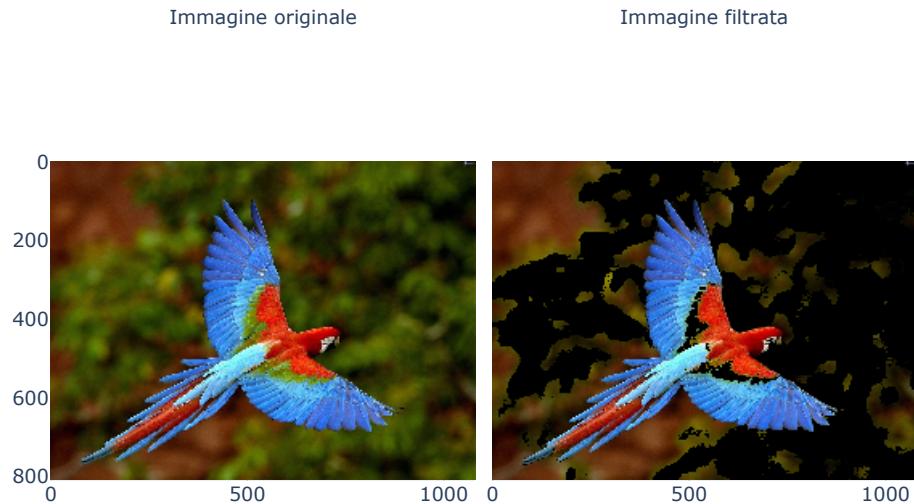


Immagine filtrata





0.7 Filtro 3:

Data un’immagine a colori e un range di colore, trasforma i colori all’interno di questo range in scala di grigi e ri-mappa all’interno del range selezionato tutti gli altri colori.

0.7.1 Descrizione metodi

grayScale(hsv): Funzione che trasforma un’immagine hsv in scala di grigi

Input

- **hsv:** immagine in formato hsv

Funzionamento

1. : Viene azzerata la saturazione dell’immagine

Output

L’immagine in scala di grigi

componentCalculation(lowerBound, upperBound, span, originalValues, isHue): Funzione che calcola il nuovo valore di una componente dell’immagine hsv

Input

- **lowerBound**: limite inferiore del range di colori della scala monocromatica
- **upperBound**: limite superiore del range di colori della scala monocromatica
- **span**: span del nuovo range di valori
- **originalValues**: valori originali della componente dell'immagine hsv
- **isHue**: booleano che indica se la componente è la hue o meno

Funzionamento

1. : Viene calcolato il peso di un'unità nella nuova scala (corrispondente a *span*), così da poter mappare i valori originali (in range 0-1) all'interno del nuovo range (ridotto)
2. : Se la componente è la hue, la scala originale è circolare, di conseguenza bisogna prendere in considerazione il fatto che 0 e 1 sono due valori adiacenti. Perciò, la scala originale viene divisa in due sezioni: i valori della prima (0-0.5) vengono mappati nel nuovo range dal lowerBound all'upperBound, i valori della seconda (0.5-1) invece vengono mappati dall'upperBound al lowerBound.
3. : Se la componente non è la hue, i nuovi valori vengono tutti mappati dal lowerBound all'upperBound.

Output

I nuovi valori della componente dell'immagine hsv

calcSpan(lowerVal, upperVal): Funzione che calcola lo span tra due valori

Input

- **lowerVal**: limite inferiore del range
- **upperVal**: limite superiore del range

Funzionamento

1. : Viene fatta la differenza tra upperVal e lowerVal

Output

Lo span tra i due valori

monocromatic_scale(hsv, lowerBound, upperBound): Funzione che converte un'immagine hsv in scala monocromatica

Input

- **hsv**: immagine in formato hsv
- **lowerBound**: limite inferiore del range di colori della scala monocromatica
- **upperBound**: limite superiore del range di colori della scala monocromatica

Funzionamento

1. : Vengono calcolati i bounds di hue, saturation e value dai due bounds forniti

2. : Viene calcolato lo span di hue, saturation e value tramite la funzione *calcSpan*
3. : Vengono calcolati i nuovi valori di hue, saturation e value tramite la funzione *componentCalculation*

Output

L'immagine monocromatica

createMaskInRange(hsv, lowerBound, upperBound): Funzione che crea una maschera di selezione dei colori di un'immagine hsv all'interno di un range

Input

- **hsv:** immagine in formato hsv
- **lowerBound:** limite inferiore del range di colori target
- **upperBound:** limite superiore del range di colori target

Funzionamento

1. : La maschera viene realizzata facendo l'And tra i valori dell'immagine $\geq \text{lowerBound}$ e $\leq \text{upperBound}$.
2. : Al risultato dell'operazione sopra viene applicata la funzione `np.all()`, sul terzo asse, per ottenere una matrice di booleani che indica quali pixel sono all'interno del range.

Output

La maschera

createInvMask(maskToInv): Funzione che inverte una maschera

Input

- **maskToInv:** maschera da invertire

Funzionamento

1. : La maschera inversa viene creata facendo il Not di *maskToInv*

Output

La maschera inversa

applyMask(img, mask): Funzione che applica una maschera booleana in 2 dimensioni ad un'immagine

Input

- **img:** immagine su cui applicare la maschera
- **mask:** maschera da applicare

Funzionamento

1. : Siccome la maschera è booleana ma l'immagine è di interi, viene convertita in interi ad 8 bit senza segno

2. : Siccome la maschera è bidimensionale ma l'immagine è a 3 dimensioni, tramite la funzione `np.repeat()` la maschera viene ripetuta 3 volte sul secondo asse.
3. : La maschera viene resa tridimensionale, tramite la funzione `np.reshape()`
4. : L'immagine viene finalmente moltiplicata per la maschera

Output

L'immagine con la maschera applicata

applyFilterToImg(img, lowerBound, upperBound): Funzione che applica il filtro monocromatico ad un'immagine rgb

Input

- **img:** immagine a cui applicare il filtro
- **lowerBound:** limite inferiore del range di colori della scala monocromatica
- **upperBound:** limite superiore del range di colori della scala monocromatica

Funzionamento

1. : L'immagine viene convertita in hsv
2. : Viene realizzata la corrispondente immagine in scala di grigi
3. : Viene realizzata la corrispondente immagine monocromatica nel range stabilito
4. : Viene creata la maschera di selezione dei colori all'interno dei due bounds
5. : Viene creata la maschera inversa
6. : Viene applicata la maschera all'immagine in scala di grigi (foreground)
7. : Viene applicata la maschera inversa all'immagine monocromatica (background)
8. : Viene fatta la somma tra il background e il foreground, convertendo il risultato in rgb.

Output

L'immagine in rgb col filtro applicato

convertTo255(img): Funzione che converte un'immagine di float compresi tra 0 e 1 in un'immagine di interi compresi tra 0 e 255. Viene utilizzata per mostrare le immagini filtrate.

Input

- **img:** immagine da convertire

Funzionamento

1. : L'immagine viene moltiplicata per 255
2. : L'immagine viene convertita in interi senza segno a 8 bit

Output

L'immagine convertita

Funzioni utilizzate per la visualizzazione delle immagini

convertTo255(img): Funzione che converte un'immagine di float compresi tra 0 e 1 in un'immagine di interi compresi tra 0 e 255. Viene utilizzata per mostrare le immagini filtrate.

Input

- **img**: immagine da convertire

Funzionamento

1. : L'immagine viene moltiplicata per 255
2. : L'immagine viene convertita in interi senza segno a 8 bit

Output L'immagine convertita

showImgs(imgs, labels): Funzione che mostra all'interno dello stesso plot le immagini passate come parametro con le relative etichette

Input

- **imgs**: array di immagini da mostrare
- **labels**: lista di etichette da mostrare

Funzionamento

1. : Viene creato il plot con le immagini
2. : Vengono aggiunte le etichette
3. : Viene mostrato il plot

Output Nessuno

0.7.2 Codice

```
[ ]: # import delle librerie
import numpy as np
import plotly.express as px
import skimage as ski
from skimage.color import rgb2hsv, hsv2rgb

# Funzione che converte un'immagine hsv in scala di grigi
def grayScale(hsv):
    grayScaleHsv = np.copy(hsv)

    # La saturazione viene messa a 0 per ottenere la scala di grigi
    grayScaleHsv[:, :, 1] = 0
    return grayScaleHsv

# Funzione che calcola i nuovi valori di una componente dell'immagine hsv, dati i bounds inferiori e superiori della nuova scala,
# lo span tra questi, i valori originali della componente e se la componente è la hue o meno
```

```

def componentCalculation(lowerBound, upperBound, span, originalValues, isHue = False):
    # Viene calcolato il peso di una unità sulla nuova scala, corrispondente allo span del nuovo range di mappatura.
    # Questo peso permetterà di mappare valori in scala 0-1 in valori in scala lowerBound-upperBound
    weight = span

    # Se la componente è la hue, la scala originale è circolare, quindi bisogna tenere conto del fatto che 0 e 1 sono adiacenti.
    # Quindi, la scala originale viene divisa in due parti, e i valori nella prima parte vengono mappati dal lower bound all'upper bound, la seconda parte viceversa
    if(isHue):
        # Il peso viene raddoppiata, poichè la scala è stata divisa in due sezioni che verranno considerate singolarmente
        weight *= 2

        # Se il valore originale è <= 0.5, viene mappato dal lower bound all'upper bound sommando weight * originalValues al lowerBound.
        # Notare che quindi weight * originalValues è un valore compreso tra 0 e span.
        # Altrimenti viene mappato dall'upper bound al lower bound sottraendo (weight * originalValues - span) all'upperBound.
        # Notare che quindi weight * originalValues è un valore maggiore di span, quindi (weight * originalValues - span) è un valore compreso tra 0 e span
        newVals = np.where(originalValues <= 0.5, lowerBound + weight * originalValues, upperBound - (weight * originalValues - span))

    else:
        # Se la componente non è la hue, la scala originale è lineare, quindi i valori vengono tutti mappati dal lower bound all'upper bound
        newVals = lowerBound + weight * originalValues

    return newVals

# Funzione che calcola lo span tra i due valori passati come parametri
def calcSpan(lowerVal, upperVal):
    return upperVal - lowerVal

# Funzione che converte un'immagine hsv in scala monocromatica del colore compreso tra i due bounds
def monochromaticScale(hsv, lowerBound, upperBound):
    hsvMonochromatic = np.copy(hsv)

    # Vengono presi i bounds superiori e inferiori dei tre canali

```

```

upperHue, upperSaturation, upperValue = upperBound
lowerHue, lowerSaturation, lowerValue = lowerBound

# Viene preso lo span dei tre canali
hueSpan, saturationSpan, valueSpan = calcSpan(lowerHue, upperHue),  

    ↪calcSpan(lowerSaturation, upperSaturation), calcSpan(lowerValue, upperValue)

# I tre canali dell'immagine vengono convertiti in scala monocromatica
hsvMonochromatic[:, :, 0] = componentCalculation(lowerHue, upperHue, hueSpan,  

    ↪hsvMonochromatic[:, :, 0], True)
hsvMonochromatic[:, :, 1] = componentCalculation(lowerSaturation,  

    ↪upperSaturation, saturationSpan, hsvMonochromatic[:, :, 1])
hsvMonochromatic[:, :, 2] = componentCalculation(lowerValue, upperValue,  

    ↪valueSpan, hsvMonochromatic[:, :, 2])

return hsvMonochromatic

# Funzione che crea una maschera sull'immagine hsv per i colori compresi tra i due bounds
def createMaskInRange(hsv, lowerBound, upperBound):
    return np.all((hsv >= lowerBound) & (hsv <= upperBound), axis=2)

# Funzione che crea la maschera inversa della maschera passata come parametro
def createInvMask(maskToInv):
    return np.logical_not(maskToInv)

# Funzione che applica la maschera all'immagine
def applyMask(img, mask):
    # La maschera viene convertita in unsigned int a 8 bit
    mask = mask.astype('uint8')

    # La maschera viene ripetuta 3 volte e trasformata in 3 dimensioni, in modo da poterla applicare ai 3 canali dell'immagine
    mask = np.repeat(mask, 3, axis=1)
    mask = np.reshape(mask, img.shape)

    # La maschera viene applicata all'immagine e ritornata
    return img * mask

# Funzione che applica il filtro, basato sui bounds, all'immagine
def applyFilterToImg(img, lower_bound, upper_bound):
    # L'immagine viene convertita da rgb in hsv
    hsvImg = rgb2hsv(img)

    # Viene creata la scala di grigi dell'immagine
    grayScaleImg = grayScale(hsvImg)

```

```

# Viene creata l'immagine monocromatica tra i bounds specificati
monochromaticImg = monochromaticScale(hsvImg, lower_bound, upper_bound)

# Viene creata una maschera sull'immagine hsv per i colori compresi tra i due bounds
mask = createMaskInRange(hsvImg, lower_bound, upper_bound)

# Viene creata la maschera inversa
invMask = createInvMask(mask)

# Viene creato il foreground applicando la maschera all'immagine originale
foreground = applyMask(grayScaleImg, mask)

# Viene creato il background applicando la maschera inversa alla scala di grigi
background = applyMask(monochromaticImg, invMask)

# L'immagine finale viene realizzata sommando il background al foreground e convertendo il risultato in rgb
return hsv2rgb(background + foreground)

# Funzione che converte un'immagine di float compresi tra 0 e 1 in un'immagine di int compresi tra 0 e 255
def convertTo255(img):
    img = img * 255
    img = img.astype('uint8')
    return img

# Funzione che mostra all'interno dello stesso plot le immagini passare come parametro con le relative etichette
def showImgs(imgs, labels):
    fig = px.imshow(imgs, facet_col=0, binary_string=True)

    for i, label in enumerate(labels):
        fig.layout.annotations[i]['text'] = label

    fig.show()

```

0.7.3 Risultati

```
[ ]: # Vengono lette le immagini sulle quali verranno applicati i filtri
rocket = ski.data.rocket()
jackets = ski.io.imread("jackets.jpeg")
parrot = ski.io.imread("parrot.jpg")
flowers = ski.io.imread("flowers.jpg")
```

```

# Vengono definiti i bounds per i vari colori
lowerBound_red = np.array([0.94,0.39,0.2])
upperBound_red = np.array([1,1,1])

lowerBound_blue = np.array([0.49,0,0])
upperBound_blue = np.array([0.71,1,1])

lowerBound_green = np.array([0.16,0,0])
upperBound_green = np.array([0.27,1,1])

lowerBound_orange = np.array([0.1,0,0])
upperBound_orange = np.array([0.2,1,1])

# Viene applicato il filtro alle immagini
finalJackets = applyFilterToImg(jackets, lowerBound_red, upperBound_red)
finalFlowers = applyFilterToImg(flowers, lowerBound_orange, upperBound_orange)
finalParrot = applyFilterToImg(parrot, lowerBound_green, upperBound_green)
finalRocket = applyFilterToImg(rocket, lowerBound_blue, upperBound_blue)

# Vengono mostrate le immagini
# Notare che le immagini ottenute con l'applicazione del filtro sono state
    → convertite in interi senza segno ad 8 bit in range 0-255
# così da essere dello stesso tipo dell'immagine originale e poterle quindi
    → mostrare con px.imshow

labels = ['Immagine originale', 'Immagine filtrata']
showImgs(np.array([jackets, convertTo255(finalJackets)]), labels)
showImgs(np.array([rocket, convertTo255(finalRocket)]), labels)
showImgs(np.array([flowers, convertTo255(finalFlowers)]), labels)
showImgs(np.array([parrot, convertTo255(finalParrot)]), labels)

```

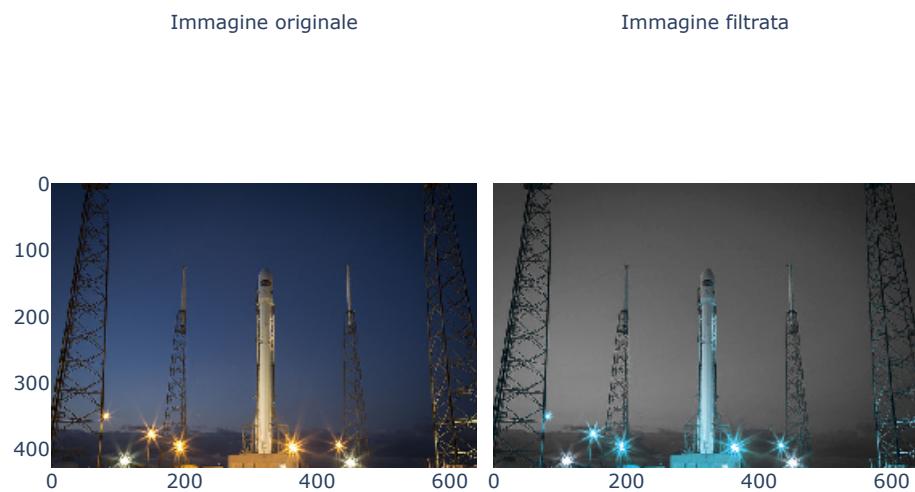


Immagine originale



Immagine filtrata





0.7.4 Commenti ai risultati

Soglie colori Le soglie dei colori sono ovviamente state scelte in base al colore che si voleva evidenziare; prendendo soglie differenti, i colori catturati dalla maschera sarebbero stati chiaramente diversi, e quindi il risultato ottenuto non sarebbe stato come quelli sopra. Ovviamente con soglie diverse il risultato ottenuto avrebbe comunque rispettato le specifiche del filtro, ma l'immagine finale non sarebbe stata piacevole alla vista come quelle ottenute.

Motivazione di alcune scelte implementative

- Nella funzione componentCalculation si è deciso di trattare nella maniera descritta nella sezione 3.1 la componente Hue di modo da mantenere l'armonia dei colori presente nell'immagine. Dato che la Hue è una componente circolare, mapparla in un nuovo range ristretto senza considerare questa sua proprietà ma considerandola bensì come lineare, porta ad avere delle discontinuità di colore nell'immagine finale dovute al fatto che i valori di rosso molto vicini a 0 e 1 sono adiacenti ma nell'immagine finale vengono mappati agli opposti del nuovo range. Di seguito i risultati assumendo la Hue come lineare:

```
[ ]: def componentCalculation(lowerBound, upperBound, weight, originalValues, isHue=False):
    ↵= False):
        return lowerBound + weight * originalValues
```

```

finalFlowers = applyFilterToImg(flowers, lowerBound_orange, upperBound_orange)
finalParrot = applyFilterToImg(parrot, lowerBound_green, upperBound_green)

# Vengono mostrate le immagini
# Notare che le immagini ottenute con l'applicazione del filtro sono state
↳ convertite in interi senza segno ad 8 bit in range 0-255
# così da essere dello stesso tipo dell'immagine originale e poterle quindi
↳ mostrare con px.imshow

labels = ['Immagine originale', 'Immagine filtrata']
showImgs(np.array([flowers, convertTo255(finalFlowers)]), labels)
showImgs(np.array([parrot, convertTo255(finalParrot)]), labels)

```

Immagine originale

Immagine filtrata



Immagine originale



Immagine filtrata

