Informatica LM-18

Relazione progetto Multimedia e Laboratorio

Rosario Leonardi W82000171

Prof. Filippo Stanco

Prof. Dario Allegra

1.Introduzione

Lo scopo di questo progetto è integrare al metodo *Line2D*, utilizzato per la detection di oggetti 3D textureless e descritto nel paper *"Gradient Response Maps for Real-Time Detection of Textureless Objects" [2]*, il pre-processo di normalizzazione dell'illuminazione presentato nel paper *"Enhanced Local Texture Feature Sets for Face Recognition Under Difficult Lighting Conditions" [1]* e descriverne i risultati ottenuti.

Inizialmente si implementerà il pre-processo utilizzando il linguaggio di programmazione python e si verificherà la corretta implementazione di esso paragonando i risultati ottenuti su un dataset composto da 30 immagini rispetto a quelli del paper di riferimento *[1]*. Infine si effettueranno degli esperimenti per testare l'efficacia del processo confrontando i risultati ottenuti da Line2D su un video di test prima e dopo aver applicato il pre-processo di normalizzazione.

2. Normalizzazione dell'illuminazione

In questa sezione verranno descritte le operazioni che costituiscono la fase di pre-processing.

2.1 Gamma Correction

La prima fase del pre-processo consiste nell'utilizzare la "Gamma Correction", ovvero una trasformazione non lineare del livello di intensità di grigio di un'immagine descritta dalla seguente legge:

$$I_{out} = I_{in}^{\gamma}$$

dove γ è un parametro positivo scelto dall'utente.

Il processo prende il nome di "gamma compression" se $\gamma < 1$ o "gamma expansion" se $\gamma > 1$.

Il grafico in Figura 2.1 mostra l'andamento della trasformazione al variare di γ.

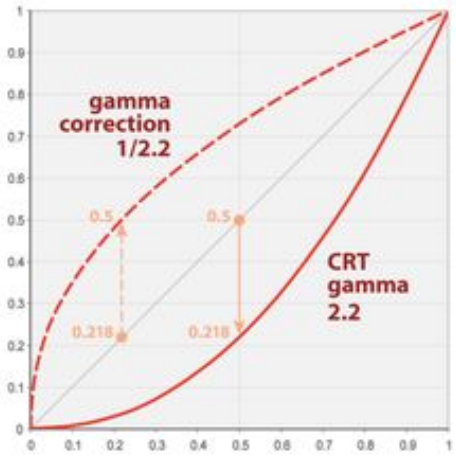


Figura 2.1

L'idea di base è sfruttare il processo di "gamma compression", che permette di espandere il range dinamico nelle zone scure dell'immagine e comprimerlo nelle zone luminose, per dare rilevanza alle zone "indipendenti" dall'illuminazione.

Nei vari esperimenti effettuati si è utilizzato un valore:

$$\gamma = 0.2$$

2.2 Difference of Gaussian (DoG) Filtering

Il secondo passo è l'utilizzo della "Difference of Gaussian" (DoG), tecnica che si basa sul calcolare la differenza tra due immagini a cui è stato applicato un filtro gaussiano con due diverse deviazioni standard, l'operatore viene utilizzato per il rilevamento dei contorni (ovvero la struttura) degli oggetti presenti nelle immagini (caratteristiche dei volti), inoltre tramite esso è possibile approssimare il comportamento di un filtro passa banda.

La scelta dei valori ottimali delle due deviazioni standard σ_0 , σ_1 influisce sui risultati ottenuti (ad esempio aumentando il valore di σ_0 è possibile ottenere dei bordi più netti), nei vari test sono stati individuati i seguenti valori per le deviazioni standard:

$$\sigma_0 = 1 \ , \ \sigma_1 = 2$$

Nella figura 2.2 sono rappresentante da sinistra verso destra:

1. Immagine originale;
2. Immagine a cui è stato applicato (dopo il passo di gamma correction preliminare) un filtro di smoothing gaussiano con $\sigma_0 = 1$;

3. Immagine a cui è stato applicato (dopo il passo di gamma correction preliminare) un filtro di smoothing gaussiano con $\sigma_1 = 2$;
4. La differenza tra le immagini 2 e 3 (DoG).

(a) (b) (c) (d)

Fig 2.2. (a) Immagine originale. (b) Filtro gaussiano con std=1.

(c) Filtro gaussiano con std=2. (d) DoG.

2.3 Contrast Equalization

L'ultimo step del pre-processo permette di scalare i valori di intensità di grigio associati ai pixel dell'immagine in modo da ottenere una misura robusta del contrasto generale o della variazione di intensità.

L'approccio utilizzato nel paper *[1]* consiste nell'aggiornare i valori dei pixel applicando i seguenti due passi:

$$I(x,y) = \frac{I(x,y)}{(mean(|I(x,y)|^{\alpha}))^{\frac{1}{\alpha}}} \qquad (1)$$

$$I(x,y) = \frac{I(x,y)}{(mean(min(\tau,|I(x,y)|)^{\alpha}))^{\frac{1}{\alpha}}} \qquad (2)$$

dove α è un parametro che permette di ridurre l'influenza dei valori "alti" e τ è una soglia utilizzata per rimuovere i valori "alti" ($> \tau$) presenti dopo aver applicato il primo passo.

Infine tramite l'utilizzo della formula:

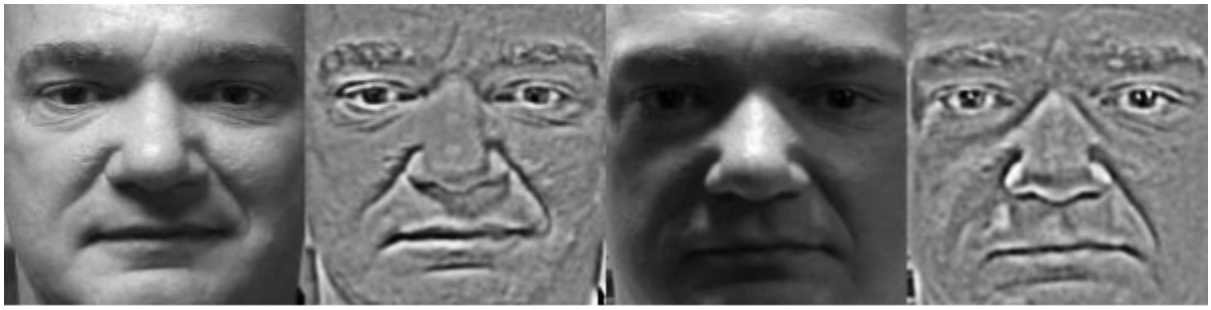
$$I(x,y) = \tau \tanh\left(\frac{I(x,y)}{\tau}\right)$$

è possibile ridurre il range dei valori a: $(-\tau, \tau)$.

Nei test effettuati sono stati utilizzati i seguenti valori:

$$\alpha = 0.1 \ , \ \tau = 10$$

Nella figura 2.3 è mostrato un confronto tra due immagini di uno stesso viso (sotto due diverse illuminazioni) prima e dopo il pre-processing.



(a) (b) (c) (d)

Figura 2.3. (a) Immagine originale 1. (b) Immagine 1 post pre-processo.
(c) Immagine originale 2. (d) Immagine 2 post pre-processo.

3. Local Binary Patterns

Il “Local Binary patterns” è un metodo robusto e veloce utilizzato per descrivere texture di oggetti in un'immagine.

La versione base dell’operatore assegna ad ogni pixel dell’immagine un valore calcolato utilizzando i suoi otto-vicini connessi. Nello specifico, sia p_c il valore del pixel centrale e siano p_i , $i\varepsilon\,[0,7]$ i valori degli 8 vicini connessi, il metodo assegna ad ogni pixel:

$$-1 \;\; se \;\; p_i > p_c$$

$$-0 \;\; se \;\; p_i \leq p_c$$

dopodiché combina i valori ottenuti (seguendo l’ordine mostrato nella Figura 3.1) e converte la stringa binaria risultante in decimale ottenendo il nuovo valore di p_c .

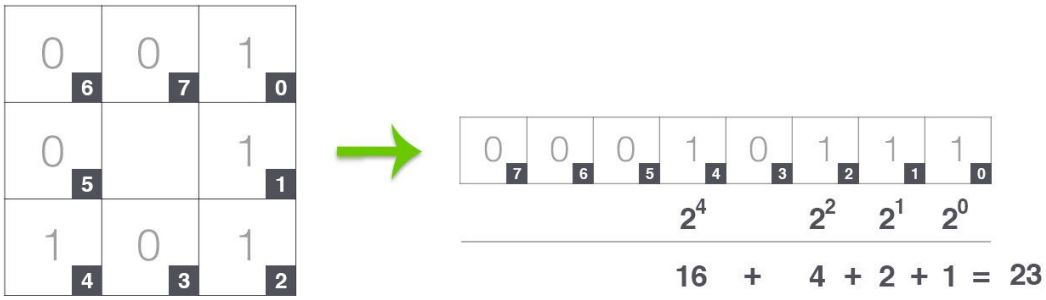


Figura 3.1

Un esempio del procedimento è mostrato in Figura 3.1 e in Figura 3.2.

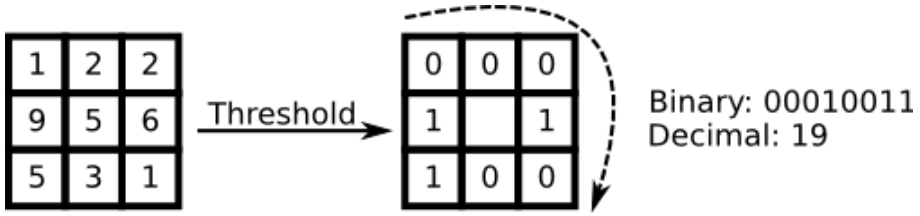


Figura 3.2

Un'estensione del metodo base è considerare solo pattern (stringhe binarie) uniformi, ovvero pattern che contengono al massimo due transizioni 0-1 o 1-0 (ad esempio: 11110011). Questa modifica riduce il range di valori da 255 a 59, ovvero 58 pattern uniformi + 1 che rappresenta tutti i pattern non uniformi eventualmente presenti. Dopo aver applicato questo metodo è possibile costruire un istogramma composto da 59 bins che descrive l'immagine e definire una metrica di similarità tra immagini basata sulla differenza di due istogrammi LBP.

Nella Figura 3.3 sono rappresentati da destra verso sinistra:

1. Istogramma LBP Figura 3.4a;
2. Istogramma LBP Figura 3.4b;
3. Intersezione tra i due istogrammi.

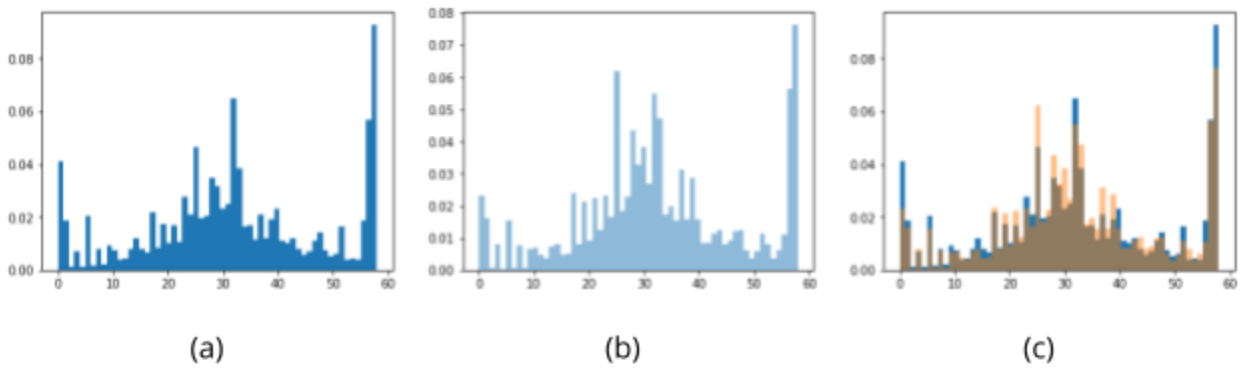


Figura 3.3. (a) LBP Figura 3.4a. (b) LBP Figura 3.4b. (c) Intersezione tra i due istogrammi.

(a) (b)

Figura 3.4

4. Esperimenti e risultati

Nei vari esperimenti condotti sono stati scelti i seguenti parametri:

Gamma correction	$\alpha = 0.2$
DoG	$\sigma_0 = 1$
DoG	$\sigma_1 = 2$
Contrast equalization	$\alpha = 0.1$
Contrast equalization	$\tau = 10$

Per calcolare le distanze tra i vari istogrammi è stata utilizzata la distanza del chi quadrato definita dalla seguente formula:

$$distance(p,q) = \sum_i \frac{(p_i - q_i)^2}{p_i + q_i}$$

dove p, q rappresentano i due istogrammi e p_i, q_i i valori degli i-esimi bins, inoltre i valori dei bins degli istogrammi LBP sono stati precedentemente normalizzati nel range $[0, 1]$ in modo da ottenere:

$$\sum_i p_i = 1$$

Il primo test è stato effettuato sulle immagini in Figura 2.3, ottenendo come risultati:

Distanza senza pre-processing	0.03794
Distanza con pre-processing	0.00538

risultati comparabili ai test effettuati nel paper di riferimento [1] sulle stesse immagini.

In Figura 4.1 sono mostrati i due istogrammi LBP calcolati sulle immagini senza la fase di pre-processing e l'intersezione tra di essi.

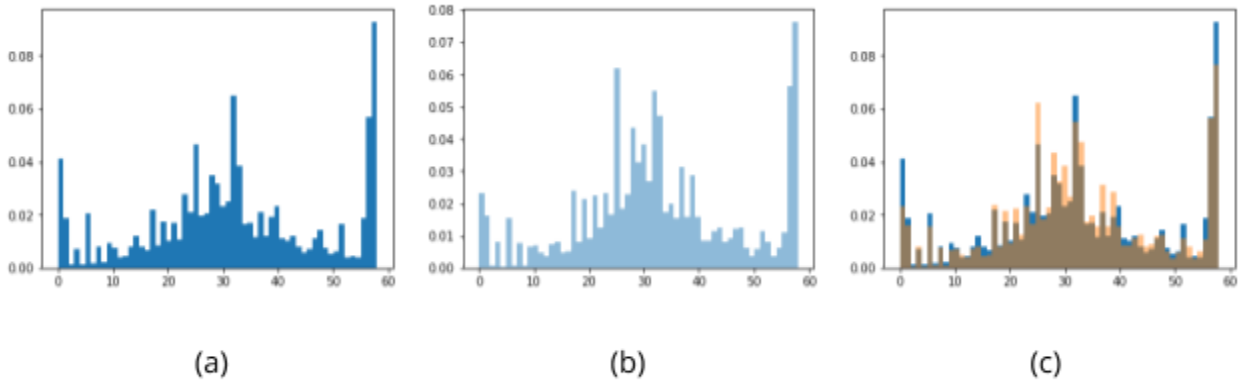


Figura 4.1. Istogrammi LBP immagini originali.

(a) Immagine 1. (b) Immagine 2. (c) Intersezione istogrammi.

In Figura 4.2 sono mostrati i due istogrammi LBP calcolati sulle immagini con la fase di pre-processing e l'intersezione tra di essi.

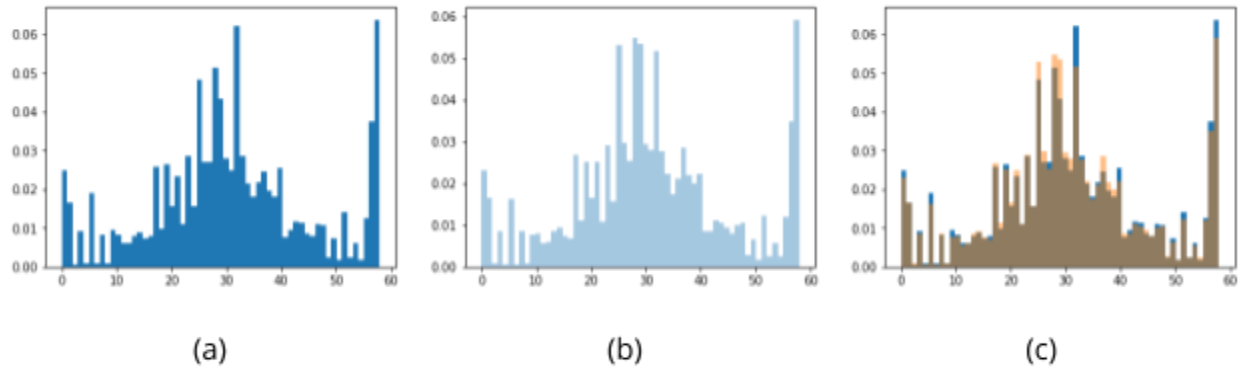


Figura 4.2. Istogrammi LBP immagini con preprocessing.

(a) Immagine 1. (b) Immagine 2. (c) Intersezione istogrammi.

Infine, in Figura 4.3 è mostrato il confronto tra le due intersezioni precedenti.

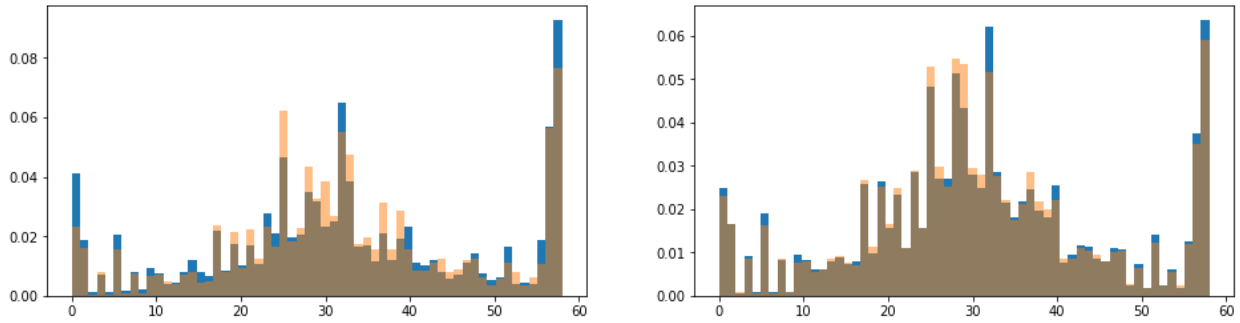


Figura 4.3: Confronto LBP

Il secondo test è stato effettuato su un dataset contenente 30 foto di 15 soggetti (2 per soggetto) scattate sotto due tipi di illuminazione diversa. In Figura 4.4 è mostrato il dataset prima e dopo il processo di normalizzazione.

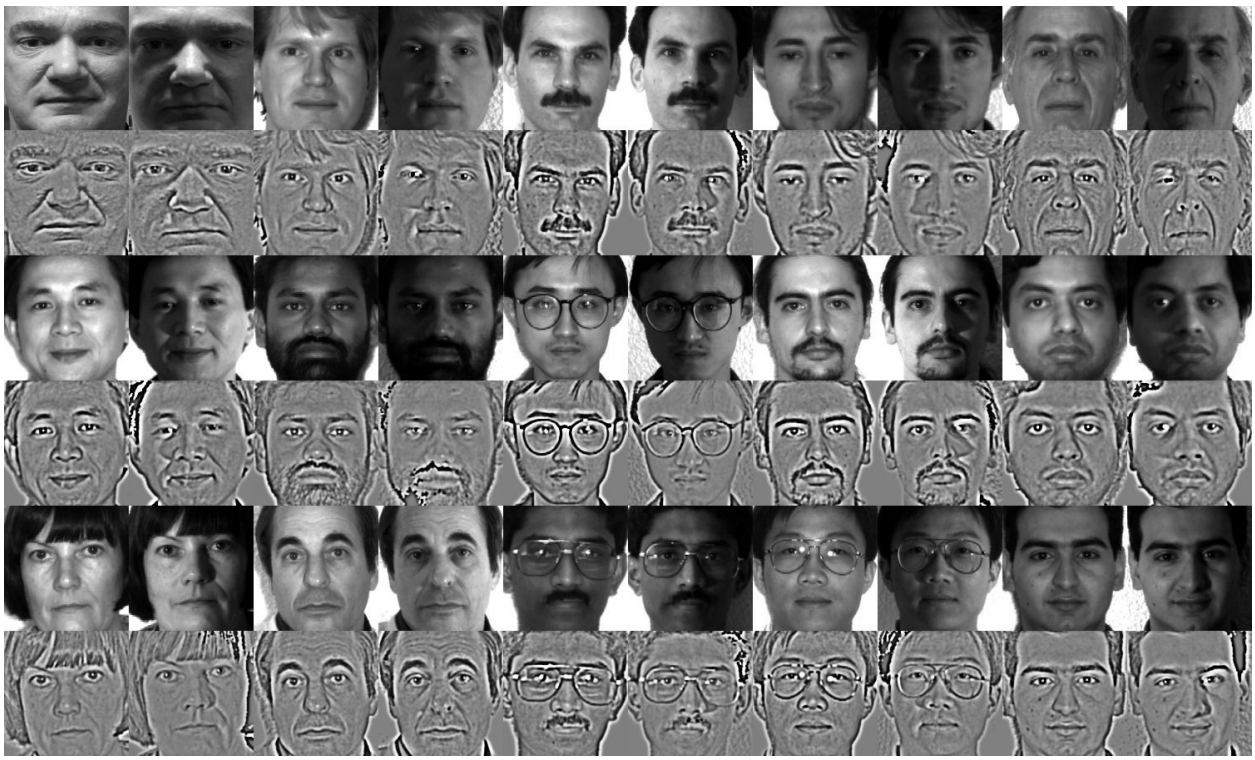


Figura 4.4

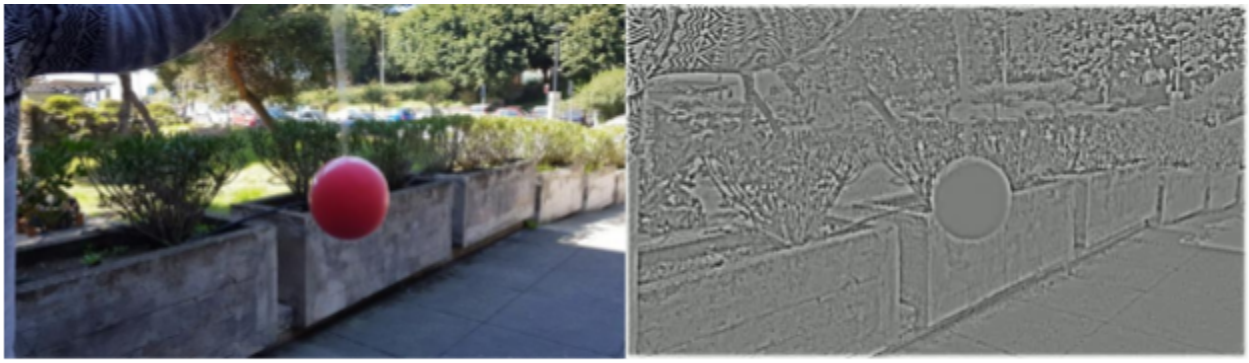
Il processo effettuato restituisce i seguenti risultati:

Media distanze senza pre-processing	0.06333
Media distanze con pre-processing	0.0121

risultati comparabili a quelli ottenuti nel test precedente.

L'ultimo test consisteva nel applicare il pre-processo per convertire diversi video prima di eseguire l'algoritmo Line2D su di essi (con lo scopo di riconoscere una palla ripresa nei videi in questione).

Un esempio di frame dei video è mostrato in Figura 4.5.



(a) (b)

Figura 4.5. (a) Frame originale. (b) Frame post pre-processo.

Dopo aver applicato il pre-processo l'algoritmo di detection però ottiene risultati generalmente inferiori/uguali alla versione base nei video di test. Una delle possibili cause è la conversione dell'immagine in gray-scale avvenuta nella fase iniziale del processo di normalizzazione, Line2D infatti calcola i gradienti (in ogni punto dell'immagine) per ogni canale (RGB) utilizzando successivamento quello di magnitudo maggiore.

5. Implementazione

Nel seguente capitolo verrà mostrata l'implementazione del pre-processo.

Uno dei problemi riscontrati durante la fase di sviluppo è stata la scarsa efficienza del metodo implementato, ciò ha portato a sviluppare due differenti versioni:

1. La prima versione in cui le operazioni nelle immagini sono svolte scorrendo, tramite utilizzo di due for annidati, tutti i pixel dell'immagine (intuitiva e semplice ma poco efficiente);
2. Una seconda versione ottimizzata che utilizza la libreria numpy per applicare le stesse operazioni (circa 40 volte più efficiente della precedente versione).

Di seguito sono mostrate alcune delle funzioni in comune tra le due versioni:

```
import numpy as np
import cv2
import os
from skimage import feature
import matplotlib.pyplot as plt
import time

def chi2_distance(histA, histB, eps = 1e-10):
    d = np.sum([((a - b) ** 2) / (a + b) for (a, b) in zip(histA, histB)])
    return d

def percentage_value(obj):
    lista = []
    for i in zip(set(obj)):
        lista.append(len(obj[i==obj])/len(obj))
    return np.array(lista)

def lbp(im):
    lbp = feature.local_binary_pattern(im,P=8,R=1,method='nri_uniform').reshape(1,-1)[0]
    return lbp

def robustness():
    dataset_path = "./dataset/"
    distance_with_preproc = distance_without_preproc = 0
    iteration=0
    for file in os.listdir(dataset_path):
            if(int(file[2])!=2):
                iteration+=1
                file_path_1 = os.path.join(dataset_path,file)
                file_path_2 = os.path.join(dataset_path,file.replace("1","2",1))
                im1 = cv2.imread(file_path_1)
                im2 = cv2.imread(file_path_2)
```

```
                im1_preproc = preprocessing(im1)
                im2_preproc = preprocessing(im2)
                im1_lbp = lbp(cv2.cvtColor(im1, cv2.COLOR_BGR2GRAY))
                im2_lbp = lbp(cv2.cvtColor(im2, cv2.COLOR_BGR2GRAY))
                im1_preproc_lbp = lbp(im1_preproc)
                im2_preproc_lbp = lbp(im2_preproc)

                x1 = percentage_value(im1_lbp)
                x2 = percentage_value(im2_lbp)
                y1 = percentage_value(im1_preproc_lbp)
                y2 = percentage_value(im2_preproc_lbp)

                distance_without_preproc+=chi2_distance(x1,x2)
                distance_with_preproc+=chi2_distance(y1,y2)

    distance_with_preproc/=iteration
    distance_without_preproc/=iteration
    print("mean chi_distance without preproc: ",distance_without_preproc)
    print("mean chi_distance with preproc: ",distance_with_preproc)

def preprocessing(img):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = img.astype(np.float64)
    #gamma correction
    img_gamma = adjust_gamma_1(img,gamma=0.2)
    #DOG
    img_dog = doG(img_gamma,sigma0=1,sigma1=2)
    #Contrast Equalization
    img_contrast_equalization = contrast_equalization(img_dog,a=0.1,t=10)
    img_final = cv2.normalize(img_contrast_equalization, None, 255,0, cv2.NORM_MINMAX, cv2.CV_8UC1)
    return img_final
```

Prima versione:

```
def adjust_gamma_1(image, gamma=0.2, use_threshold=False, threshold=0):
    im_copy = image.copy()
    for i in range(im_copy.shape[0]):
        for j in range(im_copy.shape[1]):
            if((use_threshold) & (im_copy[i][j]>threshold)):
                im_copy[i][j] = threshold**gamma
            else:
                im_copy[i][j] = image[i][j]**gamma
    return im_copy

def doG(image, sigma0=1, sigma1=2):
    g1 = cv2.GaussianBlur(image,(0,0),sigma0,borderType=cv2.BORDER_REPLICATE)
    g2 = cv2.GaussianBlur(image,(0,0),sigma1,borderType=cv2.BORDER_REPLICATE)
    doG = g1 - g2
    return doG

def contrast_equalization(image,a=0.1, t=10):
    im_copy = image.copy()
    d1 = adjust_gamma_1(abs(im_copy), gamma=a).mean()**(1/a)
```

```
    for i in range(im_copy.shape[0]):
        for j in range(im_copy.shape[1]):
            im_copy[i][j] = im_copy[i][j] / d1

    d2 = adjust_gamma_1(abs(im_copy),gamma=a,use_threshold=True, threshold=10).mean()**(1/a)
    for i in range(im_copy.shape[0]):
        for j in range(im_copy.shape[1]):
            im_copy[i][j] = im_copy[i][j]/d2

    for i in range(im_copy.shape[0]):
        for j in range(im_copy.shape[1]):
            im_copy[i][j] = t*np.tanh(im_copy[i][j]/t)

    return im_copy
```

Seconda versione ottimizzata sviluppata tramite l'utilizzo della libreria numpy:

```
def adjust_gamma_1(image, gamma=0.2, use_threshold=False, threshold=0):
    im_copy = image.copy()
    if(use_threshold):
        im_copy[im_copy>threshold] = threshold
    im_copy = np.power(im_copy,gamma)
    return im_copy

def doG(image, sigma0=1, sigma1=2):
    g1 = cv2.GaussianBlur(image,(0,0),sigma0,borderType=cv2.BORDER_REPLICATE)
    g2 = cv2.GaussianBlur(image,(0,0),sigma1,borderType=cv2.BORDER_REPLICATE)
    doGim = g1 - g2
    return doGim

def contrast_equalization(image,a=0.1, t=10):
    im_copy = image.copy()
    d1 = adjust_gamma_1(abs(im_copy), gamma=a).mean()**(1/a)
    im_copy = np.divide(im_copy,d1)
    d2 = adjust_gamma_1(abs(im_copy),gamma=a,use_threshold=True, threshold=10).mean()**(1/a)
    im_copy = np.divide(im_copy,d2)
    im_copy = np.multiply(np.tanh(np.divide(im_copy,t)),t)
    return im_copy
```

6. Conclusioni

In questo progetto è stato implementato il pre-processo di normalizzazione dell'illuminazione descritto nel paper*[1]* utilizzando il linguaggio di programmazione python e verificando il funzionamento di esso comparando i risultati ottenuti su un dataset formato da 30 immagini di 15 diversi soggetti rispetto a quelli del paper di riferimento.

Inoltre il pre-processo è stato integrato all'algoritmo di object detection Line2D *[2]* con l'intento di irrobustire quest'ultimo in situazioni di illuminazione "dinamica". Purtroppo il processo ha ottenuto degli esiti generalmente inferiore-uguali rispetto alla versione base sui video di test. Una probabile causa degli scarsi risultati ottenuti è la conversione in gray-scale delle immagini nella fase iniziale del pre-processo, essa porta infatti alla perdita di informazioni presenti nei tre canali RGB e utilizzate da Line2D per la detection degli oggetti.

References

[1] Tan X., Triggs B. (2007) Enhanced Local Texture Feature Sets for Face Recognition Under Difficult Lighting Conditions. In: Zhou S.K., Zhao W., Tang X., Gong S. (eds) Analysis and Modeling of Faces and Gestures. AMFG 2007. Lecture Notes in Computer Science, vol 4778. Springer, Berlin, Heidelberg

[2] S. Hinterstoisser *et al*., "Gradient Response Maps for Real-Time Detection of Textureless Objects" in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 5, pp. 876-888, May 2012.