

# Pre - Processing

*“Non è importante che l’immagine in output sia bella, ma è importante che funzioni per il mio metodo.”*

---

## Filtraggio del rumore

La scelta del filtro da utilizzare si è basata su queste considerazioni, in ordine (circa) di importanza:

### 1. Adattamento.

Non possiamo prevedere esattamente che tipologia di rumore - gaussiano o impulsivo - potrebbe presentarsi nei frame in input.

Al contempo, non è semplice trovare un ottimo metodo per filtrare qualsiasi tipologia di rumore in un tempo ragionevole.

Una prima scelta potrebbe essere il filtro mediano, ma abbiamo appreso a lezione che non sarebbe il massimo usarlo su un’immagine a colori.

### 2. Velocità.

Vi è la necessità di un filtro **non troppo oneroso computazionalmente**, ovvero che non incida esageratamente sull’esperienza computazione; bisogna ricordarsi che si deve processare un video, che solitamente contiene dai 15 ai 60 frame al secondo, e non una singola immagine.

Per questo sono state scartate opzioni come nfilter e colfilt, che seppur più prestanti ai fini del risultato non lo sono a livello computazionale.

### 3. Preservazione degli edge

Quando si effettua un filtraggio, è quasi inevitabile perdere dei *dettagli* dell’immagine. Tuttavia, cercare di evitare un “eccesso di smoothing”, utilizzando un filtro che, nel limite del possibile, effettui un buon filtraggio del rumore in più contesti, preservando dignitosamente le alte frequenze dell’immagine, può permettere di ridurre future computazioni per ulteriori sharpening dei frame.

Per questo è stata scartata, ad esempio, la funzione colfilt: effettuava un ottimo risultato nel filtraggio - in un tempo però poco ragionevole - ma la perdita

degli edge iniziava ad essere fin troppo consistente.

#### 4. “*Semplice*”.

Grazie alla documentazione Matlab, abbiamo visto vari [metodi efficaci](#); tuttavia le nostre conoscenze non sono sufficienti per usare con minima consapevolezza metodi simili, per cui a priori abbiamo dovuto escluderle.

## Wiener2

Premessa: non eravamo alla ricerca della miglior tecnica per filtrare il rumore. Volevamo un buon filtro versatile e non troppo oneroso computazionalmente, per rendere il nostro progetto maggiormente versatile.

Il vincitore di questa disputa è rappresentato dalla funzione Matlab [Wiener2](#).

Le motivazioni sono le seguenti:

### 1. Adattamento

La documentazione Matlab lo riporta come filtro adattativo, seppur abbia maggiori performance sul rumore gaussiano, sapendo che prima di effettuare il filtraggio egli esegue: “estimates of the additive noise power”.

Nello specifico, il filtro Wiener2 è un filtro passa basso senza una singola frequenza di taglio: la frequenza viene adattata in base alla regione, utilizzando un

- taglio più basso nelle regioni a basso dettaglio
- taglio più alto nelle regioni in cui vi sono edge

Spiegandolo in altre parole e non precise/corrette al 100%, il filtro Wiener è una sorta di filtro di media si adatta alla varianza dell'immagine locale, eseguendo uno

- Smoothing leggero dove la varianza è ampia, in quanto ci si aspetta che saranno presenti degli edge
- Smoothing maggiore laddove la varianza è minore

```
imagefilter = wiener2(image, [n, n]);
```

Lasciando di default il parametro *noise* (non specificato), la funzione calcola la media della varianza locale.

### Algorithms

wiener2 estimates the local mean and variance around each pixel.

$$\mu = \frac{1}{NM} \sum_{n_1, n_2 \in \eta} a(n_1, n_2)$$

and

$$\sigma^2 = \frac{1}{NM} \sum_{n_1, n_2 \in \eta} a^2(n_1, n_2) - \mu^2,$$

where  $\eta$  is the  $N$ -by- $M$  local neighborhood of each pixel in the image A. wiener2 then creates a pixelwise Wiener filter using these estimates,

$$b(n_1, n_2) = \mu + \frac{\sigma^2 - \nu^2}{\sigma^2} (a(n_1, n_2) - \mu),$$

where  $\nu^2$  is the noise variance. If the noise variance is not given, wiener2 uses the average of all the local estimated variances.

[1] Lim, Jae S. *Two-Dimensional Signal and Image Processing*, Englewood Cliffs, NJ, Prentice Hall, 1990, p. 548, equations 9.44, 9.45, and 9.46.

## 2. Velocità

La documentazione di Matlab riporta genericamente che “*wiener2 non è veloce come un generico filtro lineare*”.

Le slide del Professor Schettini riportano che i metodi adattativi potrebbero risultare computazionalmente onerosi.

Queste affermazioni tuttavia non ci forniscono un’idea per quantificare, e soprattutto soppesare, l’onere computazionale richiesto.

Soggettivamente, dopo ore e ore di progetto, possiamo affermare che il filtro wiener2 consente di avere un buon risultato generico con una velocità più che discreta.

Fornendo dei **dati oggettivi**, usando come misura oggettiva l’elapsed time di Matlab (nel codice, inserendo i simbatici `tic` - `toc`), abbiamo calcolato quanto possa essere, in generale, la differenza computazionale tra questo e un filtro lineare, prendendo un generico gaussiano, applicandoli sui singoli canali RGB: come risultati abbiamo ottenuto un +35 - 70 % in base alla dimensione dell’input.

Abbiamo poi stimato la differenza tra wiener2 e l’altro candidato, colfilt, sempre filtrando i singoli canali RGB con grandezza della maschera uguale: il filtro wiener 2 è più veloce circa del 75 - 90%.

La conclusione?

Con questi dati e le nostre valutazioni, abbiamo concluso che Wiener2 non sarà una scheggia, ma ci garantisce comunque un buon risultato sapendo che questo filtro non fa pasticci come un gaussiano se il tipo di rumore non è consono.

## 3. Preservazione degli edge.

La documentazione Matlab riporta la seguente frase: “*preserving edges and other high-frequency parts of an image.*”.

Come prima, ci aspettiamo che i gli edge vengano sufficientemente preservati rispetto ad un comune filtro di media o simile, ma è sempre meglio farsi

un'idea di cosa ci potremmo aspettare.

Mostriamo ora un esempio pratico, facendo “sfidare” wiener2 con il colfilt.:

*filtrando i singoli canali RGB di in un frame in input, con rumore stimato a sigma = 0.577026, con la funzione*

```
colfilt(canale, [5 5], 'sliding', @mean);
```

*restituisce un ottimo livello di sigma, pari a 0.278929, dove invece la funzione*

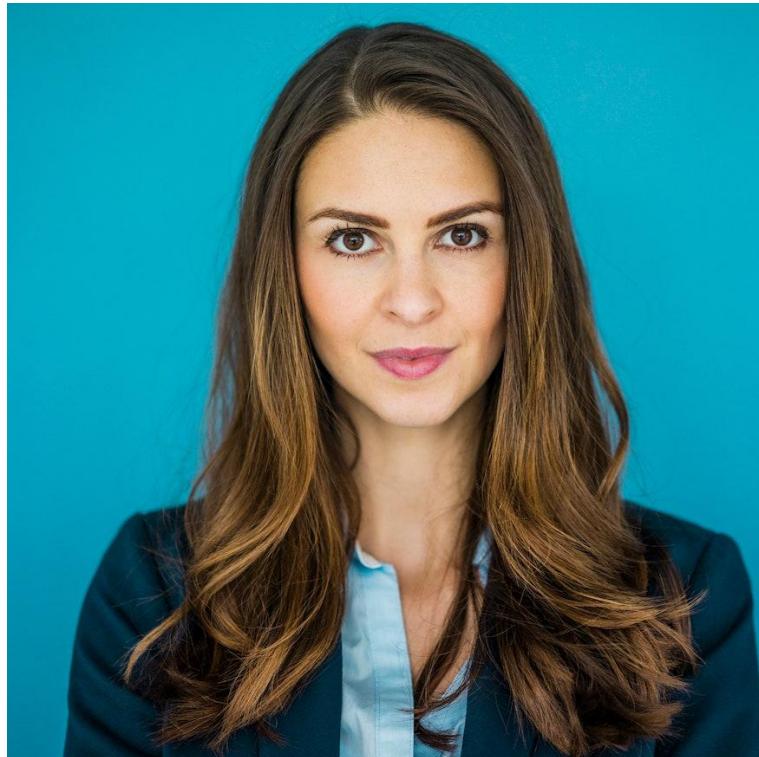
```
wiener2(canale, [5 5]);
```

*arriva ad una stima del rumore di sigma = 0.3254473.*

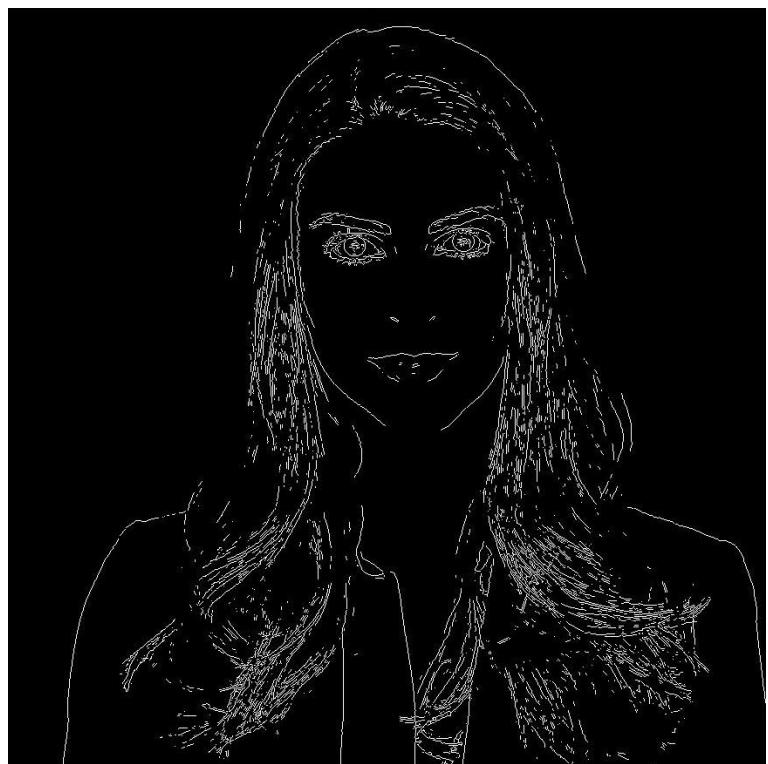
*Tuttavia, trovando gli edge ad esempio con Sobel, salta subito all'occhio la differenza nel mantenimento delle alte frequenze.*

*Inoltre, ricordiamo che la velocità di computazione del filtro wiener2 è risultata oggettivamente migliore del filtro colfilt - non uno stacco netto, ma comunque percepibile dall'utente.*

Vengono ora mostrati degli esempi significativi



**Originale**



**edge (Sobel) dopo il filtraggio con Wiener2**



**edge (Sobel) dopo il filtraggio con colfilt**

## Conclusioni

La funzione Wiener2 non rappresenta il filtro perfetto per qualsiasi situazione, né la soluzione definitiva. Per il nostro progetto tuttavia, lo riteniamo sufficientemente adatto.

Riguardo gli edge: non sappiamo attualmente se preservare maggiormente gli edge sia più importante o meno. Ci aspettiamo però che la cromaticità, e non gli edge, siano di maggior rilevanza nei metodi di segmentazione della pelle o simili.

## Filtrare il rumore in RealTime

Il problema dell'avere un video realtime è la velocità di computazione: le cose vanno fatte velocemente, altrimenti tanto vale non farle.

Presentiamo allora 3 alternative:

- a. Filtrare solo il canale Green con il filtro wiener2
- b. Filtrare il frame con filtro mediano, funzione predefinita medfilt3
- c. Filtrare i singoli canali RGB con il filtro mediano, funzione medfilt2

### **Nota: perché il canale G di RGB?**

*Dai paper e dalla documentazione online non è chiaro se sia meglio filtrare questo canale o il canale blu: molto pare dipendere anche dalla fonte di illuminazione della scena, ma sappiamo anche che la maggior parte dei sensori moderni sono progettati per catturare più informazioni sul canale verde.*

*Al che, abbiamo effettuato la cosa migliore che si potesse fare: abbiamo preso dei frame di prova significativi e abbiamo testato i risultati filtrando i vari canali.*

*Abbiamo poi iconcluso che filtrare il canale verde era sempre la scelta migliore, seguita dal canale blu e poi da quello rosso.*

Dai nostri test, sulla base dell'elapsed time e dello stimatore sigma del rumore, abbiamo concluso ciò:

### **- Velocità**

- Filtrare solo il canale Green con la funzione wiener2 permette di guadagnare il **50-60%** circa del tempo
- Usare la funzione medfilt3 non porta alcun guadagno - anzi, un rallentamento - in termini prestazionali.
- Filtrare i singoli canali RGB con la funzione medfilt2 - sempre il filtro mediano è - permette di **guadagnare l'80/88% del tempo rispetto a filtrare il solo canale G con Wiener2**

### **- Stima del rumore**

- Il risultato tra il filtrare solo il canale Green con wiener2 e filtrare tutti i canali RGB con il filtro mediano è molto simile.  
Con i frame significativi di prova, in cui il rumore è molto presente, è pressoché uguale. Con immagini meno rumorose, il risultato è leggermente migliore.
- Il filtro mediano introduce dei leggerissimi artefatti, percettibili solo zommando l'immagine - sempre premettendo che la maschera sia molto piccola.

A fronte di questi risultati, dopo aver sperimentato con Matlab il metodo di classificazione real-time, abbiamo concluso che sono **inutili**.

O meglio: ci possiamo permettere comunque di filtrare i singoli canali RGB dell'immagine anche in real-time, dunque non vi è bisogno di un cambio di approccio.

Nonostante ciò, la domanda che ci siamo posti inizialmente ci ha aperto comunque la strada a nuove conoscenze e altri orizzonti.

## **Nota**

*Non abbiamo trovato in rete controindicazioni nell'uso del filtro wiener2 sui canali di un'immagine RGB.*

*Inoltre, nelle slide del corso, quando si citano i problemi del filtro mediano su immagini a colori, si parla del fatto che: "non esiste un'ovvia estensione di elemento mediano su insiemi vettoriali".*

*Il filtro Wiener2 tuttavia esegue uno smoothing, e non una sostituzione dei valori dell'interno del pixel considerato con il mediano.*

<https://stackoverflow.com/questions/35972286/removing-salt-and-pepper-noise-using-wiener-filter-in-matlab>

<https://stackoverflow.com/questions/63563530/wiener-filtering-on-rgb-image>

## **RGB vs YCbCr**

Cercando tra i vari paper online, anche qui non è stato possibile capire quale sia il miglior spazio colore su cui effettuare il filtraggio: la disputa è tra lo spazio colore RGB e lo spazio YCbCr.

La timeline di consegna del progetto è comunque una realtà da sperimentare: non possiamo permetterci di provare tutte le combinazioni possibili.

Abbiamo eseguito allora qualche test sui dei frame significativi del nostro progetto.

Come detto prima, restando nello spazio RGB, pare che il canale verde sia il migliore, per la tecnologia della maggior parte dei sensori.

Comunque, con lo stimatore sigma abbiamo messo alla prova il filtro Wiener2 sui singoli canali RGB e poi i canali YCbCr.

Abbiamo concluso che, filtrando i singoli canali RGB, si otteneva un risultato migliore del ~ 10% a parità di dimensione della maschera.

Riportiamo, ad esempio, 4 test significativi effettuati usando una maschera 5x5 in ambo i casi:

- Frame test 1 > 10 %  
Noise estimation wiener2 image RGB 0.365317  
Noise estimation wiener2 image YCbCr 0.411608
- Frame test 2 > 10 %  
Noise estimation wiener2 image RGB 0.160009  
Noise estimation wiener2 image YCbCr 0.181322
- Frame test 3 ~ 9%  
Noise estimation wiener2 image RGB 0.149559  
Noise estimation wiener2 image YCbCr 0.165934
- Frame test 4 > 10 %  
Noise estimation wiener2 image RGB 0.162065  
Noise estimation wiener2 image YCbCr 0.184477

Non vi è motivo dunque di lavorare in YCbCr rispetto che al canale RGB.

Nota:

Abbiamo appreso comunque che, i canali riguardo la Luminanza (Y di YCbCr, L di Lab\*, ecc...), possono essere molto interessanti per filtrare il singolo canale.

---

## Sharpening

Abbiamo deciso di utilizzare la funzione predefinita di Matlab Imsharpen, definendo un valore di amount poco più alto di quello standard (da 0.8 a 1.0 circa).

Questa parte di pre-processing dell'immagine è meglio effettuarla dopo aver fatto filtrato il rumore presente, in quanto queste tecniche sono parecchio sensibili ad esso. A dimostrazione di ciò, effettuando dei test sulle immagini campione del progetto e usando lo stimatore sigma per capire il livello di rumore presente, senza il filtraggio del rumore si ottiene un peggioramento stimato tra il 13 e il 34 % rispetto a quando invece lo si effettua; si è osservato inoltre che, questa differenza si fa più ampia quando la “qualità” generale dell'immagine in input si fa peggiore: questo dato è molto significativo, in quanto l'obiettivo del pre-processing non è di migliorare sensibilmente un'immagine considerabile già “buona”, ma lavorare meglio su un'immagine in input di minor qualità.

La funzione Imsharpen è a tutti gli effetti un unsharp masking, quindi si ottiene l'immagine “sharpened” sottraendo una versione “unsharp” della stessa.

La funzione non richiede alcuna conversione in qualche spazio colore: prende automaticamente in input l'immagine codificata in RGB, la converte nello spazio colore L\*a\*b\* (lavorando solo sul canale L) per restituire in output l'immagine *sharpened* in formato RGB.

Ai fini del progetto, non risulta necessario enfatizzare eccessivamente l'immagine: l'obiettivo posto era ottenere un risultato, per quanto possibile, il più possibile simile al frame originale.

Abbiamo testato prettamente i frame di prova con questo comando

```
imfSharpen = imsharpen(imfWiener2, "amount", 1.0);
```

La tendenza, fino a questo punto, è quella di preferire un “pre-processing leggero” e non troppo invasivo: se, ad esempio, filtrassimo maggiormente il rumore con una maschera più grande (una 9x9 ad esempio), ma poi per recuperare aumentassimo il valore di amount per provare a “recuperare” nitidezza, la texture della pelle in ogni caso andrebbe comunque persa - questa feature potrebbe, chissà mai, ritornare utile ad esempio per la classificazione; ricordiamo inoltre che il filtro wiener2 preserva più che discretamente le alte frequenze.

Morale della favola: alla fine, abbiamo deciso di scartare questa parte di pre-processing, perché nel rapporto costo / beneficio non ha alcun senso con il metodo adottato.

---

## Auto White Balance (AWB)

Sono stati testati i 3 algoritmi [presentati in questa pagina](#).

Sinceramente, non abbiamo effettuato dei test per avere dei dati per scegliere oggettivamente quale fosse il miglior algoritmo.

In mancanza di ciò, possiamo comunque vantare il fatto che le valutazioni soggettive sono state fatte con l'ausilio di un monitor calibrato.

Il monitor in questione è un BenQ PD2700Q, il quale monta un pannello IPS a 10 bit simulati (8 bit + FRC), coprendo al 99.7% lo spazio colore sRGB.

La calibrazione viene eseguita mensilmente tramite un colorimetro Datacolor Spyder 5 Pro, software DisplayCal, luminanza a 120 cd/m<sup>2</sup>, punto di bianco 6500 kelvin, ΔE color medio = 0.2 e uno massimo di 0.8.

Si riportano qua i risultati della calibrazione:

### ▼ Overview

#	Device Values				Nominal Values			Measured Values				Color distance			
	R	G	B	L*	a*	b*	V	L*	a*	b*	V	ΔL* <sup>00w</sup>	ΔC <sup>00w</sup>	ΔH <sup>00w</sup>	ΔE <sub>00</sub>
01	255	255	255	100	0	0		100	0	0	0	0	0	0	0
02	0	0	0	1.1	0.05	0		1.1	-0.33	-0.57	0	0.67	-0.45	0.8	
03	13	13	13	2.28	-0.27	-0.17	2	2.32	-0.39	-0.73	1.99	0.03	0.48	0.31	0.57
04	26	26	26	6.67	-0.09	-0.17	2.13	6.84	0.13	-0.05	2.12	0.1	-0.02	0.35	0.37
05	38	38	38	13.76	-0.61	-0.35	2.15	13.71	-0.25	-0.19	2.15	-0.03	-0.54	0.06	0.54
06	51	51	51	19.88	-0.04	-0.8	2.19	19.85	-0.02	-0.86	2.19	-0.03	0.06	0.04	0.07
07	64	64	64	26.42	-0.5	-0.16	2.18	26.54	-0.52	-0.19	2.17	0.09	0.03	0.03	0.1
08	77	77	77	32.19	-0.48	-0.09	2.19	32.43	-0.88	-0.4	2.18	0.18	0.63	0.16	0.67
09	89	89	89	37.67	-0.29	-0.01	2.2	37.56	-0.13	-0.18	2.21	-0.1	-0.17	0.24	0.31
10	102	102	102	43.33	-0.22	-0.01	2.2	43.25	-0.22	-0.13	2.2	-0.07	0.02	0.12	0.14
11	115	115	115	48.63	-0.31	0.53	2.2	48.66	-0.37	0.54	2.2	0.01	0.06	0.06	0.09
12	128	128	128	53.65	-0.18	0.06	2.21	53.68	-0.38	0.09	2.21	0.03	0.3	0.03	0.3
13	140	140	140	58.7	-0.2	0.21	2.21	58.53	-0.21	0.09	2.22	-0.15	-0.03	0.12	0.19
14	153	153	153	63.74	-0.14	0.12	2.2	63.64	-0.21	0.04	2.21	-0.08	0.08	0.11	0.16
15	166	166	166	68.53	-0.37	0.08	2.2	68.5	-0.44	-0.13	2.21	-0.03	0.1	0.2	0.22
16	179	179	179	73.33	-0.39	0.13	2.2	73.36	-0.36	0.05	2.19	0.03	-0.06	0.07	0.1
17	191	191	191	77.7	0.15	-0.18	2.23	77.46	0.14	-0.36	2.25	-0.18	0.13	-0.13	0.25
18	204	204	204	82.58	-0.27	0.19	2.19	82.45	-0.54	0.12	2.21	-0.09	0.36	0.17	0.41
19	217	217	217	87.1	-0.05	0.38	2.18	87.05	-0.04	0.2	2.19	-0.03	-0.17	0.02	0.18
20	230	230	230	91.48	-0.06	0.51	2.17	91.6	-0.08	0.35	2.14	0.08	-0.15	0.06	0.18
21	242	242	242	95.85	-0.12	0.24	2.13	95.62	-0.1	-0.03	2.25	-0.14	-0.14	0.24	0.31
22	128	0	26	29.77	48.85	41.78		26.89	48.8	41.82		-0.06	0	0.04	0.08
23	255	0	0	55.7	82.96	82.49		55.58	82.87	82.63		-0.11	0.01	0.07	0.13
24	255	128	128	69.49	51.59	25.45		69.41	51.54	25.3		-0.07	-0.03	-0.06	0.1
25	0	128	0	46.04	-0.50	53.69		45.96	-0.50	53.44		-0.08	-0.03	0.09	0.13
26	0	255	0	87.88	-0.87	91.12		87.65	-0.75	91.16		-0.13	0.01	0	0.13
27	128	255	128	90.77	-0.35	53.07		90.57	-0.33	52.85		-0.12	-0.03	0.06	0.14
28	0	0	128	11.08	42.32	-69.05		11.11	42.31	-69.15		0.02	0.02	-0.03	0.05
29	0	0	255	27.77	74.99	-118.35		27.97	75.5	-118.09		0.16	0.01	0.28	0.32
30	128	128	255	58.17	27.61	-67.26		58.07	27.69	-67.53		-0.09	0.06	0.01	0.12
31	0	128	128	47.96	-33.52	-8.87		47.87	-33.56	-8.91		-0.09	0.02	0.02	0.09
32	0	255	255	90.26	-5.52	-18.03		90.15	-5.57	-16.27		-0.07	0.01	0.13	0.15
33	170	255	255	94.51	-28.4	-9.16		94.32	-28.5	-9.41		-0.11	0.07	0.14	0.2
34	128	0	128	29.98	58.16	-37.72		29.87	58.07	-37.8		-0.08	-0.01	-0.05	0.1
35	255	0	255	60.54	98.37	-62.55		60.49	98.31	-62.67		-0.04	0	-0.04	0.06
36	255	170	170	80.68	43.62	-29.92		80.65	43.47	-30.12		-0.02	0	-0.13	0.13
37	128	128	0	52.05	-9.59	62.47		51.97	-9.66	62.78		-0.08	0.08	0.02	0.12
38	255	255	0	97.97	-17.04	106.25		97.75	-17.08	106.23		-0.13	0	0.02	0.13
39	255	170	170	98.79	-9.84	42.99		98.67	-9.81	42.81		-0.07	-0.06	0.01	0.09
40	170	85	85	47.45	37.74	19.09		47.26	37.66	19.01		-0.19	-0.03	-0.02	0.19
41	85	170	85	63	-44.46	38.71		63.07	-44.85	38.87		0.06	0.11	0.06	0.14
42	85	85	170	39.02	20.33	-49.79		38.84	20.65	-50.19		-0.16	0.14	0.12	0.2
43	85	170	170	64.47	-29.63	-9.14		64.5	-29.85	-9.21		0.03	0.1	0	0.1
44	170	85	170	49.62	47.66	-31.95		49.44	47.64	-32.32		-0.17	0.05	-0.16	0.24
45	170	170	85	68.71	-9.56	47.82		68.71	-10.04	47.96		0	0.07	0.29	0.3
46	255	0	170	57.77	89.55	-18.03		57.69	89.57	-18.4		-0.07	0.02	-0.12	0.14
47	170	255	0	92.11	-51.85	97.65		91.97	-51.84	97.68		-0.08	0.02	0.06	0.1
48	0	170	255	65.51	-15.54	-55.45		65.63	-15.88	-55.32		0.09	-0.01	-0.18	0.2
49	0	255	170	88.82	-73.17	27.69		88.83	-73.35	27.6		0.01	0.03	0.06	0.07
50	170	0	255	45.28	85.47	-88.31		45.09	85.49	-88.75		-0.19	0.05	-0.12	0.23
51	255	170	0	77.23	26.23	91.54		77.19	26.06	91.36		-0.03	-0.04	0.06	0.08

Qui il test dell'uniformità del pannello nelle sue varie sezioni:

● 100%: 175.4 cd/m <sup>2</sup> (-11.55%), 1.66 ΔC <sup>00</sup>	● 100%: 167.35 cd/m <sup>2</sup> (-15.61%), 0.25 ΔC <sup>00</sup>	● 100%: 176.2 cd/m <sup>2</sup> (-11.15%), 1.58 ΔC <sup>00</sup>	● 100%: 159.87 cd/m <sup>2</sup> (+9.38%), 2.05 ΔC <sup>00</sup>	● 100%: 163 cd/m <sup>2</sup> (-17.81%), 2.73 ΔC <sup>00</sup>
● 75%: 90.9 cd/m <sup>2</sup> (-0.65%), 0.82 ΔC <sup>00</sup>	● 75%: 90.9 cd/m <sup>2</sup> (-0.32%), 0.82 ΔC <sup>00</sup>	● 75%: 90.9 cd/m <sup>2</sup> (-0.32%), 0.96 ΔC <sup>00</sup>	● 75%: 90.9 cd/m <sup>2</sup> (-0.32%), 0.96 ΔC <sup>00</sup>	● 75%: 90.9 cd/m <sup>2</sup> (-0.32%), 1.27 ΔC <sup>00</sup>
● 50%: 37.5 cd/m <sup>2</sup> (-2.64%), 0.54 ΔC <sup>00</sup>	● 50%: 37.5 cd/m <sup>2</sup> (-0.37%), 0.54 ΔC <sup>00</sup>	● 50%: 37.5 cd/m <sup>2</sup> (-0.37%), 0.54 ΔC <sup>00</sup>	● 50%: 37.5 cd/m <sup>2</sup> (-0.37%), 0.54 ΔC <sup>00</sup>	● 50%: 37.5 cd/m <sup>2</sup> (-0.37%), 0.76 ΔC <sup>00</sup>
● 25%: 8.16 cd/m <sup>2</sup> (-0.55%), 0.9 ΔC <sup>00</sup>	● 25%: 8.16 cd/m <sup>2</sup> (-0.55%), 0.9 ΔC <sup>00</sup>	● 25%: 8.16 cd/m <sup>2</sup> (-0.55%), 0.9 ΔC <sup>00</sup>	● 25%: 8.16 cd/m <sup>2</sup> (-0.55%), 0.9 ΔC <sup>00</sup>	● 25%: 8.16 cd/m <sup>2</sup> (-0.55%), 1.27 ΔC <sup>00</sup>
● Average: -12.58 cd/m <sup>2</sup> (-0.45%), 0.92 ΔC <sup>00</sup>	● Average: -13.37 cd/m <sup>2</sup> (-0.71%), 0.14 ΔC <sup>00</sup>	● Average: -13.47 cd/m <sup>2</sup> (-0.71%), 0.15 ΔC <sup>00</sup>	● Average: -15.97 cd/m <sup>2</sup> (-9.47%), 1.44 ΔC <sup>00</sup>	● Average: -17.53 cd/m <sup>2</sup> (-9.47%), 2.73 ΔC <sup>00</sup>
● Maximum: -26.08 cd/m <sup>2</sup> (-14.47%), 0.51 ΔC <sup>00</sup>	● Maximum: -26.08 cd/m <sup>2</sup> (-10.21%), 0.51 ΔC <sup>00</sup>	● Maximum: -26.08 cd/m <sup>2</sup> (-10.21%), 0.68 ΔC <sup>00</sup>	● Maximum: -20.23 cd/m <sup>2</sup> (-10.21%), 1.82 ΔC <sup>00</sup>	● Maximum: -21.31 cd/m <sup>2</sup> (-10.21%), 3.44 ΔC <sup>00</sup>
● Contrast deviation: 0.32%	● Contrast deviation: 0.4%	● Contrast deviation: 0.4%	● Contrast deviation: 0.3%	● Contrast deviation: 0.29%
● Recommended tolerance passed				
● 100%: 179.72 cd/m <sup>2</sup> (-14.42%), 1.51 ΔC <sup>00</sup>	● 100%: 177.37 cd/m <sup>2</sup> (-10.56%), 0.52 ΔC <sup>00</sup>	● 100%: 182.03 cd/m <sup>2</sup> (-0.41%), 0.58 ΔC <sup>00</sup>	● 100%: 178.08 cd/m <sup>2</sup> (-10.2%), 1.62 ΔC <sup>00</sup>	● 100%: 166.52 cd/m <sup>2</sup> (-16.03%), 2.44 ΔC <sup>00</sup>
● 75%: 93.78 cd/m <sup>2</sup> (-0.65%), 0.82 ΔC <sup>00</sup>	● 75%: 93.78 cd/m <sup>2</sup> (-0.32%), 0.82 ΔC <sup>00</sup>	● 75%: 93.78 cd/m <sup>2</sup> (-0.32%), 0.96 ΔC <sup>00</sup>	● 75%: 93.78 cd/m <sup>2</sup> (-0.32%), 0.96 ΔC <sup>00</sup>	● 75%: 93.78 cd/m <sup>2</sup> (-0.32%), 1.27 ΔC <sup>00</sup>
● 50%: 37.59 cd/m <sup>2</sup> (-2.67%), 0.54 ΔC <sup>00</sup>	● 50%: 37.59 cd/m <sup>2</sup> (-0.37%), 0.54 ΔC <sup>00</sup>	● 50%: 37.59 cd/m <sup>2</sup> (-0.37%), 0.54 ΔC <sup>00</sup>	● 50%: 37.59 cd/m <sup>2</sup> (-0.37%), 0.54 ΔC <sup>00</sup>	● 50%: 37.59 cd/m <sup>2</sup> (-0.37%), 0.76 ΔC <sup>00</sup>
● 25%: 8.40 cd/m <sup>2</sup> (-0.55%), 0.9 ΔC <sup>00</sup>	● 25%: 8.40 cd/m <sup>2</sup> (-0.55%), 0.9 ΔC <sup>00</sup>	● 25%: 8.40 cd/m <sup>2</sup> (-0.55%), 0.9 ΔC <sup>00</sup>	● 25%: 8.40 cd/m <sup>2</sup> (-0.55%), 0.9 ΔC <sup>00</sup>	● 25%: 8.40 cd/m <sup>2</sup> (-0.55%), 1.27 ΔC <sup>00</sup>
● Average: -9.73 cd/m <sup>2</sup> (-0.92%), 0.92 ΔC <sup>00</sup>	● Average: -12.29 cd/m <sup>2</sup> (-1.15%), 0.14 ΔC <sup>00</sup>	● Average: -12.52 cd/m <sup>2</sup> (-1.15%), 0.15 ΔC <sup>00</sup>	● Average: -13.31 cd/m <sup>2</sup> (-1.15%), 1.62 ΔC <sup>00</sup>	● Average: -17.53 cd/m <sup>2</sup> (-1.15%), 2.73 ΔC <sup>00</sup>
● Maximum: -22.99 cd/m <sup>2</sup> (-11.55%), 1.68 ΔC <sup>00</sup>	● Maximum: -29.17 cd/m <sup>2</sup> (-14.71%), 1.54 ΔC <sup>00</sup>	● Maximum: -31.39 cd/m <sup>2</sup> (-14.71%), 0.68 ΔC <sup>00</sup>	● Maximum: -37.79 cd/m <sup>2</sup> (-14.71%), 1.88 ΔC <sup>00</sup>	● Maximum: -23.53 cd/m <sup>2</sup> (-11.55%), 2.73 ΔC <sup>00</sup>
● Contrast deviation: 2.32%	● Contrast deviation: 2.3%	● Contrast deviation: 2.3%	● Contrast deviation: 2.07%	● Contrast deviation: 2.75%

Ricordiamoci però il principio fondamentale del pre-processing in questo progetto:

*“Non è importante che l’immagine in output sia bella, ma è importante che funzioni per il mio metodo.”*

Abbiamo testato i 3 algoritmi con i diversi frame di prova, significativi per il nostro progetto, variando di volta in volta i percentili (ovvero la porzione di pixel esclusi “alle code”, ovvero quei pixel troppo scuri o troppo chiari).

Riportiamo ora i risultati, prima soggettivi e poi con delle immagini allegate:

- PCA
  - Non comporta un significativo risultato, in particolare non corregge bene le aree in cui la tinta prevalente è tendente al viola.  
In alcuni frame introduce un’eccessiva tonalità bluastra.
- White Patch Retinex
  - Comporta un risultato significativo solo impostando un percentile alto ( $>10$  circa), ma introduce con costanza una dominante di verde eccessiva, anche quando non ve n’è bisogno.
- **Gray World**
  - Algoritmo vincente con un percentile uguale a 1, corregge sufficientemente bene le zone violacee della pelle introducendo una tonalità verde, mentre in immagini già ben bilanciate non effettua alcuna distruttiva come White Patch Retinex.  
A volte la tonalità di verde è leggermente più forte di quanto voluto, ma non è un grosso problema in quanto ciò avviene in aree “non pelle”, ed anzi una pelle tendente al violaceo corretta con l’introduzione di una tonalità verde risulta migliorata.

Mostriamo ora tre esempi significativi, partendo da un frame con tonalità violacea:



**Originale**



**PCA**



### White Patch Retinex



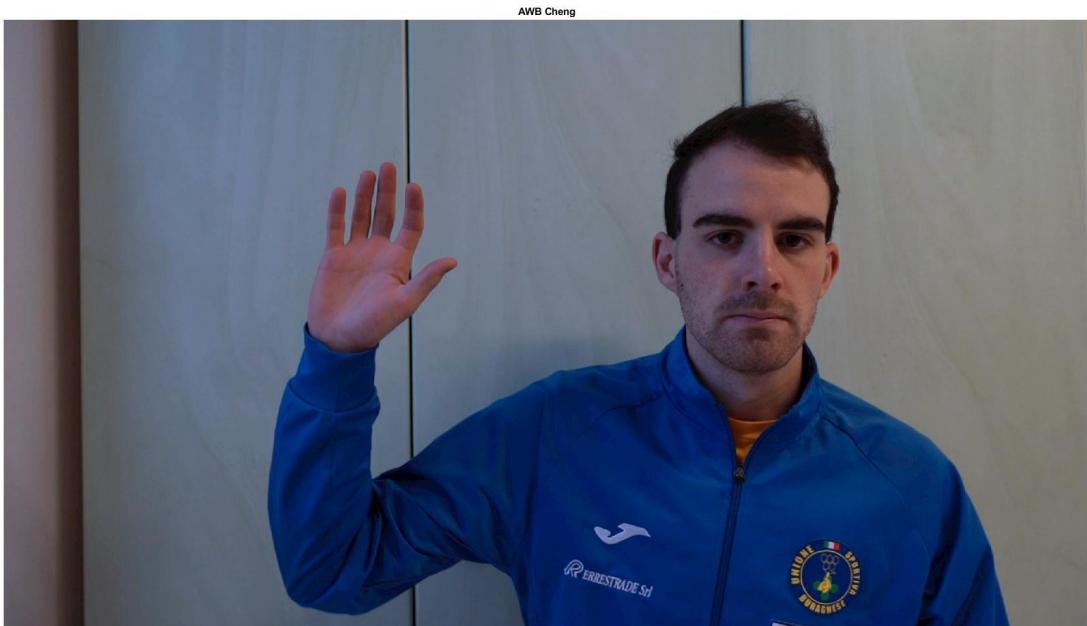
### Gray World

L'immagine originale, tendente al violaceo, viene corretta abbastanza bene dal White Patch Retinex (percentile 10) e dal Gray World (percentile 1). Il PCA apporta una piccola correzione (forse difficile da percepire dalla presentazione) anche abbastanza coerente, ma un po' insufficiente per rendere lo skin tone più naturale.

Mostriamo ora un altro test su un frame ben bilanciato in partenza (acquisito da una Sony A7III, bilanciamento del bianco automatico in camera), significativo anche per altri 4 frame di test:



Originale



PCA



**White Patch Retinex**



**Gray World**

L'immagine originale, che presenta uno scarso contrasto (profilo flat in camera, più precisamente un S-Log2) e una tonalità violacea, viene mal corretta dall'algoritmo PCA, che introduce una dominante di bluastra / viola eccessiva.

L'algoritmo Gray World si comporta bene, eliminando una leggerissima tonalità giallognola, mentre il WPR si comporta sufficientemente bene, introducendo però una dominante violacea/bluastra un po' superiore.

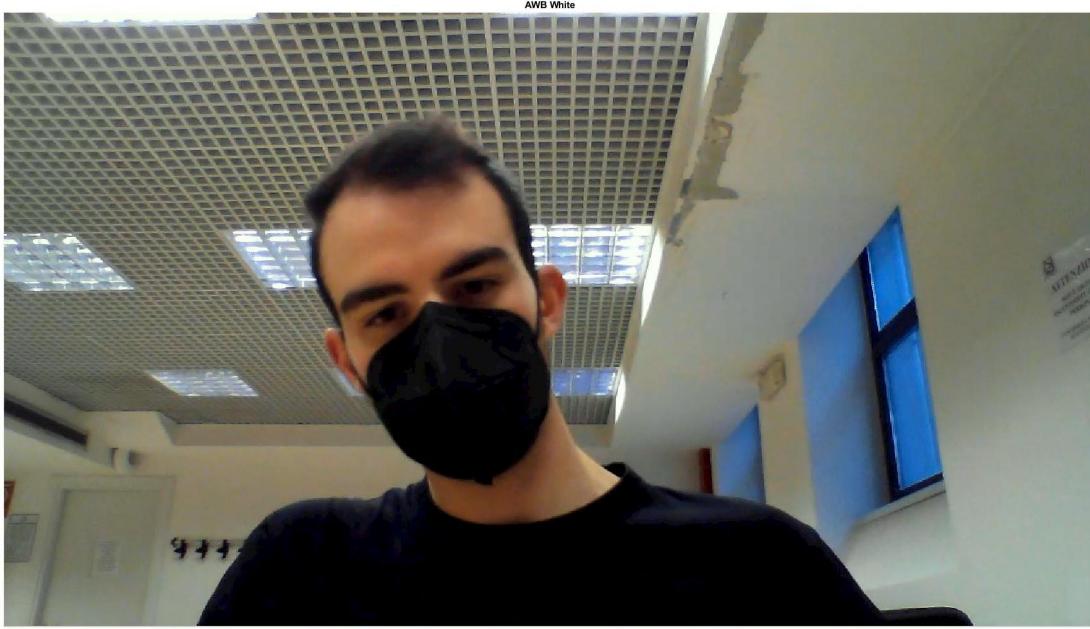
Vediamo ora l'ultimo esempio, con un'immagine di bassa qualità acquisita da una comune webcam integrata in un notebook nell'aula studio U14:



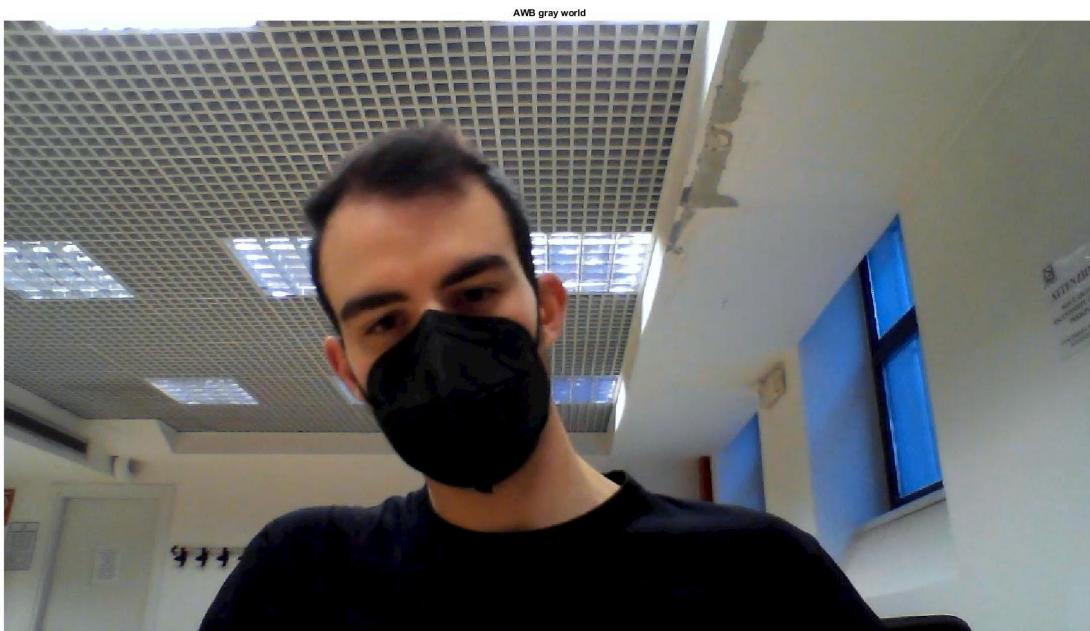
**Originale**



**PCA**



**White Patch Retinex**



**Gray World**

In questo caso, l'algoritmo Gray World stravince: corregge molto bene l'immagine di partenza, eliminando leggermente la dominante giallognola, facile da notare osservando le pareti che diventano maggiormente bianche.

Non bene PCA, che invece esalta leggermente la dominante giallo/verdognola al posto di ridurla (facile da notare sempre osservando le pareti).

Male White Patch Retinex, che introduce una tonalità verdognola eccessiva.

## **Nota sulla velocità computazionale**

è stato testato l'onere computazionale per effettuare l'AWB con l'algoritmo selezionato, e si è notato che richiede ugual o più tempo circa di effettuare un filtraggio del rumore con la funzione wiener2 (maschera 5x5).

Lo riteniamo dunque un costo accettabile ai possibili benefici - analizzati in seguito - dell'applicazione di tale metodo.

---

# **Contrast Adjustment**

Sono state valutati due approcci principali:

1. Uso della funzione imadjust, ovvero uno “stretch” dell’istogramma
2. Adapthisteq, ovvero una equalizzazione *adattativa* (che lo rende più adatto di histeq) dell’istogramma.

Un primo vantaggio del secondo approccio è sicuramente la possibilità di regolare il processamento: di default il risultato è eccessivo, ma giocando con i parametri a ribasso, si può ottenere un risultato discreto.

Il vantaggio invece di usare **imadjust** è indubbiamente la **velocità**: vi è una differenza di circa il **~90% delle prestazioni per la computazione** - una media di ben **0,144 secondi contro 1,468 secondi** di media nei nostri test.

I risultati? A occhio abbastanza contrastanti.

Vediamo ora le nostre analisi:

### **→ Imadjust**

- ◆ Velocità di computazione nettamente superiore
  - Lavora sull’immagine RGB
- ◆ Ponendo come limiti di contrasto dell’immagine in input un’operazione tra media e varianza (suggerimento di Matlab).
- ◆ Tendenzialmente compie un buon lavoro, ma su alcuni frame “esagera” un po’, calcando eccessivamente i neri.
- ◆ Su altri frame invece si comporta molto meglio di adapthisteq, enfatizzando a giusto modo lo skin tone.

## → Adapthisteq

- ◆ Abbiamo già detto che ha una velocità di computazione nettamente inferiore a imadjust
  - Non lavora sull'immagine RGB ma solo sul canale L dello spazio L\*a\*b, come suggerito da Matlab
- ◆ Settato con parametri molto “leggeri” rispetto a quelli standard, altrimenti crea un effetto alone tremendo intorno al soggetto
  - *Il risultato assomiglia molto alla funzione “chiarezza” che troviamo nei vari software di editing foto e video*
- ◆ Tendenzialmente non effettua grandi stravolgimenti dell'immagine, tranne in qualche caso dove, invece, imadjust invece rimane più *leggero*, va a calare maggiormente il contrasto, senza però andare a deteriorare l'immagine.
- ◆ Migliora molto il contrasto sui dettagli, dando la sensazione di un effetto di “sharpening”.
- ◆ Il **problema** principale è lo **skin tone**: a volte sembra essere penalizzato.

Riportiamo ora 7 esempi pratici per maggior trasparenza espositiva

### Esempio 1



Originale

imadjust



### Imadjust

adaphisteq



### Adaphisteq

#### Esempio 2

Originale)



## Originale



## Imadjust



## Adaphthisteq

### Esempio 3



## Originale



## Imadjust



## Adaphisteq

### Esempio 4

Originale



Originale

imadjust



Imadjust

adaphisteq



Adaphisteq  
Esempio 5

Originale



Originale

imadjust



Imadjust

adaphisteq



Adaphisteq

## Esempio 6

Originale



## Originale

imadjust



## Imadjust

adapthisteq



## Adapthisteq

Personalmente, mi ritrovo molto combattuto nella scelta: tendenzialmente sceglierrei il metodo che prevede l'utilizzo di `adaphisteq`, tuttavia non vorrei che lo skin tone, così importante per il nostro progetto, possa essere penalizzato rispetto all'utilizzo invece di `imadjust`.

Tuttavia, ricordiamoci sempre che il fine non è di mandare in output una bella immagine, ma di apportare miglioramenti ed essere compatibile con il nostro algoritmo di risoluzione del problema.

**L'onere computazionale** richiesto per **`adaphisteq`** è **eccessivo**, mentre `imadjust` funziona molto velocemente: siamo costretti a scartare il primo, e a valutare l'efficacia di questo metodo direttamente in base ai risultati dati dal classificatore.

---

## ASSUNZIONI

Per il nostro progetto, **assumiamo che le immagini non siano tendenzialmente bimodali**, in quanto ci aspettiamo un contesto con una illuminazione tendenzialmente uniforme.

Per dare degli esempi, ci aspettiamo una stanza illuminata di una casa qualsiasi o in un ambiente esterno di giorno.

Non ci aspettiamo dei frame con una persona collocata in una stanza buia che si punta una torcia in faccia, né dei frame in esterna al tramonto / sera con il flash sparato sul soggetto, perché il nostro algoritmo **potrebbe**, e ripetiamo **potrebbe fallire**.

Nel caso si voglia una maggiore garanzia di risultato - ma il nostro algoritmo in generale non è stato pensato per questa tipologia di immagini - basta disattivare l'AWB e l'aggiustamento del contrasto.

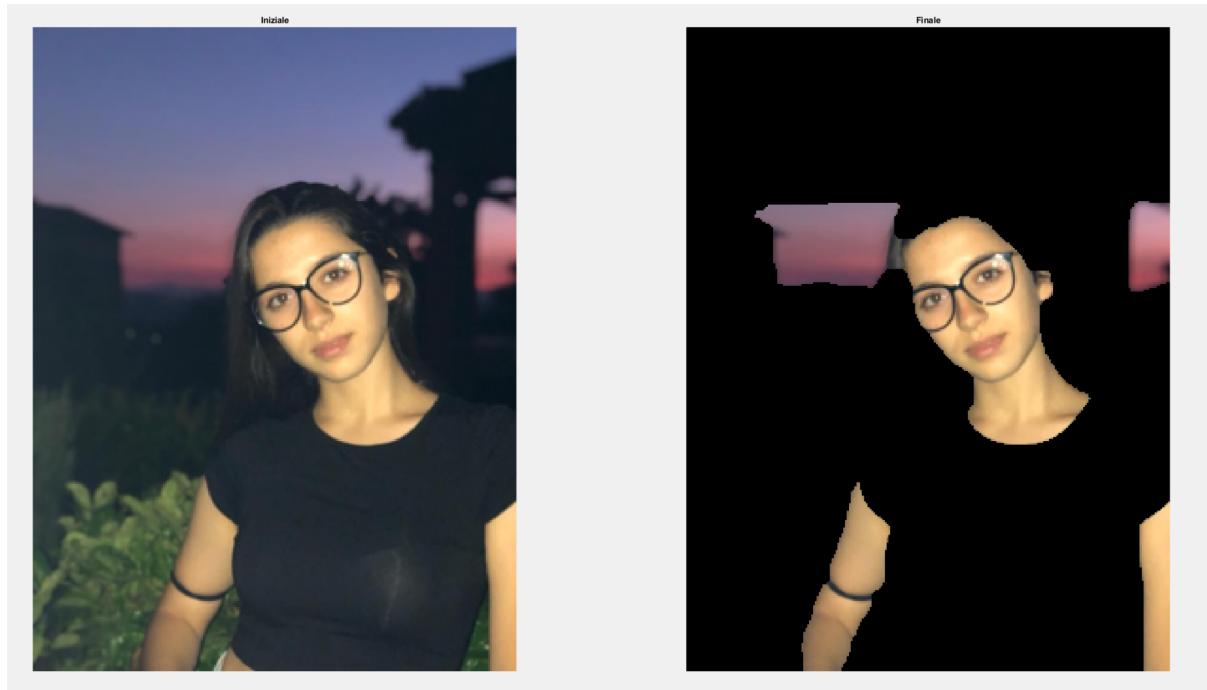
Nella prossima pagina illustreremo due test di prova con dei frame gentilmente concessi da un'amica, **tali per cui chiediamo di non prelevarli da questo paper al fine di non ledere in alcun modo la sua immagine**.

## Prova frame 1: fallimento.

L'aggiustamento del contrasto e l'AWB **deteriorano** fortemente non l'immagine generale, ma lo **skin tone**.

```
cm =  
  
    struct with fields:  
  
        cm_raw: [2×2 double]  
        cm: [2×2 double]  
        labels: [2×1 logical]  
        accuracy: 0.8336  
  
ans =  
  
    0.9433    0.0567  
    0.9341    0.0659
```

Disattivandoli, il nostro algoritmo ritorna ad avere risultati efficientissimi, come dimostrato dalla matrice di confusione.



```

cm =

struct with fields:

    cm_raw: [2×2 double]
        cm: [2×2 double]
    labels: [2×1 logical]
accuracy: 0.9459

ans =

0.9480    0.0520
0.0685    0.9315

```

## Prova frame 2: Successo!

Un frame sempre fatto di sera con illuminazione complessa, in un contesto a noi sconosciuto come il precedente.

L'aggiustamento del contrasto e l'AWB **non compromettono** lo skin tone, ottenendo risultati pressoché identici

```

cm =

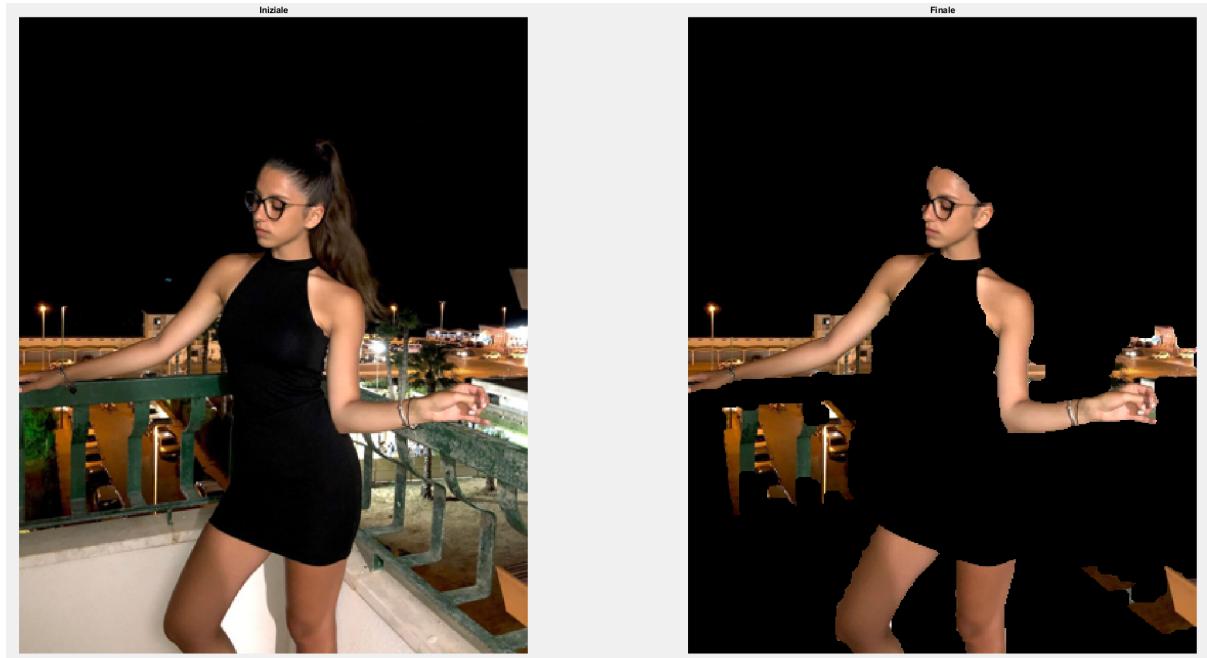
struct with fields:

    cm_raw: [2×2 double]
        cm: [2×2 double]
    labels: [2×1 logical]
accuracy: 0.9270

ans =

0.9243    0.0757
0.0500    0.9500

```



Senza AWB e imadjust il risultato è pressoché identico, come mostrato dalla matrice di confusione

```
cm =  
  
    struct with fields:  
  
        cm_raw: [2×2 double]  
        cm: [2×2 double]  
        labels: [2×1 logical]  
        accuracy: 0.9290  
  
ans =  
  
    0.9248    0.0752  
    0.0355    0.9645
```

Per altri test, consultare le slide di presentazione.