



UNIVERSITÀ DEGLI STUDI DI TRIESTE

DIPARTIMENTO DI FISICA

Corso di Laurea in Fisica

Tesi di Laurea Magistrale

**Un'implementazione in Python della
Eulerian Video Magnification per
rivelare cambiamenti quasi invisibili
in sequenze di immagini**

Laureando:
Vincenzo Vitale

Relatore:
prof. Edoardo Milotti.

ANNO ACCADEMICO 2019-2020

Indice

1	Introduzione	2
2	Elaborazione digitale delle immagini	4
2.1	Le immagini per un calcolatore	6
2.2	Convoluzioni	7
2.3	Sfocatura	10
2.4	Rappresentazione piramidale delle immagini	11
2.4.1	Piramide Gaussiana	12
2.4.2	Piramide Laplaciana	14
2.5	Filtri di rinforzo	15
2.6	Edge detection	16
2.6.1	Filtraggio del rumore	16
2.6.2	Convoluzione e gradiente	17
2.6.3	Ricerca dei massimi	18
2.6.4	Isteresi bordo debole e bordo forte	19
2.7	Riconoscimento di oggetti	20
2.8	Algoritmo di Lucas e Kanade	21
2.9	Flusso ottico	24
2.10	Calcolare il flusso ottico	27
2.11	Metodo di Horn e Schunck	29
3	Eulerian Video Magnification	32
3.1	Approccio Euleriano e approccio Lagrangiano.	32
3.2	EVM	34
3.3	Teoria	35

3.4	Problema dell'amplificazione del rumore	37
3.5	Limiti	38
4	Algoritmo	43
4.1	GOOGLE COLAB PRO	44
4.2	Importazione del video.	45
4.3	Decomposizione spaziale	46
4.4	Elaborazione temporale.	49
4.5	Filtro passa banda	51
4.6	Nuovo segnale	52
4.7	Amplificazione del segnale	52
4.7.1	Amplificazione totale	53
4.7.2	Amplificazione parziale	53
4.8	Ricostruzione video finale	55
5	Esperimenti iniziali	57
5.1	Video artificiali	57
5.2	Oscillazioni cromatiche	58
5.3	Piccole oscillazioni armoniche.	61
5.4	Allargamento	65
5.5	Piccole oscillazioni con rumore.	66
5.6	Amplificazione parziale o totale?	68
5.7	Oscillazioni a frequenze differenti	71
5.8	Movimenti casuali	73
6	Video	77
6.1	Volto e cambi colore.	77
6.2	Respirazione e movimenti	81
6.3	Pendolo	83
6.3.1	2 minuti	84
6.3.2	5 minuti	91
7	Conclusioni	94

Capitolo 1

Introduzione

Lo scopo di questo lavoro di tesi è quello di implementare una versione in Python dell'algoritmo proposto da Wu et al. [17] nel 2012. L'algoritmo in questione è noto con il nome di Eulerian Video Magnification (EVM) ed è finalizzato a rivelare cambiamenti all'interno delle successioni di immagini. Le potenzialità dell'EVM sono molteplici: in campo medico può essere utilizzato per monitorare a distanza le funzionalità vitali dei pazienti. In campo industriale/ingegneristico permette di visualizzare le piccole oscillazioni a cui sono soggette costruzioni e macchine industriali. Questo permette di prevenire l'usura e il deterioramento. In campo fisico l'EVM potrebbe essere utilizzato per individuare le oscillazioni di apparati sperimentali. In questo lavoro sono presentate alcune tecniche di analisi digitale delle immagini elaborate nei decenni passati. È poi introdotta la base teorica su cui poggia l'Eulerian Video Magnification ed è analizzato punto per punto il codice che implementa l'algoritmo. I risultati ottenuti sono riportati negli ultimi capitoli della tesi. L'algoritmo è stato scritto in Python, linguaggio sempre più in uso e dalle potenzialità incredibili, con l'obiettivo di renderlo accessibile al maggior numero di persone possibili. Gli effetti ottenuti grazie all'EVM sono mostrati grazie a video creati artificialmente. Questo permette di stimare quantitativamente l'importanza dell'amplificazione ottenuta. I video ottenuti con l'algoritmo implementato in Python sono confrontati con quelli ottenuti dagli autori dell'articolo in cui viene presentata la tecnica dell'EVM.

Questo è possibile perché si è lavorato con gli stessi video utilizzati in [17]. Infine, nella parte finale, è stato studiato l'effetto dell'amplificazione sul video di un pendolo le cui oscillazioni sono state (quasi) totalmente smorzate dall'attrito dell'aria.

Eulerian Video Magnification è già stata utilizzata in fisica dei plasmi da Ryan [12] riuscendo a rendere visibili le oscillazioni temporali del plasma del MAST (Mega Ampere Spherical Tokamak). L'EVM può essere applicata a qualsiasi fenomeno oscillatorio in un plasma luminescente.

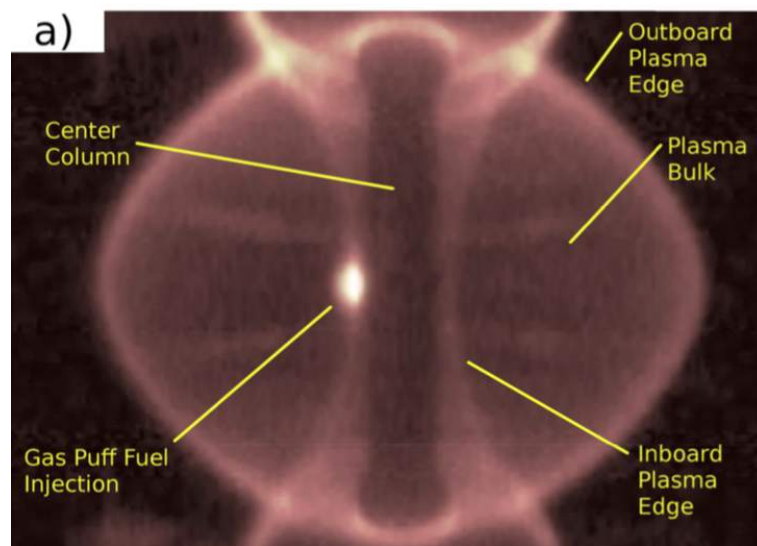


Figura 1.1: Un fotogramma del video rappresentante le oscillazioni del plasma del MAST analizzato in [12].

Capitolo 2

Elaborazione digitale delle immagini

La visione artificiale è un ramo dell'informatica finalizzato a rendere i calcolatori capaci di "vedere". Per comprendere questa affermazione è necessario chiedersi cosa si intenda con il verbo vedere. Nel mondo biologico la visione non è altro che la capacità di percepire stimoli luminosi. In informatica parleremo di visione artificiale quando un computer è capace di trasformare un input (che ha il formato di un'immagine o di un video) in un output che contenga informazioni esplicite sul contenuto dell'input stesso. E' necessario fare una distinzione tra visione artificiale ed elaborazione digitale delle immagini. Quest'ultima infatti non è finalizzata a ottenere informazioni specifiche sul contenuto dell'input ma si limita a generare un altro file a partire dallo stesso formato dell'originale. Le modifiche possono essere lievi come nel caso dell'inversione dei colori o significative come nel caso degli algoritmi di identificazione dei bordi. I primi studi in questo campo arrivarono subito dopo l'introduzione delle immagini in formato digitale nel 1957 [6] e si ponevano obiettivi molto semplici come la sfocatura delle immagini o la creazione di negativi digitali. Per decenni l'apparato visivo umano è rimasto estremamente più efficace dei sistemi informatici e la ricerca si focalizzava sullo sviluppo di tecniche di elaborazione digitale piuttosto che sulla visione artificiale. Con il progredire delle capacità di calcolo la visione artificiale ha

cominciato a raggiungere obiettivi sempre più concreti e a essere applicabile in vari campi. Una spinta definitiva in questa direzione è arrivata con lo sviluppo delle tecniche di "Machine Learning" che hanno permesso di raggiungere obiettivi inimmaginabili fino a qualche anno prima. Ad oggi se si dispone del software giusto, di un computer e di una telecamera è possibile diagnosticare malattie, riconoscere persone, animali e oggetti, controllare la stabilità di alcune strutture e anche eseguire previsioni meteorologiche. Da qualche anno esistono negli Stati Uniti dei supermercati in cui non vi è la necessità di passare dalla cassa: un insieme di telecamere è infatti capace di seguire il cliente e di individuare tutti gli articoli che esso inserisce nel carrello per poi addebitargli il conto tramite il suo metodo di pagamento preferito. Il sorprendente sviluppo della visione artificiale ha tratto moltissima ispirazione dai tentativi di comprensione del funzionamento della visione umana e animale. Le prime reti neurali provavano a replicare il funzionamento di un cervello composto da neuroni collegati da sinapsi. Lo stesso concetto di addestramento è di ispirazione biologica. Nel 1959 i neuropsicologi David Hubel e Torsten Wiesel [5] dimostrarono che la corteccia visiva dei gatti non era attivata dalla presenza degli oggetti stessi ma dal riconoscimento di strutture molto semplici come può essere un contorno orientato correttamente. Questa scoperta, apparentemente indipendente dallo sviluppo delle tecniche di visione artificiale, sarà utilizzata nello sviluppo delle reti neurali convoluzionali (CNN). Quest'insieme di aspetti rendono la visione artificiale un naturale punto di incontro tra varie discipline scientifiche: dalla biologia, alla matematica passando per i modelli fisici e le applicazioni informatiche.

In questo capitolo sono presentate alcune tecniche che hanno segnato la storia dello sviluppo dell'elaborazione digitale delle immagini. Alcune di queste, come i metodi piramidali [1], verranno usate direttamente nell'implementazione dell'Eulerian Video Magnification; altre invece introducono al mondo dell'analisi digitale delle successioni di immagini. I metodi di Lucas and Kanade [8] e di Horn and Schunck [4] indicano alcune metodologie per calcolare il flusso ottico, concetto strettamente legato al rilevamento di movimenti nei video. L'Eulerian Video Magnification rientra tra le tecniche di elaborazione digitale delle successioni di immagini. I video forniti come input

subiscono una serie di trasformazioni finalizzate alla creazione di un nuovo file dello stesso formato e delle stesse dimensioni di quello di partenza in cui gli spostamenti, anche minimi, siano resi visibili all'occhio umano sprovvisto di ulteriori strumenti.

L'anno zero per *l'image processing* è da identificare con il 1957, data in cui Kirsch [6] e i suoi colleghi del "National Bureau of Standards" riuscirono a trasformare una fotografia del figlio di Kirsch in una matrice di dimensioni 176 x 176.



Figura 2.1: La prima rappresentazione digitale creata con la scannerizzazione di una fotografia. La risoluzione non è elevatissima ma permette di distinguere il soggetto. [6]

2.1 Le immagini per un calcolatore

Ma cosa è un'immagine per un computer? La telecamera, lo scanner o qualsiasi metodo di acquisizione si decida di usare trasforma l'input visivo in una matrice bidimensionale. Più alta è la risoluzione maggiore sarà la dimensione della matrice e più dettagliata sarà l'immagine stessa. Ogni punto della matrice viene chiamato pixel. In ogni pixel sono contenute esclusivamente le

informazioni riguardanti la luminosità e il colore di quel singolo punto. Le prime immagini digitali furono rappresentate in scala di grigi, era sufficiente un unico numero intero compreso tra 0 e 255 per ogni pixel. Successivamente con lo sviluppo degli schermi fu possibile rappresentare l'insieme dei colori. Ad ogni pixel non corrisponde più un unico numero ma tre. Questi tre numeri rappresentano rispettivamente l'intensità del rosso, del verde e del blu che combinati permettono di riprodurre qualsiasi colore. Per passare alla dinamicità dei video è sufficiente allineare una serie di immagini. Il loro rapido susseguirsi rende possibile la riproduzione del video.

L'elaborazione delle immagini digitali si riduce dunque all'insieme delle operazioni che è possibile effettuare sui valori numerici associati ai pixel.

2.2 Convolutioni

Tra le prime tecniche di elaborazione digitale delle immagini sviluppate c'è la convoluzione. Quest'ultima consiste nell'applicare un filtro (chiamato kernel o mascherina) di piccole dimensioni per generare il valore dei pixel dell'immagine finale. Questi filtri si presentano sotto forma di matrici le cui dimensioni sono dispari in modo da identificarne con semplicità il centro. Di seguito è possibile vedere come vengono determinati i valori in output partendo dall'immagine originale.

Il filtro F :

$$F = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

applicato all'immagine i cui pixel sono indicati con m_{ij} produce una nuova matrice \tilde{m}_{ij} definita nel seguente modo:

$$\begin{aligned} \tilde{m}_{ij} = & a m_{i-1;j-1} + b m_{i-1;j} + c m_{i-1;j+1} + d m_{i;j-1} + e m_{i;j} + \\ & + f m_{i;j+1} + g m_{i+1;j-1} + h m_{i+1;j} + i m_{i+1;j+1} \end{aligned}$$

Facendo scorrere il filtro lungo tutta l'immagine originale si ottiene quindi un nuovo insieme di numeri che può essere considerato come una nuova

figura. In generale per garantire che la luminosità dell'immagine venga preservata i filtri vengono normalizzati. Per ottenere con una convoluzione un'immagine di dimensioni uguali a l'originale è necessario trovare una strategia per permettere al filtro di lavorare anche con i confini dell'immagine. La procedura più comune consiste nell'estendere in orizzontale il valore dei pixel dei bordi laterali e in verticale i valori dei pixel dei bordi inferiore e superiore. E' importante ricordare che nonostante la sua semplicità, la convoluzione rimane un processo irreversibile: il suo utilizzo implica un'inevitabile perdita di informazione. Ecco l'effetto ottenuto applicando i seguenti filtri:

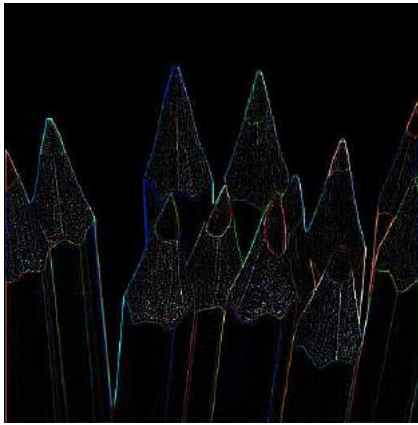
$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad (2.1)$$



(a) Immagine originale



(b) Primo filtro in (2.1)



(c) Secondo filtro in (2.1)



(d) Terzo filtro in (2.1)

Figura 2.2: Risultati ottenuti tramite convoluzione. Sono stati usati i filtri rappresentati in (2.1)

Tramite una convoluzione è possibile anche riprodurre l'immagine originale. In questo caso si usa il filtro:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

2.3 Sfocatura

Sebbene la sfocatura sia accompagnata da un'inevitabile perdita d'informazione, l'operazione può essere interessante per alcuni scopi specifici. Quando un'immagine viene sfocata il rumore è attenuato poiché viene distribuito su una superficie più estesa. Questo va a distribuire uniformemente su una zona di 9 pixel ciò che prima era contenuto in un singolo pixel. Il filtro *media* non è ovviamente l'unico filtro utilizzabile per sfocare le immagini. Il filtro Gaussiano è probabilmente il più usato e ripartisce l'intensità dei pixel seguendo la distribuzione Gaussiana. Ecco due filtri Gaussiani di dimensioni diverse:

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad \frac{1}{273} \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \qquad (2.2)$$

Il filtro *mediano* non funziona come una vera convoluzione dato che non effettua la moltiplicazione tipica vista in (??).

Esso genera una lista contenente il valore di nove pixel contenuti in un quadrato. Questa lista viene ordinata in maniera crescente per poi selezionare il valore mediano che verrà inserito nella posizione centrale. Di seguito sono riportati i risultati ottenuti con filtri Gaussiani di dimensioni diverse e con un filtro mediano. Si è partiti da un'immagine a cui era stato aggiunto del rumore per evidenziare l'efficacia dei filtri in fase di "denoising".



Figura 2.3: In alto a sinistra l'immagine originale. In alto a destra l'output del filtro mediano. In basso i risultati ottenuti con i filtri rappresentati in (2.2)

2.4 Rappresentazione piramidale delle immagini

Una convoluzione con un filtro di sfocatura restituisce un'immagine di dimensioni uguali a quella originale. Certe volte può essere utile lavorare con una versione dell'immagine originale di dimensioni ridotte. La riduzione delle dimensioni e quindi del numero di pixel, rende più veloce l'esecuzione di certi algoritmi. L'insieme di immagini a diversa risoluzione generate a partire da un'originale è chiamata piramide. Questo appellativo è facilmente comprensibile dal momento che, se sovrapposte in ordine di risoluzione decrescente, formano una struttura piramidale. E' possibile creare tanti tipi diversi di piramide in funzione del metodo di ridimensionamento utilizzato.

2.4.1 Piramide Gaussiana

La libreria OpenCv costruisce ogni elemento della piramide Gaussiana in due fasi. In un primo momento si effettua una convoluzione con un filtro Gaussiano:

$$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} \quad (2.3)$$

Il passo successivo consiste nella semplice rimozione delle colonne e delle righe di numero pari: si ottiene così un'immagine di dimensioni uguali a un quarto dell'originale. La piramide non viene generata fino alla "cima" ma si decide arbitrariamente un punto oltre il quale non si ritiene necessario procedere. Le piramidi create con questo meccanismo sono chiamate piramidi Gaussiane. Per comprendere al meglio il risultato di questa operazione riporto qui di seguito l'insieme di immagini della piramide costruite partendo da una foto della terra.

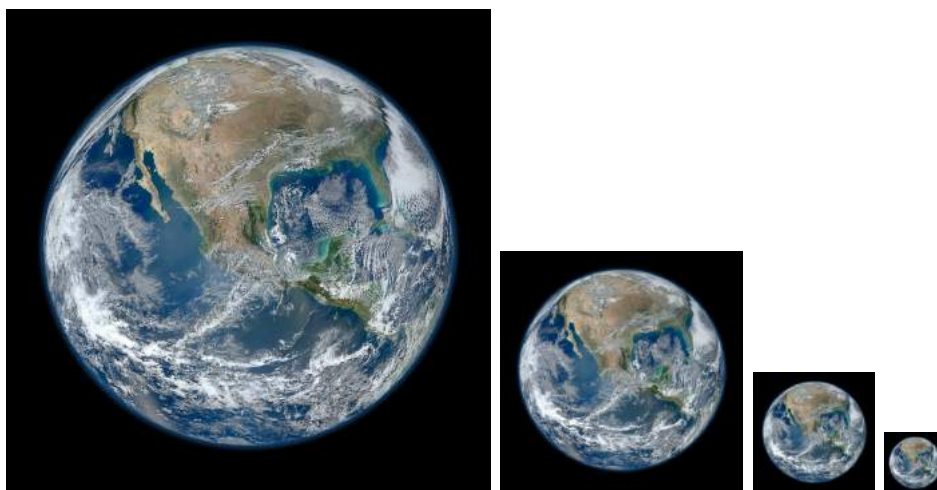


Figura 2.4: L'insieme di immagini che compongono la piramide Gaussiana.

Per ogni strato della piramide vi è una duplice inevitabile perdita di informazioni. La convoluzione e l'eliminazione delle righe e delle colonne pari. Ciononostante è possibile definire un processo inverso per ottenere immagini con una risoluzione uguale all'originale partendo dai livelli più alti della piramide Gaussiana. La libreria OpenCV esegue questa operazione nel seguente modo:

- Vengono inserite delle righe e delle colonne di pixel uguali a 0.
- Viene dunque applicato il filtro Gaussiano visto in (2.3) moltiplicato di un fattore 4 per definire i nuovi pixel.

La moltiplicazione per un fattore 4 garantisce la conservazione dell'intensità media dei pixel e quindi dei colori presenti nell'immagine. Questo è dovuto al fatto che a seguito dell'aggiunta di righe e colonne ogni punto non è più circondato da una regione di dimensioni 5×5 composta di 24 pixel diversi da zero. Adesso 16 di questi 24 pixel sono uguali a zero. Andando a valutare la perdita di intensità della nuova immagine grazie ai valori presenti in (2.3) è possibile stimare a 4 il fattore moltiplicativo che permette di conservare l'intensità media dei pixel.



Figura 2.5: L'immagine originale affiancata al terzo livello della piramide Gaussiana riportato alle dimensioni originali.

Per confrontare numericamente queste immagini è necessario fare attenzione alle dimensioni. Gli algoritmi che permettono di passare da un livello all'altro della piramide rimuovono e inseriscono colonne e righe con indice pari. È possibile che le dimensioni di un'immagine della piramide riportata alla risoluzione originale come nella figura 2.5 differiscano di qualche unità da quelle originali. Se l'immagine in questione è quadrata di dimensioni $n \times n$ con n dispari allora verranno rimosse $\frac{n-1}{2}$ righe e $\frac{n-1}{2}$ colonne nel salire di livello all'interno della piramide. L'operazione successiva aggiunge una riga e una colonna a ogni riga e ad ogni colonna. Verranno quindi aggiunte $\frac{n+1}{2}$ righe e $\frac{n+1}{2}$ colonne. Salire di un livello nella piramide per poi scendere può quindi comportare una modifica delle dimensioni del file.

2.4.2 Piramide Laplaciana

La piramide Laplaciana viene costruita a partire da quella Gaussiana ma contiene informazioni fondamentalmente diverse. Anche in questo caso la struttura piramidale richiama l'ordine per risoluzione decrescente. Il livello i della piramide Laplaciana è definito come la differenza tra il livello i della piramide Gaussiana e il livello $i + 1$ della piramide Gaussiana riportato alle dimensioni del livello i . Ogni strato sarà dunque utile per evidenziare la perdita di informazioni che si ha quando si effettua un filtraggio per convoluzione con una maschera Gaussiana.



Figura 2.6: La piramide Laplaciana creata a partire da quella gaussiana rappresentata in figura 2.4. Data la sua definizione la piramide Laplaciana presenta sempre uno strato in meno.

2.5 Filtri di rinforzo

Nella figura 2.2 si è visto che esiste una convoluzione che ha come effetto quello di evidenziare i bordi. Questo tipo di filtro viene chiamato filtro di rinforzo e la sua struttura tende ad evidenziare le zone in cui ci sono repentini e considerevoli cambiamenti di intensità. Se si guarda al meccanismo di funzionamento delle convoluzioni si comprende che se applicato su una regione uniforme allora l'output finale avrà un'intensità media uguale a zero. Ecco il filtro di rinforzo:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad (2.4)$$

2.6 Edge detection

L'utilizzo del filtro di rinforzo è il primo passo nello sviluppo degli algoritmi di "Edge detection". Con questo nome vengono catalogate le tecniche che permettono di evidenziare i contorni di un'immagine. Un bordo può essere identificato se un lieve spostamento sull'immagine è accompagnato da un significativo cambiamento dell'intensità dei pixel. Se l'obiettivo è avere esclusivamente una visione dei principali contorni presenti in un'immagine è possibile utilizzare tecniche più raffinate sviluppate nel corso degli anni. Nel 1986 Canny [2] elaborò un algoritmo abbastanza semplice dai risultati sorprendenti. L'algoritmo si divide in 4 fasi:

- Filtro Gaussiano
- Calcolo del gradiente
- Ricerca dei massimi
- Soglia inferiore e soglia superiore.

2.6.1 Filtraggio del rumore

Dopo aver convertito l'immagine in scala di grigi è possibile procedere con il filtraggio Gaussiano finalizzato alla rimozione del rumore. Questa fase è importante poiché le repentine variazioni dell'intensità dei pixel possono essere anche dovute al rumore che degrada la qualità dell'immagine. Questo può avere diverse origini; l'usura dei dispositivi, il processo di acquisizione o un errore nei processi di memorizzazione dei file. Il primo passo dell'algoritmo consiste nell'applicare un filtro di tipo Gaussiano in modo da ridurre il rumore. Le dimensioni del filtro Gaussiano ne influenzano la sensibilità come si è visto nella figura 2.3. Con il crescere delle dimensioni del filtro il rumore viene filtrato più efficacemente.

2.6.2 Convoluzione e gradiente

Dal momento che l'immagine è stata convertita in scala di grigi, ogni pixel è caratterizzato da un unico valore numerico compreso tra 0 e 255. Questi numeri possono essere visualizzati come un insieme discreto di valori assunti da una funzione a due variabili. Se l'obiettivo dell'algoritmo è quello di individuare le zone di confine, identificabili dalle variazioni di intensità dei pixel, allora il calcolo del gradiente dei valori dei pixel è uno strumento utile. Nelle zone in cui il colore è uniforme il gradiente sarà quasi nullo mentre in prossimità dei contorni sarà significativamente diverso da zero. La seconda fase dell'algoritmo di Canny è dunque finalizzata al calcolo del gradiente di un'immagine in formato digitale. Essendo il gradiente un vettore bidimensionale si procede al calcolo indipendente del valore delle due componenti. Questa operazione può essere effettuata grazie alla convoluzioni. Un primo metodo può essere l'utilizzo dei kernel:

$$\begin{bmatrix} 1 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

che per ogni punto (x, y) di intensità $I(x, y)$ calcolano i valori:

$$\Delta x = I(x + 1, y) - I(x, y) \quad \Delta y = I(x, y + 1) - I(x, y)$$

Questa tecnica ha il difetto di coinvolgere due soli pixel per ogni calcolo e di utilizzare dei filtri di dimensioni pari. Per conseguire l'obiettivo è opportuno tenere conto delle variazioni in una regione meno limitata al fine di eliminare l'influenza di ulteriori impurità dell'immagine che non rappresentano di certo un contorno netto. Per questo obiettivo risultano molto utili i filtri di Prewitt (a sinistra in (2.5)) e Sobel (a destra in (2.5)) che differiscono tra loro esclusivamente per il peso attribuito ai primi vicini nella direzione del calcolo.

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (2.5)$$

Se si considera il pixel (x, y) con intensità $I(x, y)$ rappresentato tramite la matrice:

$I(x-1, y+1)$	$I(x, y+1)$	$I(x+1, y+1)$
$I(x-1, y)$	$I(x, y)$	$I(x+1, y)$
$I(x-1, y-1)$	$I(x, y-1)$	$I(x+1, y-1)$

che permette di visualizzare l'intensità dei pixel adiacenti. Le componenti del gradiente calcolate con i filtri di Sobel saranno dunque:

$$G_x = \frac{\partial}{\partial x} I(x, y) = -I(x-1, y+1) - 2I(x-1, y) - I(x-1, y-1) \\ + I(x+1, y+1) + 2I(x+1, y) + I(x+1, y-1)$$

$$G_y = \frac{\partial}{\partial y} I(x, y) = -I(x-1, y+1) - 2I(x, y+1) - I(x+1, y+1) \\ + I(x-1, y-1) + 2I(x, y-1) + I(x+1, y-1)$$

Da queste espressioni è possibile calcolare il modulo del gradiente e l'angolo della sua direzione θ :

$$G = \|\vec{\nabla} I(x, y)\| = \sqrt{G_x^2 + G_y^2} \quad \theta = \arctan\left(\frac{G_y}{G_x}\right)$$

L'angolo θ viene arrotondato a una delle quattro direzioni principali: le due diagonali e i due assi orizzontali e verticali.

2.6.3 Ricerca dei massimi

La presenza di un gradiente diverso da zero non è una condizione sufficiente a identificare un bordo. Effetti di prospettiva o di illuminazione possono essere cause alternative di una graduale e continua variazione nell'intensità dei pixel. Il passo successivo consiste dunque nell'azzerare i valori in cui il gradiente è diverso da zero ma che non possono essere associati a situazioni di bordo. Per questo ogni valore del gradiente viene paragonato ai due valori adiacenti lungo la direzione calcolata grazie a θ . Solo i punti che presentano un massimo locale nel valore del gradiente restano candidati alla ricerca di bordi. Tutti gli alti punti vengono azzerati.

2.6.4 Isteresi bordo debole e bordo forte

A questo punto non resta che selezionare un valore di soglia al di sopra del quale si ritiene che il gradiente rappresenti un bordo. Questa soglia è da determinare empiricamente dato che dipende molto dal tipo di immagine su cui si sta lavorando. L'algoritmo funziona però con due livelli di soglia per selezionare i valori del gradiente da conservare. Se l'intensità del gradiente è superiore alla soglia alta, allora il punto è considerato come un bordo forte e non viene alterato. Se invece l'intensità è compresa tra la soglia alta e quella bassa, il punto viene catalogato come bordo debole. Questo tipo di pixel verrà considerato come bordo esclusivamente nel caso in cui ci sia un pixel catalogato come bordo forte nell'insieme di otto punti che lo circondano. Ecco alcuni risultati ottenuti utilizzando la libreria OpenCV di Python.

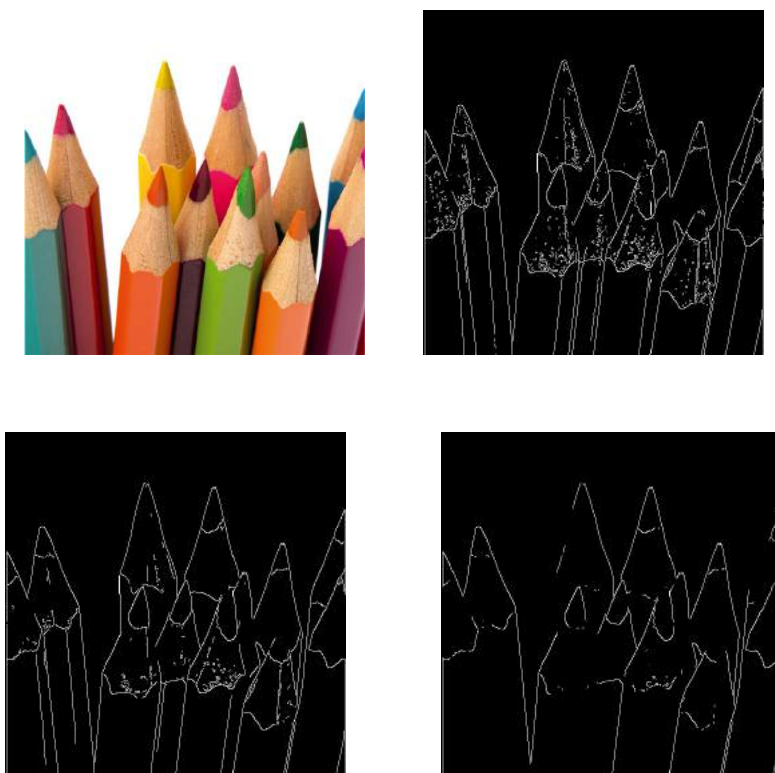


Figura 2.7: Edge detection effettuata con l'algoritmo di Canny. I tre risultati corrispondono a diversi range per la fase finale di filtraggio dell'algoritmo. In ordine sono riportati i risultati ottenuti con: 50-100, 120-180 e 200-250.

2.7 Riconoscimento di oggetti

I metodi numerici per l'analisi di immagini digitali si sono sviluppati enormemente negli ultimi decenni. Questo sviluppo ha portato anche a una ramificazione dei campi d'interesse. La rivelazione di oggetti in un fotogramma è uno dei rami più studiati della visione artificiale. Acquisita la capacità di identificare gli oggetti, si è sviluppato lo studio dei metodi di tracciamento che permettono di seguire gli spostamenti di una particolare struttura in una successione di fotogrammi. Lo sviluppo considerevole dello studio del comportamento dinamico degli oggetti è spiegabile in virtù della molteplicità di applicazioni tecnologiche e industriali correlate ad esso. I progressi dell'ingegneria informatica hanno permesso di applicare su macchine non troppo costose l'insieme delle teorie sviluppate a partire dalla seconda metà del secolo scorso.

Tra i vari settori interessati agli sviluppi della visione artificiale vi è la guida assistita. Questa necessita di analizzare in tempo reale una successione di frame per poter anticipare gli eventi evitando frenate improvvise o addirittura incidenti. Anche in ambito biomedico vi è un continuo sviluppo di tecnologie finalizzate all'analisi non invasiva che permettono di monitorare fattori vitali come il battito cardiaco o il respiro senza dover necessariamente entrare in contatto con il paziente. A seguito della crisi sanitaria è sicuramente più facile comprendere gli incredibili vantaggi della possibilità di una diagnosi a distanza.

Uno dei primi problemi ad essere stato affrontato con metodi matematico-numerici relativamente all'analisi di immagini riguarda quella che viene chiamata la *registrazione* d'immagini. Il problema consiste nell'individuare la presenza di forme uguali o molto simili all'interno di figure diverse. Il metodo deve essere in grado di riconoscere questa similitudine a meno di traslazioni o di rotazioni del corpo studiato. Questo problema è strettamente correlato al movimento degli oggetti e rappresenta il primo tentativo di tracciare il movimento di un corpo in una successione di immagini. Come vedremo in 2.8 il riconoscimento di oggetti simili avviene tra 2 fotogrammi diversi. Applicare questo metodo iterativamente a una successione di immagini permette di

generare un primo rilevatore dei movimenti.

2.8 Algoritmo di Lucas e Kanade

L'algoritmo che fu proposto nel 1981 da Lucas e Kanade[8] è semplice quanto efficace e permette di studiare il mondo dell'analisi delle immagini con uno sguardo fisico e senza disporre di specifiche competenze informatiche.

In una dimensione il problema diventa il seguente: date due funzioni $F(x)$ e $G(x)$ con $F(x+h) = G(x)$, è possibile trovare il valore di h senza conoscere la forma analitica delle funzioni? $F(x)$ e $G(x)$ non hanno una forma analitica nota a priori ma sono definite su un insieme di punti discreti. Queste due funzioni rappresentano il profilo di un particolare oggetto in due fotogrammi diversi. Il valore di h quantifica la differenza tra i due fermo-immagine.

Con un valore piccolo di h è possibile utilizzare l'approssimazione della derivata di $F(x)$ per ottenere un'approssimazione di h .

$$F'(x) \approx \frac{F(x+h) - F(x)}{h} = \frac{G(x) - F(x)}{h}$$
$$h \approx \frac{G(x) - F(x)}{F'(x)}$$

Questa tecnica elementare può essere utilizzata su tutti i punti del nostro campionamento di $F(x)$ e $G(x)$ in modo da ottenere la seguente stima di h :

$$h \approx \frac{\sum_x \frac{G(x) - F(x)}{F'(x)}}{\sum_x 1} \quad (2.6)$$

Questa stima può essere migliorata andando a pesare il contributo di ogni punto in funzione della qualità dell'approssimazione. La stima di h sarà infatti più precisa nei punti in cui il comportamento di $F(x)$ è approssimativamente lineare.

Sapendo che la derivata seconda di una funzione è un ottimo indicatore di linearità, è possibile dunque utilizzarla per pesare i contributi alla sommatoria in (2.6).

Ecco dunque i pesi:

$$w(x) = \frac{1}{|G'(x) - F'(x)|} \quad (2.7)$$

ottenuti eliminando il fattore h (supposto costante e piccolo) dalla stima della derivata seconda di $F(x)$:

$$F''(x) = \frac{G'(x) - F'(x)}{h}$$

In questo modo i punti con una derivata seconda prossima allo zero avranno un contributo maggiore di quelli con un'evidente non linearità. La sommatoria per la stima di h può dunque essere riscritta nel seguente modo:

$$h \approx \frac{\sum_x w(x) \frac{G(x) - F(x)}{F'(x)}}{\sum_x w(x)} \quad (2.8)$$

Per implementare praticamente questa stima di h si procede con un algoritmo iterativo.

$$h_0 = 0$$

$$h_{k+1} = h_k + \frac{\sum_x \frac{w(x)[G(x) - F(x+h_k)]}{F'(x+h_k)}}{\sum_x w(x)} \quad (2.9)$$

Il valore di h è inizialmente uguale a 0. Man mano che l'indice iterativo k cresce, diminuisce la differenza $G(x) - F(x+h_k)$. Questo rende sempre più piccole le modifiche di h man mano che l'indice k cresce. Le sommatorie \sum_x coinvolgono la totalità dei punti su cui sono definite $F(x)$ e $G(x)$. Questo metodo non è facile da applicare ai sistemi bidimensionali come le immagini. Fu dunque necessario trovare una strada alternativa che generasse gli stessi risultati.

La soluzione proposta da Lucas e Kanade consiste nell'andare a minimizzare rispetto a h il quadrato della differenza tra le due funzioni. In una dimensione il problema si formula nel seguente modo:

$$E = \sum_x [F(x+h) - G(x)]^2$$

$$\frac{\partial E}{\partial h} \approx \frac{\partial}{\partial h} \sum_x [F(x) + hF'(x) - G(x)]^2$$

$$\frac{\partial E}{\partial h} \approx \sum_x 2F'(x)[F(x) + hF'(x) - G(x)]$$

Andando a imporre la condizione $\frac{\partial E}{\partial h} = 0$ si ottiene :

$$\begin{aligned} \sum_x 2F'(x)[F(x) + hF'(x) - G(x)] &\approx 0 \\ \sum_x F'(x)[F(x) - G(x)] + \sum_x hF'(x)^2 &\approx 0 \\ h \sum_x F'(x)^2 &\approx \sum_x F'(x)[G(x) - F(x)] \\ h &\approx \frac{\sum_x F'(x)[G(x) - F(x)]}{\sum_x F'(x)^2} \end{aligned} \quad (2.10)$$

Questo metodo è sostanzialmente equivalente al precedente ma presenta dei vantaggi. Innanzitutto, questa forma di approssimazione lineare è generalizzabile a due dimensioni. Inoltre, in (2.10) avremmo il problema della divisione per 0 solo se $F'(x) = 0$ in ogni punto, mentre in (2.8) era sufficiente avere un unico punto x_k per cui $F'(x_k) = 0$ per avere il denominatore uguale a zero. La versione finale della successione che porta a calcolare h è quindi la seguente:

$$\begin{aligned} h_0 &= 0 \\ h_{k+1} &= h_k + \frac{\sum_x w(x)F'(x + h_k)[G(x) - F(x + h_k)]}{\sum_x w(x)F'(x + h_k)^2} \end{aligned} \quad (2.11)$$

con $w(x)$ dato da (2.7). Anche in questo caso all'aumentare dell'indice iterativo diminuisce la differenza $G(x) - F(x + h_k)$ rendendo sempre più piccolo l'incremento di h .

Per poter applicare questi metodi a dei veri problemi di riconoscimento delle immagini è necessario generalizzare al caso bidimensionale.

$$F(\vec{x} + \vec{h}) = F(\vec{x}) + \vec{h} \cdot \nabla F(\vec{x})$$

Come in precedenza la minimizzazione della distanza tra $F(\vec{x})$ e $G(\vec{x})$ porta alla seguente equazione:

$$\nabla_{\vec{h}} E \approx \nabla_{\vec{h}} \sum_{\vec{x}} [F(\vec{x}) + \vec{h} \cdot \nabla_{\vec{x}} F(\vec{x}) - G(\vec{x})]^2$$

$$\nabla_{\vec{h}} E \approx \sum_{\vec{x}} 2(\nabla_{\vec{x}} F(\vec{x}))^T [F(\vec{x}) + \vec{h} \cdot \nabla_{\vec{x}} F(\vec{x}) - G(\vec{x})]$$

Da qui è possibile ricavare la forma finale di h imponendo $\nabla_{\vec{h}} E = 0$:

$$\begin{aligned} \sum_{\vec{x}} 2(\nabla_{\vec{x}} F(\vec{x}))^T [F(\vec{x}) - G(\vec{x})] + \sum_{\vec{x}} 2(\nabla_{\vec{x}} F(\vec{x}))^T [\vec{h} \cdot \nabla_{\vec{x}} F(\vec{x})] &\approx 0 \\ \sum_{\vec{x}} 2(\nabla_{\vec{x}} F(\vec{x}))^T [\vec{h} \cdot \nabla_{\vec{x}} F(\vec{x})] &\approx \sum_{\vec{x}} 2(\nabla_{\vec{x}} F(\vec{x}))^T [G(\vec{x}) - F(\vec{x})] \\ \vec{h} \sum_{\vec{x}} 2(\nabla_{\vec{x}} F(\vec{x}))^T \nabla_{\vec{x}} F(\vec{x}) &\approx \sum_{\vec{x}} 2(\nabla_{\vec{x}} F(\vec{x}))^T [G(\vec{x}) - F(\vec{x})] \\ \vec{h} &\approx \left[\sum_{\vec{x}} (\nabla_{\vec{x}} F(\vec{x}))^T [G(\vec{x}) - F(\vec{x})] \right] \left[\sum_{\vec{x}} (\nabla_{\vec{x}} F(\vec{x}))^T (\nabla_{\vec{x}} F(\vec{x})) \right]^{-1} \end{aligned} \quad (2.12)$$

Questa tecnica diventa applicabile non solo alle traslazioni ma ad ogni tipo di trasformazione rappresentabile per via matriciale. Tutto quanto è stato detto finora rimane valido se la relazione tra le funzioni $F(\vec{x})$ e $G(\vec{x})$ è del tipo: $F(A\vec{x} + \vec{h}) = G(\vec{x})$. È necessario stimare dunque tre valori differenti: le due componenti di \vec{h} e A . Il metodo di Lucas e Kanade per il riconoscimento degli oggetti in fotogrammi diversi cerca di affrontare il problema tramite l'individuazione di un particolare profilo di intensità definito dalle funzioni $F(\vec{x})$ e $G(\vec{x})$. Questo tipo di approccio è detto Lagrangiano e ha costi computazionali molto elevati. L'algoritmo dell'Eulerian Video Magnification non necessita di individuare gli oggetti o di calcolare la derivata di una particolare funzione $F(\vec{x})$, il suo funzionamento è totalmente svincolato da ciò che compare nel video. Questo tipo di approccio è chiamato Euleriano. L'EVM non è sicuramente stato il primo tentativo di visualizzazione dei movimenti indipendentemente dal tracciamento degli oggetti. Lo studio del flusso ottico, che sarà effettuato nella prossima sezione, può essere considerato un antenato dell'EVM.

2.9 Flusso ottico

I metodi di elaborazione digitale delle immagini possono quindi essere utilizzati per evidenziare le informazioni contenute in un video. Alcuni algoritmi,

come quello di Canny, finalizzati al rilevamento dei contorni, non tengono conto del susseguirsi dei fotogrammi e lavorano su ogni immagine in maniera indipendente. Se invece si considera il susseguirsi dei fotogrammi è possibile estrapolare ulteriore informazione contenuta proprio nell'ordine della successione. Si supponga di disporre di un video girato con condizioni di luminosità costanti. In queste condizioni se il pixel (x, y) presenta tra il fotogramma t e il fotogramma $t + 1$ una significativa variazione di valore, questa sarà probabilmente dovuta al movimento repentino di un corpo o della posizione della telecamera. Se inoltre si vede che il pixel $(x, y + 4)_{t+1} \approx (x, y)_t$ allora è possibile ipotizzare che un corpo si sia spostato di 4 pixel verso l'alto tra un fotogramma e il successivo. Per cercare di analizzare quantitativamente queste informazioni contenute in un file video è utile parlare di flusso ottico. Questo concetto fu introdotto in campo biologico negli anni 50 da Gibson per provare a descrivere la visione degli insetti in volo. Il flusso ottico è un ottimo strumento per descrivere l'insieme delle velocità "apparenti" delle strutture solide che compaiono in un video. Si parla di velocità apparenti dato che il movimento individuabile osservando una successione di immagini può essere causato sia dall'effettiva traslazione di un corpo rigido, sia dal cambio di posizione dell'osservatore.

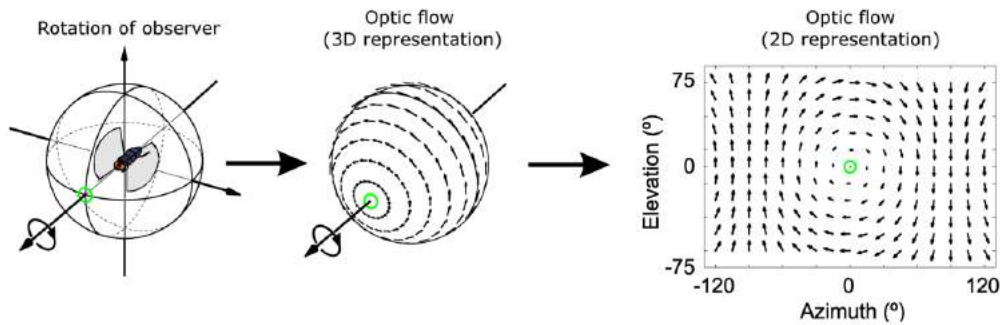


Figura 2.8: Un esempio di flusso ottico percepito da un osservatore in rotazione. La direzione e la dimensione delle frecce comunicano la direzione e l'intensità del flusso ottico in ogni punto. [15]

Per introdurre il flusso ottico da un punto di vista matematico si considera il movimento di un corpo che all'istante t si trova nella posizione (x, y) . Dopo

un istante di tempo Δt il punto si è spostato rispetto alla posizione iniziale. Chiamo $(x + \Delta x, y + \Delta y)$ la nuova posizione. Assumendo che l'esposizione alla luce sia la stessa nei due fotogrammi e nelle due posizioni è lecito supporre che anche il valore del pixel rappresentante il corpo in questione rimarrà costante. È possibile scrivere:

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \quad (2.13)$$

Se il video è registrato con un numero di fotogrammi al secondo sufficientemente elevato, è lecito supporre che il movimento in un tempo Δt sia piccolo e è quindi possibile sviluppare l'intensità al primo ordine.

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x} \Delta x + \frac{\partial I}{\partial y} \Delta y + \frac{\partial I}{\partial t} \Delta t \quad (2.14)$$

L'equazione (2.13) diventa dunque:

$$\begin{aligned} \frac{\partial I}{\partial x} \Delta x + \frac{\partial I}{\partial y} \Delta y + \frac{\partial I}{\partial t} \Delta t &= 0 \\ \frac{\partial I}{\partial x} \frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y} \frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t} \frac{\Delta t}{\Delta t} &= 0 \end{aligned}$$

che può essere scritta nel seguente modo:

$$\frac{\partial I}{\partial x} V_x + \frac{\partial I}{\partial y} V_y + \frac{\partial I}{\partial t} = 0 \quad (2.15)$$

V_x e V_y permettono di introdurre formalmente il flusso ottico. Queste rappresentano le componenti nelle due direzioni x e y . L'equazione (2.15) può essere riscritta in forma sintetica:

$$\vec{\nabla} I \cdot \vec{V} = -I_t \quad (2.16)$$

Le derivate dell'intensità dei pixel I_x , I_y e I_t possono essere calcolate numericamente ma le velocità dello spostamento del pixel in esame $V_x = \frac{\Delta x}{\Delta t}$ e $V_y = \frac{\Delta y}{\Delta t}$ rimangono incognite. Riscrivere in forma estesa l'equazione (2.16) aiuta a visualizzarne l'insieme delle soluzioni che, nel piano (V_x, V_y) , è rappresentabile come una retta. Ci sono dunque infinite soluzioni possibili e per determinare il flusso risulta necessario introdurre ulteriori vincoli.

$$V_y = \left(-\frac{I_x}{I_y}\right)V_x - \frac{I_t}{I_y}$$

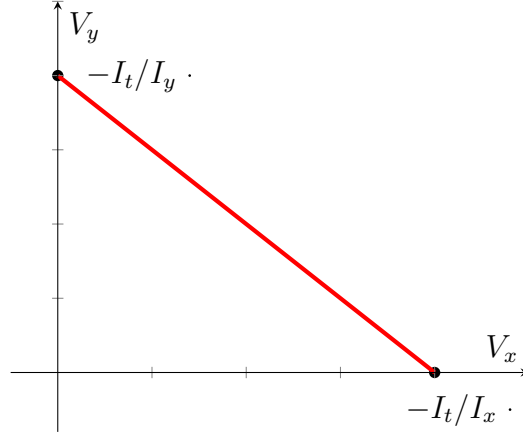


Figura 2.9: Insieme delle possibili soluzioni rappresentato nel piano $V_x ; V_y$

2.10 Calcolare il flusso ottico

Di fronte all'impossibilità di determinare una soluzione per le variabili V_x e V_y sono state inventate molte tecniche. Tutte queste devono necessariamente introdurre nuovi vincoli per poter trasformare l'equazione (2.15) in un sistema di almeno due equazioni. Una prima soluzione intuitiva e efficace suggeriva di considerare il flusso ottico costante in un intorno del pixel in analisi, in modo da poter disporre di un sistema di n equazioni, dove n rappresenta il numero di pixel considerati. La lettera q_i indica uno specifico pixel.

$$I_x(q_1)V_x + I_y(q_1)V_y = -I_t(q_1)$$

$$I_x(q_2)V_x + I_y(q_2)V_y = -I_t(q_2)$$

$$\vdots$$

$$I_x(q_n)V_x + I_y(q_n)V_y = -I_t(q_n)$$

Alla base di questa assunzione c'è l'ipotesi che il corpo in movimento occupi più di un singolo pixel, cosa molto plausibile specialmente con immagini a risoluzione elevata. Questo insieme di equazioni può essere riscritto in forma

compatta nel seguente modo $\hat{A}\vec{x} = \vec{b}$. \hat{A} è la matrice contenente le derivate spaziali della luminosità lungo l'asse x e y dei pixel q_i . \vec{b} invece è l'insieme delle derivate temporali (con il segno meno) della luminosità dei pixel q_i .

$$\begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix} \begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}$$

Nel caso in cui si decidesse di considerare esclusivamente due pixel q_1 e q_2 il sistema di equazioni qui sopra avrebbe una soluzione per le variabili V_x e V_y . La soluzione è anche visualizzabile graficamente dato che corrisponde all'intersezione delle due rette rappresentate in figura.

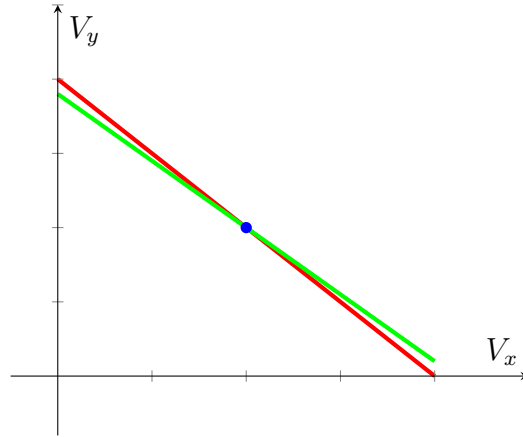


Figura 2.10: Insiemi delle possibili soluzioni. Con due equazioni si ottengono due rette che si incontrano in un unico punto. Il sistema è determinato e la soluzione è nota.

Se si decide di considerare un numero di pixel superiore a due allora il sistema di equazioni si traduce graficamente in un insieme di rette che non si intersecano in un unico punto. La soluzione a questa problematica è di tipo statistico e può essere trovata con il metodo dei minimi quadrati.

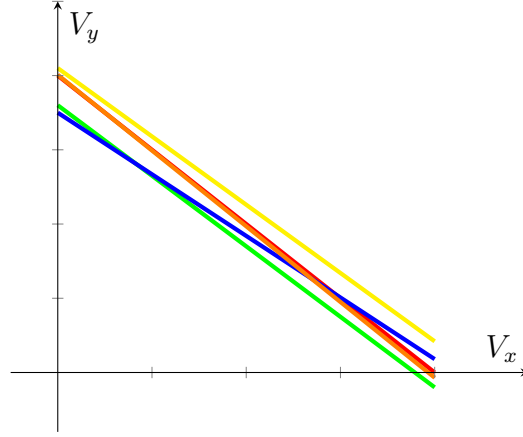


Figura 2.11: Rappresentazione grafica di un sistema lineare sovradeterminato. In questi casi bisogna procedere con un approccio statistico per trovare il punto che meglio rappresenta l'intersezione di tutte le rette.

2.11 Metodo di Horn e Schunck

Horn e Schunck nel 1981 proposero di utilizzare un metodo variazionale per stimare il flusso ottico Horn and Schunck [4]. Il funzionale da minimizzare dovrà sottostare all'equazione (2.15) e introdurre una nuova restrizione. L'ipotesi introdotta fu quella di un flusso regolare. L'operazione consiste nella minimizzazione della variazione delle componenti V_x e V_y . In termini pratici si aggiunge un vincolo ulteriore imponendo la minimizzazione del quadrato della ampiezza del gradiente di entrambe le componenti del flusso ottico. I termini da minimizzare sono i seguenti:

$$\left(\frac{\partial V_x}{\partial x}\right)^2 + \left(\frac{\partial V_x}{\partial y}\right)^2 \quad \left(\frac{\partial V_y}{\partial x}\right)^2 + \left(\frac{\partial V_y}{\partial y}\right)^2$$

Chiamo $J(V_x, V_y)$ il funzionale:

$$J(V_x, V_y) = \int \int [(I_x V_x + I_y V_y + I_t)^2 + \alpha^2 (|\nabla V_x|^2 + |\nabla V_y|^2)] dx dy \quad (2.17)$$

Con $\alpha = 0$ allora il funzionale soddisfa la condizione di minimo esattamente sulla retta rappresentata in figura 2.9. La definizione di $J(V_x, V_y)$ mostra

come α sia strettamente legato alla norma del gradiente. Si dice che esso è un fattore di regolarizzazione. Da questo punto in poi per brevità utilizzeremo la notazione (u, v) al posto della notazione (V_x, V_y) . Le equazioni di Eulero-Lagrange che permettono di minimizzare $J(u, v)$ sono le seguenti:

$$\frac{\partial J}{\partial u} - \frac{\partial}{\partial x} \left(\frac{\partial J}{\partial \frac{\partial u}{\partial x}} \right) - \frac{\partial}{\partial y} \left(\frac{\partial J}{\partial \frac{\partial u}{\partial y}} \right) = 0 \quad (2.18)$$

$$\frac{\partial J}{\partial v} - \frac{\partial}{\partial x} \left(\frac{\partial J}{\partial \left(\frac{\partial v}{\partial x} \right)} \right) - \frac{\partial}{\partial y} \left(\frac{\partial J}{\partial \left(\frac{\partial v}{\partial y} \right)} \right) = 0 \quad (2.19)$$

Applicando le equazioni (2.18) e (2.19) al funzionale (2.17) otteniamo:

$$I_x(I_x u + I_y v + I_t) - \alpha^2 \Delta u = 0 \quad (2.20)$$

$$I_y(I_x u + I_y v + I_t) - \alpha^2 \Delta v = 0 \quad (2.21)$$

che permetteranno di trovare il flusso ottico a condizione di poter calcolare numericamente i Laplaciani: $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$ e $\Delta v = \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}$. Si consideri l'intensità del pixel (i, j) al fotogramma k : $I(i, j, k)$. È possibile calcolarne le derivate parziali I_x , I_y e I_t come la media di 4 differenze di intensità:

$$I_x(i, j, k) = \frac{1}{4} (I(i, j+1, k) - I(i, j, k) + I(i+1, j+1, k) - I(i+1, j, k) + I(i, j+1, k+1) - I(i, j, k+1) + I(i+1, j+1, k+1) - I(i+1, j, k+1))$$

$$I_y(i, j, k) = \frac{1}{4} (I(i+1, j, k) - I(i, j, k) + I(i+1, j+1, k) - I(i, j+1, k) + I(i+1, j, k+1) - I(i, j+1, k+1) + I(i+1, j+1, k+1) - I(i, j+1, k+1))$$

$$I_t(i, j, k) = \frac{1}{4} (I(i, j, k+1) - I(i, j, k) + I(i+1, j, k+1) - I(i+1, j, k) + I(i+1, j+1, k+1) - I(i+1, j+1, k) + I(i, j+1, k+1) - I(i, j+1, k))$$

Per le dimensioni spaziali questo calcolo non è altro che l'unione di due convoluzioni. Il calcolo del Laplaciano di u e di v è più complesso. Considerando il pixel (i, j) e chiamando $u_{ij} = u(i, j)$ la componente orizzontale del flusso ottico nel punto (i, j) è possibile approssimare il Laplaciano $\nabla^2 u_{ij}$ nel seguente modo:

$$\nabla^2 u_{ij} \approx (\bar{u}_{ij} - u_{ij}) \quad (2.22)$$

con \bar{u}_{ij} calcolato come una media pesata del valore di u in un intorno del punto (i, j)

$$\bar{u}_{ij} = \frac{1}{6}(u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1}) + \frac{1}{12}(u_{i-1,j+1} + u_{i-1,j-1} + u_{i+1,j-1} + u_{i+1,j+1}) \quad (2.23)$$

Questa espressione tiene conto delle variazioni di u in tutte le direzioni dando maggior peso ai pixel adiacenti. In questo modo le equazioni (2.20) (2.21) diventano:

$$I_x(I_x u + I_y v + I_t) - (\bar{u} - u)\alpha^2 = 0 \quad (2.24)$$

$$I_y(I_x u + I_y v + I_t) - (\bar{v} - v)\alpha^2 = 0 \quad (2.25)$$

che possono essere riscritte nel seguente modo:

$$(\alpha^2 + I_x^2)u + I_x I_y v + I_x I_t - \alpha^2 \bar{u} = 0 \quad (2.26)$$

$$(\alpha^2 + I_y^2)v + I_x I_y u + I_y I_t - \alpha^2 \bar{v} = 0 \quad (2.27)$$

Queste equazioni sono lineari in $u = V_x$ e $v = V_y$ e permettono di calcolare il flusso ottico per ogni punto dell'immagine. Ciononostante, il valore del flusso in un punto dipende dal valore dei pixel adiacenti. Per questo motivo è necessario procedere in maniera iterativa dopo aver inizializzato i valori u_{ij}^0 e v_{ij}^0

$$u^{k+1} = \bar{u}^k - I_x \frac{I_x \bar{u}^k + I_y \bar{v}^k + I_t}{4\alpha^2 + I_x^2 + I_y^2} \quad (2.28)$$

$$v^{k+1} = \bar{v}^k - I_y \frac{I_x \bar{u}^k + I_y \bar{v}^k + I_t}{4\alpha^2 + I_x^2 + I_y^2} \quad (2.29)$$

Capitolo 3

Eulerian Video Magnification

Gli ultimi paragrafi del capitolo precedente hanno cercato di riassumere le tecniche finalizzate a individuare i movimenti in un video. In particolare, è stato introdotto il concetto di flusso ottico (*optical flow*) e sono state espone alcune tecniche elaborate nel corso degli anni per calcolarlo. La grande differenza tra il metodo di Lucas e Kanade e lo studio del flusso ottico è da ricercare nell'approccio all'individuazione dei movimenti. Il metodo di Lucas e Kanade individua gli oggetti che sono rappresentati da una successione di pixel la cui intensità è descritta dalla funzione $F(\vec{x})$. Il flusso ottico affronta il problema localmente senza dare importanza agli oggetti che compaiono nel video. Questi due approcci sono stati schematizzati nello studio della fluidodinamica. In questo capitolo sono presentate alcune tecniche di analisi dei fluidi che permettono di ampliare le prospettive sull'elaborazione digitale dei video.

3.1 Approccio Euleriano e approccio Lagrangiano.

Un fluido è un materiale continuo e deformabile. Si immagini adesso di voler descrivere la dinamica di un fluido all'interno di un contenitore (es: un tubo, una cisterna, un fiume...). Esistono due approcci principali a questo problema. Il primo, detto Lagrangiano, studia il moto delle singole parti-

celle. Il secondo è il metodo Euleriano, che prevede l'utilizzo di una griglia spaziale rispetto alla quale vengono calcolate le grandezze caratteristiche del sistema. Il metodo Lagrangiano studia quindi il comportamento di un fluido tracciando nel tempo ogni particella. Conoscendo posizione e velocità (di ogni particella ad ogni istante) diventa possibile calcolare altre grandezze (Temperatura, densità, pressione, viscosità, ...). L'approccio Euleriano non pretende di identificare e di seguire il movimento delle singole particelle ma utilizza funzioni definite sui punti di una griglia sovrapposta a una porzione di spazio-tempo all'interno della quale ha luogo l'evoluzione dinamica del sistema. La posizione non è più associata a una singola particella e dipendente dal tempo ma è una variabile indipendente rispetto alla quale vengono calcolate altre grandezze. Le informazioni che si hanno a disposizione in questo caso non rispecchiano direttamente il movimento del fluido ma permettono di ricavarlo. Nel caso della fluido-dinamica queste funzioni possono essere la densità, la temperatura, la pressione o la velocità.

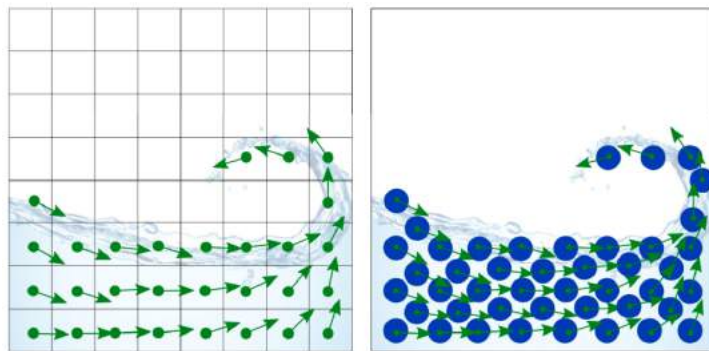


Figura 3.1: La stessa porzione di fluido analizzata con il metodo Euleriano (sinistra) e Lagrangiano (destra).

Questi due approcci possono essere applicati in diversi ambiti. È possibile studiare fenomeni naturali come l'evoluzione di uno stormo di uccelli o di un banco di pesci. Se si immagina di applicare queste due metodologie allo studio di un video in formato digitale è evidente che il metodo Euleriano è particolarmente adatto dal momento che ogni fotogramma non è altro che un insieme di pixel disposti a scacchiera. Ciononostante, a differenza dello

studio dei fluidi (ambito in cui è possibile tenere conto di un numero elevato di variabili), ogni pixel di ogni fotogramma fornisce esclusivamente tre dati relativi all'intensità dei tre colori principali (Rosso, Verde e Blu).

È possibile utilizzare anche metodi Lagrangiani per analizzare un video. Alcune tecniche viste nel capitolo 1 permettono infatti di individuare degli oggetti e di evidenziarne i contorni. Ogni corpo rigido viene dunque trattato indipendentemente dalla superficie (in pixel) che esso occupa. Queste tecniche incontrano però delle difficoltà dal momento che in un video un corpo rigido può deformarsi, cambiare aspetto o semplicemente svanire dietro a un altro oggetto più grande. I metodi Lagrangiani presentano anche delle difficoltà computazionali dato che necessitano del calcolo di termini di interazione tra pixel vicini che rendono molto più elevato il numero di operazioni da effettuare.

3.2 EVM

L'Eulerian Video Magnification (EVM) è un algoritmo di analisi dei video in formato digitale. È stato sviluppato nel 2012 a Boston da Wu et al. [17]. Lo scopo dell'EVM è quello di rendere visibili all'occhio umano le piccole oscillazioni non percepibili nel video originale. L'occhio umano non è capace di cogliere tutte le variazioni che ci possono essere nella luminosità dei pixel. Il nostro apparato visivo tende a identificare e a distinguere gli oggetti per focalizzarsi su di essi. Questo rischia di ridurre notevolmente le nostre capacità di cogliere i movimenti più lievi. L'EVM esegue una vera e propria amplificazione: il risultato finale è un nuovo video in cui si riescono a percepire a occhio nudo oscillazioni invisibili nel file originale. Per comprendere il funzionamento dell'EVM bisogna considerare che non è possibile tramite un video relazionarsi al fenomeno cui si è interessati e le uniche variabili a cui è possibile accedere sono quelle contenute all'interno dei valori dei pixel. In questo capitolo si analizzerà l'idea alla base dell'EVM, verranno introdotti gli aspetti teorici e cercheremo di comprendere come questi possano essere messi in pratica.

3.3 Teoria

In questa sezione sono presentate da un punto di vista numerico analitico le idee che stanno alla base dei processi Euleriani di amplificazione delle piccole oscillazioni. Per semplicità verrà presentato il caso monodimensionale. Un video uni-dimensionale non è altro che una retta di pixel a intensità variabile. Per rappresentare la situazione descritta è quindi sufficiente una funzione di due variabili $I(x, t)$. Ipotizzando che il video analizzato sia un video che contenga esclusivamente piccole oscillazioni sia nello spazio che nella luminosità, allora è lecito supporre che esso possa essere rappresentato nel seguente modo:

$$I(x, t) = f(x - \delta(t)) \quad (3.1)$$

con $I(x, 0) = f(x)$

Riscrivere in questo modo il segnale permette di visualizzare separatamente lo "sfondo" costante dalle variazioni temporali. Si definisce dunque $\hat{I}(x, t)$ il segnale in cui l'oscillazione è stata amplificata. L'obiettivo è proprio quello di calcolare, a partire dalla situazione iniziale $I(x, t)$, $\hat{I}(x, t)$:

$$\hat{I}(x, t) = f(x - (1 + \alpha)\delta(t))$$

È possibile approssimare l'uguaglianza (3.1) con un'espansione di Taylor al primo ordine:

$$I(x, t) \approx f(x) - \delta(t) \frac{\partial f(x)}{\partial x} \quad (3.2)$$

$B(x, t)$ è definita nel seguente modo:

$$B(x, t) = -\delta(t) \frac{\partial f(x)}{\partial x}$$

Si definisce quindi una funzione ausiliare $\tilde{I}(x, t)$ in cui al segnale originale è stato aggiunto il fattore $B(x, t)$ amplificato di un fattore α :

$$\tilde{I}(x, t) = I(x, t) + \alpha B(x, t) \quad (3.3)$$

È importante notare che questa funzione non coincide con $\hat{I}(x, t) = f(x - (1 + \alpha)\delta(t))$.

Lo sviluppo questa espressione, unito all'approssimazione di Taylor al primo ordine (3.2) appena vista, si ottiene:

$$\begin{aligned}\tilde{I}(x, t) &= f(x - \delta(t)) - \alpha\delta(t)\frac{\partial f(x)}{\partial x} \\ \tilde{I}(x, t) &\approx f(x) - \delta(t)\frac{\partial f(x)}{\partial x} - \alpha\delta(t)\frac{\partial f(x)}{\partial x} \\ \tilde{I}(x, t) &\approx f(x) - \frac{\partial f(x)}{\partial x}(1 + \alpha)\delta(t)\end{aligned}\tag{3.4}$$

Vediamo dunque che nella misura in cui il termine $(1 + \alpha)\delta(t)$ rimane "piccolo" è possibile considerare la funzione $\tilde{I}(x, t)$ come un'approssimazione di Taylor del segnale amplificato

$$\begin{aligned}\hat{I}(x, t) &= f(x - (1 + \alpha)\delta(t)) \\ \hat{I}(x, t) &\approx f(x) - \frac{\partial f(x)}{\partial x}(1 + \alpha)\delta(t)\end{aligned}\tag{3.5}$$

$$\hat{I}(x, t) \approx \tilde{I}(x, t)\tag{3.6}$$

È dunque stata trovata in $\tilde{I}(x, t)$ un'espressione che permette di approssimare il segnale amplificato:

$$\hat{I}(x, t) \approx I(x, t) + \alpha B(x, t)$$

Questo risultato è stato ottenuto supponendo che sia possibile creare un filtro che lasci passare le oscillazioni corrispondenti a una specifica frequenza. Nel caso in cui questo non fosse sufficiente a descrivere i movimenti caratterizzanti il video in input, è possibile selezionare da $\delta(t)$ una sommatoria di termini $\delta_k(t)$ contenenti le oscillazioni a varie specifiche frequenze. Ogni fattore $\delta_k(t)$ verrà smorzato dal filtro di un fattore γ_k . Il risultato del filtraggio $B(x, t)$ sarà:

$$B(x, t) = - \sum_k \gamma_k \delta_k \frac{\partial f(x)}{\partial x}\tag{3.7}$$

L'equazione (3.3) diventa quindi:

$$\tilde{I}(x, t) = I(x, t) - \alpha \sum_k \gamma_k \delta_k \frac{\partial f(x)}{\partial x} \quad (3.8)$$

Definendo $\alpha_k = \alpha \gamma_k$ si ottiene un risultato equivalente a quello in equazione (3.4):

$$\tilde{I}(x, t) \approx f(x) - \frac{\partial f(x)}{\partial x} \sum_k (1 + \alpha_k) \delta_k(t) \quad (3.9)$$

In questa circostanza vi è quindi un fattore di amplificazione che dipende dalla frequenza delle oscillazioni.

Il grafico in figura 3.2 mostra l'andamento delle principali funzioni trattate fino ad ora nel caso in cui $f(x) = \sin(x)$.

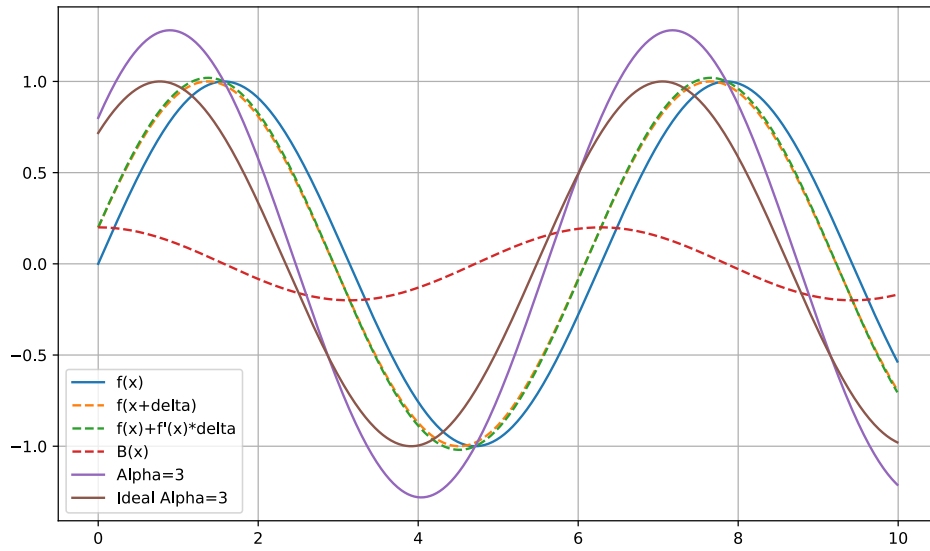


Figura 3.2

3.4 Problema dell'amplificazione del rumore

Purtroppo l'informazione contenuta nel valore dei pixel di un video include una percentuale di rumore che non può essere trascurata e che rischia di esse-

re dello stesso ordine di grandezza delle oscillazioni che vogliamo identificare e amplificare. Supponiamo che il rumore sia descritto da una distribuzione Gaussiana $N(x, t)$ con varianza σ^2 . La differenza tra due fotogrammi ha una varianza uguale a $2\sigma^2$. L'amplificazione del segnale causa anche l'amplificazione del rumore la cui varianza, a seguito dell'amplificazione, diventa uguale a $2\alpha^2\sigma^2$.

Il modo migliore per eliminare il rumore consiste nel conoscere con la maggior precisione possibile le frequenze ν che caratterizzano i movimenti da amplificare. Filtrando le frequenze temporali che caratterizzano i movimenti da amplificare con maggior precisione si riesce a eliminare quasi del tutto l'amplificazione del rumore.

3.5 Limiti

Si è visto che tra le condizioni necessarie all'implementazione dell'EVM vi è la richiesta di lavorare con piccole oscillazioni per preservare la validità dell'approssimazione di Taylor alla base del metodo utilizzato. In termini matematici questo si traduce in un valore limite del termine $\delta(t)(1+\alpha)$ oltre il quale l'approssimazione vista nell'equazione (3.5) non può essere considerata corretta. Questo valore limite non è però indipendente dalla funzione $f(x)$. Nei grafici riportati nella figura 3.3 è possibile vedere per diversi valori di α sia l'andamento di $\hat{I}(x, t) = f(x - (1 + \alpha)\delta(t))$ che l'andamento di $\tilde{I}(x, t) = I(x, t) + \alpha B(x, t)$.

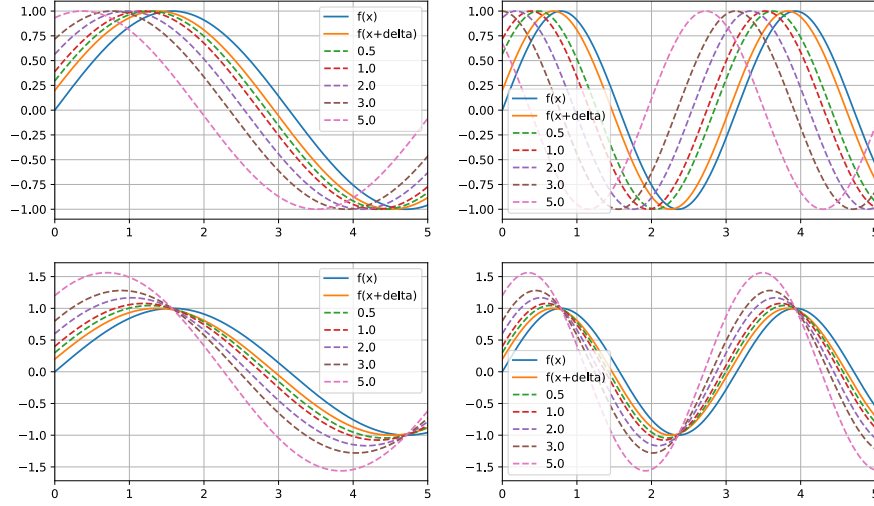


Figura 3.3: In alto sono rappresentate le funzioni $\sin(x)$ (a sinistra) e $\sin(2x)$ (a destra) con le rispettive amplificazioni $\hat{I}(x, t)$ per diversi valori di α . In basso le stesse funzioni sono affiancate all'approssimazione $\tilde{I}(x, t)$.

Con una frequenza più alta aumenta il divario tra $\hat{I}(x, t)$ e $\tilde{I}(x, t)$. Questa differenza è riportata in figura 3.4 per diverse funzioni sinusoidali caratterizzate da frequenze diverse.

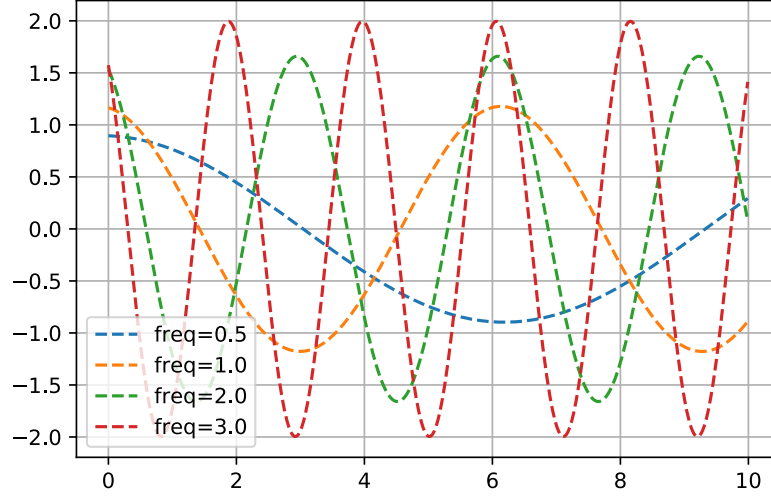


Figura 3.4: $\hat{I}(x, t) - \tilde{I}(x, t)$ con $\alpha = 2.0$. L'ampiezza dell'oscillazione di questa differenza aumenta con il crescere della frequenza.

E' dunque possibile cercare una relazione tra la funzione $f(x)$, che descrive il profilo di ciò che è rappresentato nel video, e il valore limite per il prodotto $\delta(t)(1 + \alpha)$.

Per comprendere a pieno questa relazione supponiamo che la funzione sia la seguente: $f(x) = \cos(\omega x)$. Per semplificare la trattazione definiamo $\beta = (1 + \alpha)$. L'equazione (3.4) unita all'equazione (3.6) rende possibile il calcolo di una valida approssimazione per il valore amplificato $\hat{I}(x, t)$:

$$\hat{I}(x, t) = \cos(\omega x + \beta \omega \delta(t)) \approx \cos(\omega x) - \beta \omega \delta(t) \sin(\omega x) \quad (3.10)$$

Il termine centrale in (3.10) può essere espanso seguendo le regole per il coseno di una somma.

$$\cos(\omega x) \cos(\beta \omega \delta(t)) - \sin(\omega x) \sin(\beta \omega \delta(t)) \approx \cos(\omega x) - \beta \omega \delta(t) \sin(\omega x)$$

Se ci si accontenta di una precisione del 10% sull'approssimazione è sufficiente supporre che il valore $\beta \omega \delta(t) \leq \frac{\pi}{4}$.

È possibile quindi approssimare le espressioni trigonometriche dei termini contenenti il fattore $\delta(t)$:

$$\cos(\beta\omega\delta(t)) \approx 1 \quad \sin(\beta\omega\delta(t)) \approx \beta\omega\delta(t) \quad (3.11)$$

Così facendo l'approssimazione in (3.10) diventa una vera e propria uguaglianza.

Vediamo come si concretizza sulla funzione $f(x)$ la richiesta di avere $\beta\omega\delta(t) \leq \frac{\pi}{4}$

$$\beta\omega\delta(t) \leq \frac{\pi}{4} \longrightarrow \beta \frac{2\pi}{\lambda} \delta(t) \leq \frac{\pi}{4} \longrightarrow (1 + \alpha)\delta(t) \leq \frac{\lambda}{8}$$

È importante notare che $\omega = \frac{2\pi}{\lambda}$ non contiene nessuna informazione sull'ampiezza delle oscillazioni temporali. La frequenza ω è una frequenza spaziale che descrive il profilo della funzione rappresentata nel video.

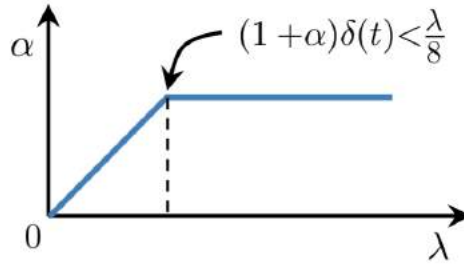


Figura 3.5: Il fattore di amplificazione α può essere attenuato in maniera graduale a partire da un certo valore della lunghezza d'onda λ

Nel caso monodimensionale è possibile scomporre in armoniche qualsiasi profilo $f(x)$:

$$f(x) \approx A_0 \sum_{n=1}^N A_n \sin(w_n t + \phi_n)$$

È quindi possibile selezionare un valore di α_n da associare a ogni frequenza ω_n in modo da preservare la validità della disequazione (3.11) rappresentata in figura 3.5.

Per poter rispettare fedelmente l'andamento proposto in figura 3.5 è necessario conoscere con esattezza l'ampiezza dello scostamento $\delta(t)$. Non

essendo questo possibile nella maggior parte dei casi si procede andando a diminuire progressivamente il fattore α man mano che la frequenza aumenta.

Nel caso di un video bidimensionale viene assegnato un valore di α diverso a ogni livello della piramide di Laplace (2.4). I livelli più alti della piramide (quelli con una minor risoluzione) sono quelli associabili a frequenze più alte. A loro verrà associato un fattore α minore.

Capitolo 4

Algoritmo

Gli autori dell'articolo [17] avevano programmato in MatLab l'algoritmo per l'Eulerian Video Magnification. La parte principale del lavoro di tesi prevedeva la programmazione di una versione in Python funzionante dell'EVM. Il codice che è stato scritto è diviso in sezioni abbastanza indipendenti tra loro che analizziamo in questo capitolo. Si vedrà inoltre come sia effettivamente possibile utilizzare metodi diversi per implementare alcuni passaggi. Le librerie che sono state usate nel codice sono: NumPy, SciPy, Matplotlib.plot e OpenCV (cv2). L'algoritmo esegue due operazioni principali:

- In un primo momento si estrapola da ogni singolo fotogramma la piramide (di Gauss o di Laplace (2.4)) corrispondente all'immagine raffigurata. Questa fase non tiene assolutamente conto dello scorrere del video e ogni fotogramma viene trattato indipendentemente. Le convoluzioni necessarie alla creazione delle piramidi sono le uniche operazioni in tutto il programma che relazionano i pixel adiacenti. Per questi motivi questa parte è identificata come la fase di elaborazione spaziale.
- L'altra parte del lavoro consiste nell'elaborazione temporale. In questa fase il valore dell'intensità di ogni pixel viene analizzato in funzione dello scorrere dei fotogrammi.

La combinazione di elaborazione spaziale e temporale è schematizzata in figura 4.1 ed è finalizzata a ottenere amplificazioni sia in termini di intensità

luminosa sia in termini di espansione del movimento.

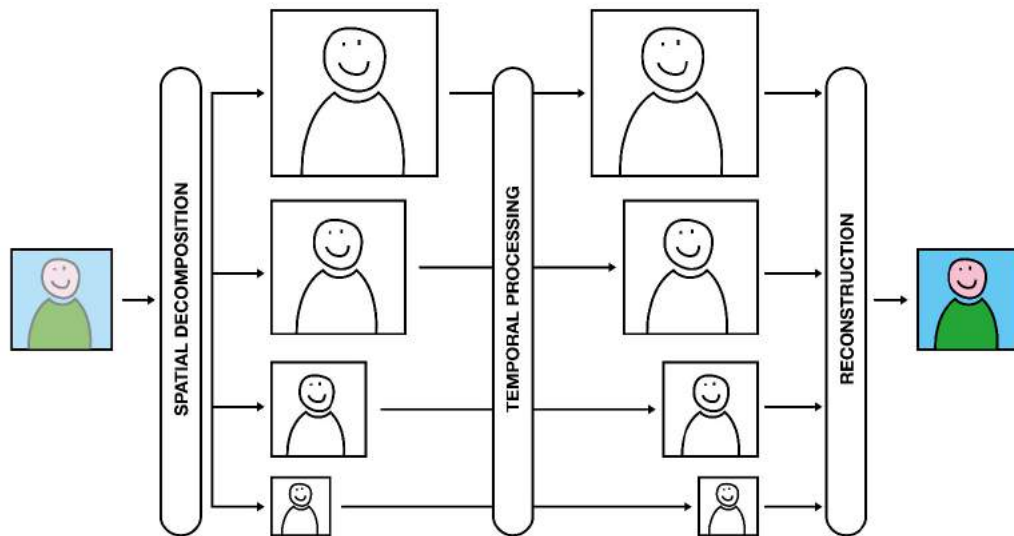


Figura 4.1: Un riassunto schematico del funzionamento dell'EVM. Il video viene scomposto in una piramide. Ogni pixel di ogni livello della piramide viene modificato in base all'evoluzione temporale descritta dallo spettro delle frequenze. L'ultima fase è quella della ricostruzione del video in cui i vari livelli della piramide vengono riassemblati.

4.1 GOOGLE COLAB PRO

Il lavoro di programmazione si è svolto quasi esclusivamente sulla piattaforma Google Colab nella versione PRO. Grazie a questa piattaforma è possibile eseguire codici direttamente in rete utilizzando la capacità di calcolo messa a disposizione da Google. In questo modo si hanno a disposizione 25GB di RAM che permettono di lavorare con variabili che occupano molta memoria. È sufficiente pensare che un video di 10 secondi registrato con un cellulare a 30 fotogrammi al secondo di dimensioni $[512, 512]$ richiede 1.887.436.944 bytes (1,9 GigaBytes) per essere memorizzato. Il motivo per cui vi è quest'enorme

differenza nella quantità di memoria necessaria a contenere il video in una matrice rispetto a salvarlo in formato .mp4 o .avi si nasconde nel fatto che non viene usata nessuna tecnica di compressione a differenza di quanto avviene nella memorizzazione dei file sul disco.

4.2 Importazione del video.

La prima fase è quella di importazione del video che viene eseguita tramite la libreria OpenCV. Ogni video viene inserito in un array quadridimensionale le cui dimensioni rappresentano rispettivamente: (Frame, Altezza, Larghezza, Colore).

```
cap=cv2.VideoCapture('videooriginale.mp4')
print('formato video=',type(cap))

frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
print('frame totali=' , frame_count)

width, height =
    int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)),int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
print('dimensioni=',width,height)

fps = int(cap.get(cv2.CAP_PROP_FPS))
print('fps=',fps)
```

In questo modo l'intensità associata a ogni colore di ogni pixel di ogni frame viene salvaguardata. Questa operazione semplicissima, come detto in precedenza, richiede molta RAM per essere effettuata. Con il video vengono importate alcune informazioni basilari ma essenziali per il proseguimento. Il numero di fotogrammi al secondo (fps), il numero di fotogrammi totali e le dimensioni del video. Questi dati permettono di creare l'array in cui viene inserito il video che per motivi pratici viene ritagliato in modo da essere in formato "quadrato" (1:1) con altezza e larghezza uguali a una potenza di due. (I video che sono stati utilizzati hanno una risoluzione di 512 x 512). Questa limitazione rende estremamente più semplice la fase iniziale relativa all'elaborazione spaziale del video.

```

video_tensor=np.zeros((300,512,512,3),dtype='float')
video_tensor_fake=np.zeros((frame_count,height,width,3),dtype='float')
count=0
while cap.isOpened():
    ret,frame=cap.read()
    if ret is True:
        video_tensor_fake[count,:,:,:]=frame
        count+=1
    else:
        break
video_tensor=video_tensor_fake[0:301,int((height-512)/2)...
...:-int((height-512)/2),int((width-512)/2):-int((width-512)/2),:]

```

4.3 Decomposizione spaziale

Una volta importato il video e trasferito all'interno di un array, comincia la prima parte dell'algoritmo. In questa fase vengono generate le piramidi di Gauss e di Laplace. Questa parte è di fondamentale importanza per rendere visibili i movimenti limitati nello spazio. Se l'algoritmo si limitasse ad amplificare l'ampiezza delle oscillazioni nel tempo dell'intensità dei colori, allora avremmo come risultato finale un video in cui i piccoli movimenti continuano ad essere localizzati nello spazio. L'unico effetto ottenuto sarebbe quello di una forte luminescenza in prossimità dei pixel cambianti nel tempo. L'elaborazione spaziale permette dunque di espandere letteralmente i movimenti su uno spazio maggiore. Nei casi più estremi è possibile distribuire su più pixel un movimento che nel video originale era percepibile e individuabile in un unico punto. L'oscillazione dell'intensità di un unico pixel può quindi essere distribuita su 32 pixel nel video finale. Dal punto di vista computazionale si procede andando a costruire la piramida Gaussiana e quella Laplaciana per ogni frame del video.

```

vg1=np.zeros((frame_count,int(alt/2),int(larg/2),3),dtype='float')
for i in range(300):
    vg1[i,:,:,:]=cv2.pyrDown(video_tensor[i,:,:,:])

```

```

vg2=np.zeros((frame_count,int(alt/4),int(larg/4),3),dtype='float')
for i in range(300):
    vg2[i,:,:,:]=cv2.pyrDown(vg1[i,:,:,:])

vg3=np.zeros((frame_count,int(alt/8),int(larg/8),3),dtype='float')
for i in range(300):
    vg3[i,:,:,:]=cv2.pyrDown(vg2[i,:,:,:])

vg4=np.zeros((frame_count,int(alt/16),int(larg/16),3),dtype='float')
for i in range(300):
    vg4[i,:,:,:]=cv2.pyrDown(vg3[i,:,:,:])

vg5=np.zeros((frame_count,int(alt/32),int(larg/32),3),dtype='float')
for i in range(300):
    vg5[i,:,:,:]=cv2.pyrDown(vg4[i,:,:,:])

piramidegaussiana=[video_tensor,vg1,vg2,vg3,vg4,vg5]

```

Questa operazione richiede nuovamente molta memoria dato che vengono creati 5 array che complessivamente occupano quasi la stessa quantità di memoria del video originale. A questo punto, partendo dalla piramide Gaussiana, viene creata la piramide di Laplace con uno strato in meno. Si procede nel seguente modo:

```

pirlaplace=[np.zeros((301,512,512,3))]
pirlaplace.append(np.zeros((301,256,256,3)))
pirlaplace.append(np.zeros((301,128,128,3)))
pirlaplace.append(np.zeros((301,64,64,3)))
pirlaplace.append(np.zeros((301,32,32,3)))
for j in range(301):
    for i in range(5,0,-1):
        gpext=cv2.pyrUp(piramidegaussiana[i][j,:,:,:])
        pirlaplace[i-1][j]=cv2.subtract(piramidegaussiana[i-1][j],gpext)

```

Si effettua un ciclo su ogni fotogramma e si utilizza una variabile provvisoria. Numerando gli "strati" delle piramidi da 0 a n allora avremmo che lo strato i della piramide di Laplace sarà dato dalla differenza tra lo strato i della piramide Gaussiana e lo strato $i + 1$ riportato a una risoluzione superiore. Quest'ultima operazione si effettua con la funzione `cv2.PyrUp` del pacchetto `OpenCv` e consiste nei seguenti passaggi.

- Aggiungere tra le righe delle righe uguali a 0 e tra le colonne delle colonne uguali a 0.
- Effettuare una convoluzione Gaussiana per distribuire l'intensità dei pixel diversi da zero.

La convoluzione nella funzione `cv2.PyrUp()` differisce da quella vista nel Capitolo 1 di un fattore finalizzato a conservare l'intensità media dei pixel. Il nuovo kernel sarà dunque:

$$\frac{1}{64} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Ovviamente, se su un'immagine si utilizza prima la funzione `cv2.PyrDown()` e poi la funzione `cv2.PyrUp()` non si ritorna all'immagine iniziale. Questo è dovuto al fatto che nella fase di "downsizing" vi è un'irreversibile perdita di informazione. Come è possibile vedere la piramide di Laplace tende a evidenziare i "bordi" degli oggetti. Questa particolarità è deducibile dal momento che è stata creata come differenza tra due immagini uguali ma con risoluzione differente.



Figura 4.2: Da sinistra: Un fotogramma del video originale, lo stesso fotogramma rappresentato nel primo strato della piramide di Laplace e nel quarto e ultimo strato.

Avendo generato la piramide per ogni frame, si hanno ora a disposizione n versioni diverse dello stesso video con risoluzioni decrescenti. Ogni versione del video è utilizzata dall'algoritmo per amplificare i movimenti o i cambi di colore.

4.4 Elaborazione temporale.

La seguente parte del codice è sicuramente meno intuitiva ma permette di amplificare le variazioni di intensità di un singolo pixel. Queste hanno un legame con le oscillazioni spaziali presenti nel video e riuscire a individuarle su ogni strato della piramide per poi amplificarle è l'obiettivo di questa sezione. Questa operazione, che matematicamente consiste nell'effettuare una trasformata di Fourier, è limitata dal fatto che non disponiamo di una funzione analitica che rappresenti l'andamento dell'intensità dei pixel. E' necessario dunque utilizzare le intensità rilevate ad ogni singolo frame per campionare il segnale e analizzarlo. Dal punto di vista computazionale questo si traduce nelle seguenti istruzioni di codice: si procede con un ciclo sui livelli della piramide con la quale si sta lavorando. Per fortuna non è neces-

sario effettuare cicli su ogni singolo colore di ogni singolo pixel. La funzione `fft` accetta infatti come argomento un array pluridimensionale a condizione che sia specificato l'asse sul quale si vuole effettuare l'analisi di Fourier. Si genera dunque un nuovo array a 4 dimensioni per ogni livello della piramide. La funzione `fftpack.fftfreq` permette di generare l'asse delle frequenze in modo che si possa associare ad ogni intensità generata dalla funzione `fft` la frequenza giusta. L'asse delle frequenze viene generato a partire dal numero di fotogrammi totali e dal numero di fotogrammi al secondo. Per un video di 10 secondi girato a 30 fotogrammi al secondo (300 fotogrammi totali) avrà la seguente forma:

$$[0.0|0.1|\dots|14.9|15.0|14.9|\dots|0.2|0.1] \quad (4.1)$$

Lo spettro generato da `fft` è simmetrico e si adatta perfettamente a l'asse (4.1)

```

for i in range(1,5):
    fs=piramide[i].shape[0]
    fps=30
    t=np.arange(0,10,1/fps)
    media=np.mean(piramide[i][:,:,:,:],axis=0)
#=====
    if (i==1):
        listamedia=[media]
    else:
        listamedia.append(media)
#=====
    film_scalato=np.subtract(piramide[i],media)
    assefreq = fftpack.fftfreq(film[i].shape[0], d= 1.0/fps)
    spettro=fft(film_scalato,axis=0)
    if (i==1):
        lista_spettri=[spettro]
    else:
        lista_spettri.append(spettro)

```

Come si vede nel codice, prima di effettuare l'analisi spettrale ogni pixel viene scalato del proprio valore mediato lungo il tempo in modo da oscillare intorno allo zero. Questa è un'operazione necessaria per assicurarsi che lo spettro delle frequenze non presenti una componente in prossimità dello zero

elevatissima che renda difficilmente identificabili le oscillazioni interessanti da individuare. Il limite nel campionare le frequenze è dato dal teorema di Nyquist [10]: se il video è registrato con una frequenza di n fotogrammi al secondo allora sarà possibile identificare le oscillazioni fino a una frequenza massima di $\frac{n}{2}$ Hz.

4.5 Filtro passa banda

A questo punto si dispone dello spettro relativo all'andamento dell'intensità di ogni colore di ogni pixel di ogni livello della piramide del video di partenza. Il prossimo passo consiste nel filtrare questo insieme di frequenze per poi generare un nuovo segnale. Esistono diversi tipi di filtri possibili ma il più efficiente ed efficace per individuare i moti specifici a una tipica frequenza è il filtro "passa banda". Questo si limita ad azzerare l'insieme delle frequenze al di fuori di un intervallo deciso a priori. Usare un filtro di questo tipo presuppone che si conosca la frequenza dell'oscillazione che si vuole amplificare. Questa richiesta non è eccessivamente limitante. Se fossimo ad esempio interessati a evidenziare i cambiamenti del colore cutaneo dovuti al battito cardiaco potremmo conservare esclusivamente le frequenze comprese tra 0.5Hz e 2Hz. In questo specifico caso si azzerà il 90% dello spettro nel caso in cui il video originale sia stato girato a 30 fotogrammi al secondo. Per individuare le frequenze da azzerare il programma utilizza l'asse (4.1). Le variabili `bound_low` e `bound_high`, generate a partire dalle indicazioni fornite al programma sulla frequenza dell'oscillazione che stiamo cercando di amplificare, contengono l'indicizzazione da utilizzare per azzerare le componenti dello spettro.

```
bound_low = (np.abs(assefreq - low)).argmin()
bound_high = (np.abs(assefreq - high)).argmin()
for j in range(4):
    spettro_filtrato=np.copy(lista_spettri[j])
    spettro_filtrato[:,bound_low,:,:,:] = 0
    spettro_filtrato[bound_high:-bound_high,:,:,:] = 0
    spettro_filtrato[-bound_low,:,:,:] = 0
    if (j==0):
```

```
    lista_spettri_filt=[spettro_filtrato]
else:
    lista_spettri_filt.append(spettro_filtrato)
```

4.6 Nuovo segnale

Lo spettro filtrato può essere *antitrasformato* per generare un nuovo segnale partendo esclusivamente dalle frequenze che sono state selezionate. Per ogni colore di ogni pixel di ogni strato della piramide si crea dunque un nuovo andamento nel tempo dell'intensità luminosa.

```
for i in range(4):
    segnale_filtrato=fftpack.iffpack(lista_spettri_filt[i],axis=0)
    lista_spettri_filt[i]=segnale_filtrato
```

Il segnale viene memorizzato nella variabile che conteneva lo spettro filtrato. Quest'ultimo non viene conservato dal programma per non occupare memoria inutilmente.

4.7 Amplificazione del segnale

Il segnale ottenuto può essere amplificato moltiplicandolo per il fattore α . Ovviamente non tutti i pixel del video sono caratterizzati da un'oscillazione che rientra nel range di frequenze da amplificare. È sufficiente immaginare un semplice video raffigurante un pendolo che compie impercettibili oscillazioni: solo i pochi pixel in prossimità dei bordi laterali del pendolo sono effettivamente da amplificare. A questo punto si può procedere in due modi:

- Amplificare il segnale in maniera uniforme per tutti i pixel di un determinato livello della piramide che stiamo utilizzando.
- Amplificare esclusivamente il segnale dei pixel che presentano un picco nella regione dello spettro delle frequenze selezionata.

Nel primo caso si rischia di produrre un'eccessiva amplificazione del rumore mentre nel secondo caso si rischia di non riuscire a ottenere l'effetto finale desiderato. Per non creare confusione il nome della variabile `lista_spettri_filt` è stato cambiato in `lista_tensori_filt` dato che non contiene più il risultato dell'analisi spettrale.

4.7.1 Amplificazione totale

Se si decide di procedere all'amplificazione del segnale di ogni pixel è sufficiente moltiplicare per un fattore α_i il video contenuto nel livello i -esimo della piramide.

```
lista_tensori_filt[0] = ampli0*lista_tensori_filt[0]
lista_tensori_filt[1] = ampli1*lista_tensori_filt[1]
lista_tensori_filt[2] = ampli2*lista_tensori_filt[2]
lista_tensori_filt[3] = ampli3*lista_tensori_filt[3]
```

I fattori α_i (nel programma `ampli0, ampli1, ampli2` e `ampli3`) sono diversi da un livello all'altro e devono soddisfare per i motivi visti in 3.5 la seguente condizione:

$$\alpha_3 \leq \alpha_2 \leq \alpha_1 \leq \alpha_0$$

4.7.2 Amplificazione parziale

Se invece si preferisce amplificare solo i pixel contenenti il movimento cui si è interessati, bisogna trovare un modo per selezionarli. È possibile procedere con modalità diverse. In questo lavoro la selezione è stata fatta confrontando (per ogni pixel di ogni livello della piramide) il valore massimo nello spazio delle frequenze con la media del valore assoluto dello spettro. Se il massimo differisce dalla media di un valore superiore a $n\sigma$ allora il pixel presenta molto probabilmente un'oscillazione interessante da amplificare. Il valore di σ non è altro che la deviazione standard del valore assoluto dei valori contenuti nello spettro delle frequenze. Il valore n è variabile e dipende sia dal video di partenza sia dal risultato che si vuole ottenere. Questa tecnica si basa sul fatto che un rumore generato con una distribuzione gaussiana

abbia anche nel dominio delle frequenze una distribuzione casuale. Il 99,73% delle componenti dovute al rumore rientrerà dunque nell'intervallo $\mu + 3\sigma$, con μ uguale alla media del valore assoluto dei valori dell'array contenente le frequenze. È quindi necessario associare a ogni pixel un valore di α su misura. Il codice riportato mostra che il valore α è stato sostituito da un insieme di 4 matrici. Ogni matrice è associata a un livello e contiene un fattore di amplificazione diverso per ogni colore di ogni pixel. Le matrici di amplificazione sono inizializzate a zero. Viene quindi calcolata la media dello spettro per ogni punto della piramide del video. La condizione contenuta nel `if` è la seguente: Per un determinato colore di un determinato pixel, se almeno un punto nel dominio delle frequenze è maggiore della somma tra la media delle frequenze e σ (`sigg`) deviazioni standard allora il pixel è da amplificare. I risultati visivamente migliori si ottengono con un valore di σ compreso tra 4 e 5.

```
matamp0=np.zeros((256,256,3))
matamp1=np.zeros((128,128,3))
matamp2=np.zeros((64,64,3))
matamp3=np.zeros((32,32,3))
mediaspettro=[np.mean(abs(lista_spettri[0][0:150,:,:,:]),axis=0)]
mediaspettro.append(np.mean(abs(lista_spettri[1][0:150,:,:,:]),axis=0))
mediaspettro.append(np.mean(abs(lista_spettri[2][0:150,:,:,:]),axis=0))
mediaspettro.append(np.mean(abs(lista_spettri[3][0:150,:,:,:]),axis=0))

sigg=8.0

for i in range(256):
    for j in range(256):
        for k in range(3):
            if(np.any(abs(lista_spettri[0][0:150,i,j,k])>(mediaspettro[0][i,j,k]+
+sigg*np.std(abs(lista_spettri[0][0:150,i,j,k]))))):
                matamp0[i,j,k]=amplificazione
for i in range(128):
    for j in range(128):
        for k in range(3):
            if(np.any(abs(lista_spettri[1][0:150,i,j,k])>(mediaspettro[1][i,j,k]+
+sigg*np.std(abs(lista_spettri[1][0:150,i,j,k]))))):
                matamp1[i,j,k]=amplificazione*0.8
for i in range(64):
```

```

for j in range(64):
    for k in range(3):
        if(np.any(abs(lista_spettri[2][0:150,i,j,k])>(mediaspettro[2][i,j,k]+
+sigg*np.std(abs(lista_spettri[2][0:150,i,j,k]))))):
            matamp2[i,j,k]=amplificazione*0.6
for i in range(32):
    for j in range(32):
        for k in range(3):
            if(np.any(abs(lista_spettri[3][0:150,i,j,k])>(mediaspettro[3][i,j,k]+
+sigg*np.std(abs(lista_spettri[3][0:150,i,j,k]))))):
                matamp3[i,j,k]=amplificazione*0.4

lista_tensori_filt[0]=matamp0*lista_tensori_filt[0]
lista_tensori_filt[1]=matamp1*lista_tensori_filt[1]
lista_tensori_filt[2]=matamp2*lista_tensori_filt[2]
lista_tensori_filt[3]=matamp3*lista_tensori_filt[3]

```

4.8 Ricostruzione video finale

A questo punto rimane esclusivamente la fase di *riassemblaggio* del video. Ogni fotogramma del video finale sarà dato da quell'operazione che può essere visualizzata immaginando di schiacciare la piramide. I vari strati vengono riportati alle dimensioni originali ([512;512]) per poi essere sommati tra loro. L'andamento finale di ogni pixel riceve dunque il contributo dell'amplificazione di ogni strato della piramide.

```

final_video=np.zeros((300,512,512,3))
for i in range(300):
    frame_finale=cv2.pyrUp(np.real(lista_tensori_filt[0][i,:,:,:]))
    frame_finale=frame_finale+
    +cv2.pyrUp(cv2.pyrUp(np.real(lista_tensori_filt[1][i,:,:,:]))
    frame_finale=frame_finale+
    +cv2.pyrUp(cv2.pyrUp(cv2.pyrUp(np.real(lista_tensori_filt[2][i,:,:,:]))))
    frame_finale=frame_finale+cv2.pyrUp(
    cv2.pyrUp(cv2.pyrUp(cv2.pyrUp(np.real(lista_tensori_filt[3][i,:,:,:]))))
    frame_finale = frame_finale + video_tensor[i,:,:,:]
    final_video[i]=frame_finale

```

Il video finale è quindi contenuto in un array delle stesse dimensioni di quello in cui era contenuto il video iniziale. Per trasformarlo in un file .avi utilizziamo nuovamente la libreria OpenCV che aveva reso possibile l'operazione inversa all'inizio della programmazione.

```
fourcc = cv2.VideoWriter_fourcc('M', 'J', 'P', 'G')
[height,width]=final_video[0].shape[0:2]
writer = cv2.VideoWriter('nome_del_video'+'.avi', fourcc, 30,
    (width, height), 1)
for i in range(0,final_video.shape[0]):
    writer.write(cv2.cvtColor(final_video[i]))
writer.release()
```

Capitolo 5

Esperimenti iniziali

5.1 Video artificiali

Con l'obiettivo di riuscire ad analizzare al meglio i risultati ottenuti con il codice visto nel capitolo precedente, in questa sezione si analizzeranno gli effetti ottenuti su alcuni video molto semplici creati artificialmente tramite la libreria OpenCV. Questo renderà possibile quantificare gli effetti di amplificazione ottenuti e di confrontarli con l'inevitabile amplificazione del rumore. I video utilizzati hanno tutti la stessa risoluzione ($[512;512]$) e un numero di frame al secondo uguale a 30. Gli esempi mostrati sono stati pensati e generati con l'obiettivo di riassumere un maggior numero di situazioni possibili. Tra i video generati artificialmente troviamo:

- I movimenti impercettibili di un quadrato. L'immagine del quadrato è generata grazie alla funzione `cv2.rectangle()`. Ogni pixel interno al quadrato è settato nel inizializzato nel seguente modo: $[0;0;255]$ (Un colore alla massima intensità e gli altri 2 spenti). Si procede quindi effettuando una convoluzione con un filtro uniforme che sfuma i bordi del quadrato. I singoli frame vengono generati andando ad aumentare a fasi alterne l'intensità delle sfumature del bordo inferiore e del bordo superiore. Lo spostamento non può essere simulato scalando il quadrato di un singolo pixel, dato che questo tipo di movimento è percepibile

anche ad occhio nudo e non aiuta a valutare l'efficacia dell'algoritmo sui movimenti molto piccoli.

- Un quadrato i cui movimenti, generati come al punto 1, saranno casuali nelle 4 direzioni. In questo caso non è quindi possibile individuare una frequenza ν che sia caratteristica del moto.
- Video in cui tutti i pixel hanno la stessa intensità luminosa. Questa varia leggermente nel tempo per poter valutare l'efficacia dell'EVM anche con i cambi di colore.
- Un quadrato creato come quello presentato nel primo punto. In questo caso però l'oscillazione del bordo superiore e di quello inferiore sono caratterizzate da frequenze diverse. È possibile amplificare esclusivamente una delle due oscillazioni?
- Infine si utilizzerà un video in cui l'oscillazione è limitata a un'unica colonna di pixel per verificare l'ampiezza dell'amplificazione nel risultato finale.

5.2 Oscillazioni cromatiche

Il primo video sottoposto all'algoritmo è formato da pixel tutti identici in ogni fotogramma. Il video è creato digitalmente e rappresenta una situazione indubbiamente poco realistica. Ciononostante, è utile per capire come si comporta l'EVM rispetto ai cambi di colore. Il programma viene eseguito una volta utilizzando la piramide Gaussiana e una volta usando la piramide Laplaciana. L'assenza di contorni nel susseguirsi dei fotogrammi fa sì che i diversi livelli della piramide di Laplace siano costantemente tutti uguali tra loro e uguali a 0. Questo fa sì che il risultato finale sia totalmente uguale a quello iniziale. La piramide di Gauss permette invece di ottenere un'amplificazione.

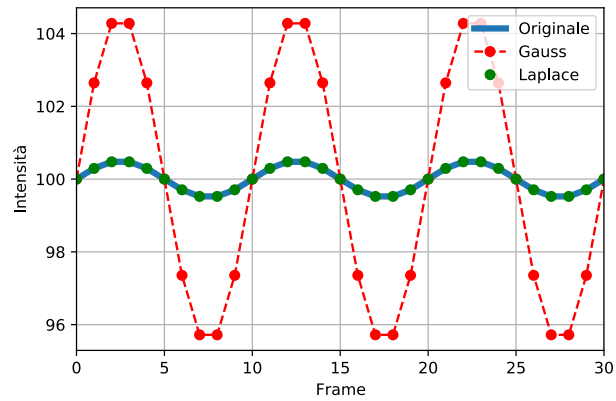


Figura 5.1: L'andamento nel tempo dell'intensità dei pixel nel video originale, nella versione amplificata con la piramide di Gauss e in quella amplificata con la piramide di Laplace.

Per visualizzare il risultato dell'amplificazione ecco un'immagine creata affiancando una riga di pixel di ogni fotogramma. In questo modo, l'asse delle ordinate rappresenta l'evoluzione temporale, mentre le ascisse continuano a descrivere l'andamento spaziale dell'immagine.

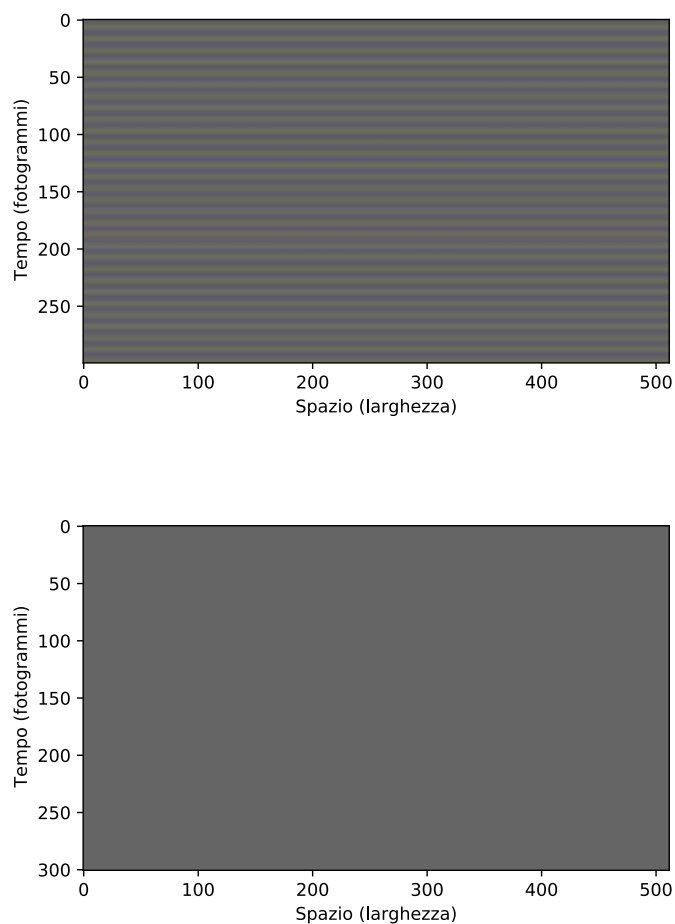


Figura 5.2: L'immagine in alto, amplificata con la piramide di Gauss, permette di visualizzare variazioni di intensità invisibili nel video originale e nell'amplificazione Laplaciana rappresentata in basso.

Anche se è stata analizzata una situazione ideale (mancanza di rumore e totale uniformità nella colorazione), queste prime considerazioni portano a concludere che l'amplificazione delle oscillazioni della colorazione di corpi statici deve essere effettuata con la piramide Gaussiana.

5.3 Piccole oscillazioni armoniche.

Il secondo video analizzato è quello che presenta un quadrato che oscilla con un moto armonico. Nella figura successiva sono rappresentati due frame nei quali il quadrato si trova agli estremi opposti dell'oscillazione. Non è possibile notare nessuna differenza né guardando i due fermi immagine né riproducendo il video.

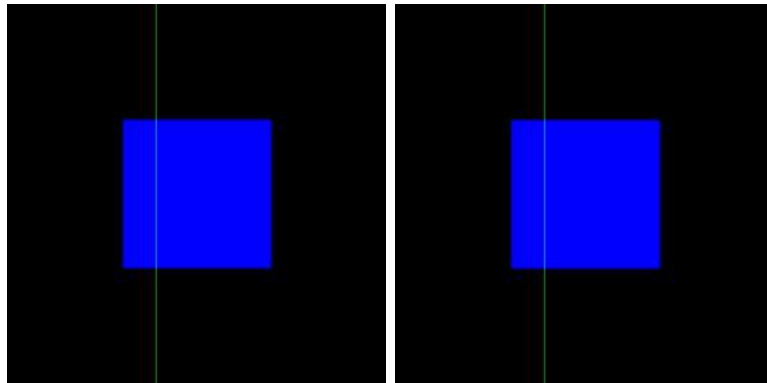


Figura 5.3: A sinistra il quadrato blu è leggermente spostato verso l'alto, a destra verso il basso. La linea verde non è parte del fotogramma originale ma è stata aggiunta come riferimento per il grafico mostrato in figura 5.4

Per visualizzare la differenza tra i due fotogrammi ecco un grafico dell'andamento dell'intensità del blu lungo la linea verde visibile in figura 5.3

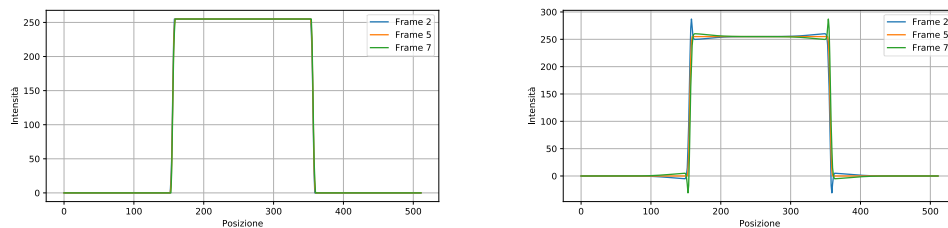


Figura 5.4: Il grafico rappresenta, per tre fotogrammi diversi, l'intensità del blu lungo l'intera colonna evidenziata in verde in figura 5.3. A sinistra è rappresentato il video originale e non vi è nessuna differenza apprezzabile, a destra il risultato finale mostra grandi differenze tra i fotogrammi soprattutto nelle zone di cambiamento.

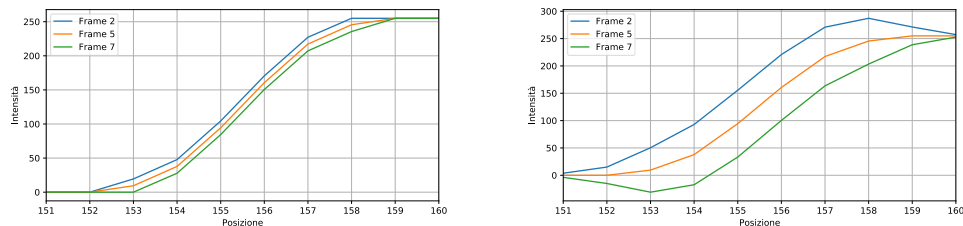


Figura 5.5: In questo ingrandimento della figura 5.4 è possibile vedere che i 3 frame rappresentanti gli estremi e il centro delle oscillazioni presentano 6 pixel la cui intensità differisce. Nel video originale (sinistra) questa differenza è molto discreta e dell'ordine di una decina. A destra invece vediamo che la differenza diventa dell'ordine di 50.

Non sorprende che la differenza sia minima nel video originale, dato che è stato volontariamente creato in maniera controllata. Nel grafico in figura 5.6 è mostrato l'andamento dell'intensità di un pixel appartenente al bordo oscillante del quadrato.

Se il video viene elaborato dall'algoritmo, i cambiamenti sono evidenti. La riproduzione del video ottenuto come output permette già di apprezzare i

risultati. I grafici mostrati sono creati partendo da un output creato da una piramide di Laplace in cui ogni strato è stato uniformemente amplificato di un fattore 5. È stato usato un filtro passa banda tra 2Hz e 4Hz, dato che la frequenza iniziale di oscillazione era di 3Hz.

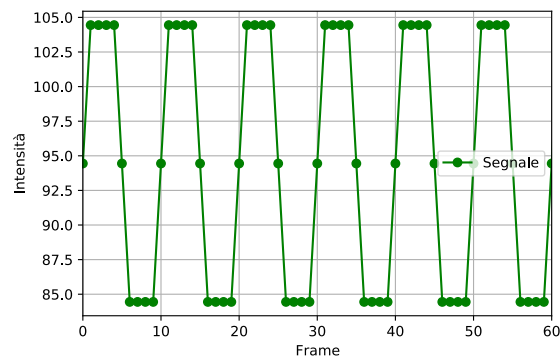


Figura 5.6: Andamento dell'intensità del blu per il pixel $[155; 256]$ nei primi due secondi del video originale. Sono visibili 60 fotogrammi.

Il programma riesce quindi a creare una differenza non solo in termini di intensità ma anche in termini di dimensioni. Ecco l'andamento (lungo la linea verde in figura 5.3) della differenza di intensità tra fotogrammi diversi. Il grafico in figura 5.7 permette di apprezzare la dipendenza del risultato dalle variazioni del fattore α .

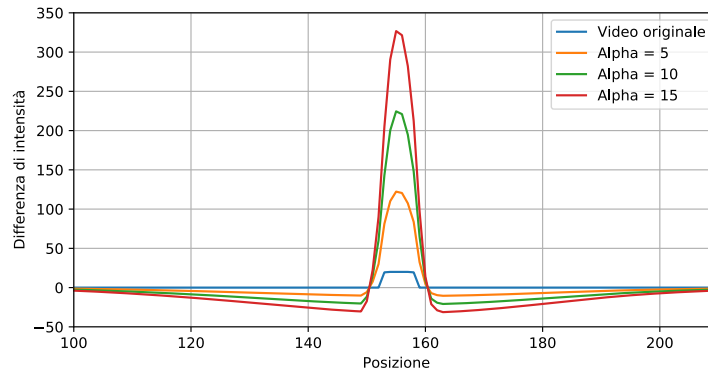


Figura 5.7: Differenza di intensità lungo la direzione verticale tra i fotogrammi 2 e 7. In blu è rappresentata la situazione presente nel video originale. Gli altri grafici mostrano il risultato per diversi fattori di α . È importante ricordare che i valori dei pixel oscillano tra 0 e 255. Se il segnale amplificato si estende al di là di questi limiti, il programma arrotonda automaticamente il valore del pixel.

Il segnale finale è ottenuto sommando le amplificazioni di ogni singolo strato di Laplace. In figura 5.8 si vede come ogni livello della piramide vada a comporre il risultato finale.

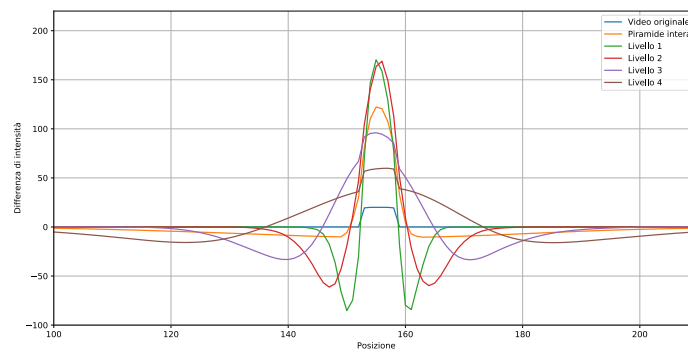


Figura 5.8: Andamento lungo la direzione verticale dell'intensità di ogni livello della piramide. In giallo è visibile il segnale finale diviso per 4. Ad ogni singolo livello viene aggiunto il segnale del video originale. Questo spiega l'apparente discontinuità riscontrabile nella zona centrale.

Il grafico in figura 5.8 descrive una situazione prevedibile: man mano che si sale di livello nella piramide, l'amplificazione diventa meno intensa ma più estesa. Purtroppo i contributi dei singoli strati non hanno sempre lo stesso segno e tendono ad annullarsi man mano che ci si allontana dalla zona centrale dell'oscillazione.

5.4 Allargamento

Il grafico in figura 5.8 aiuta a comprendere che il movimento viene allargato nello spazio e che ogni strato contribuisce in maniera diversa a questo allargamento. Adesso si mostra come un movimento appartenente ad un unico pixel venga distribuito dall'algoritmo. Il video è molto semplice ed è creato artificialmente. L'immagine è divisa in due metà. La prima è totalmente nera $([0, 0, 0])$, l'altra totalmente blu $([0, 0, 255])$. Il formato del video è sempre $(512; 512)$. La colonna centrale di pixel (256) è l'unica ad avere una colorazione differente e varia nel tempo. Il suo valore è dato da: $[0, 0, 127 + y(t)]$ con $y(t) = A \sin(\omega t)$. Il valore di ω è stato scelto arbitrariamente: $\omega = 3$. Prima dell'esecuzione del programma il video era quindi composto da righe di 512 pixel tutte uguali tra loro:

$$[0|0|\dots|0|127 + y(t)|255|255|\dots|255]$$

Il valore di $y(t)$ oscilla tra A e $-A$. La differenza tra due fotogrammi in cui l'oscillazione si trova agli estremi è quindi:

$$[0|0|\dots|0|2A|0|0|\dots|0]$$

Dopo aver eseguito il programma si vede che i valori di ogni riga di pixel sono stati uniformemente modificati. Adesso i punti in cui vi è una differenza sono 189. Questo significa che idealmente un'oscillazione che coinvolgeva un unico pixel è ora distribuita su 189 pixel. Questo numero dipende ovviamente dal numero di strati della piramide che si è deciso utilizzare (nel nostro caso 4). Fermandosi al terzo strato, l'oscillazione di un singolo pixel verrà distribuita su 93 pixel. Con due strati, questo numero scende a 45 e con un unico strato

arriva a 21. Il fatto che vi sia una differenza di intensità su 189 pixel non significa che questa differenza sia percepibile a occhio nudo. Con un fattore $A = 1$ e un fattore $\alpha = 10$ dei 189 pixel diversi da 0, solo 5 presentano una differenza maggiore di 1. Aumentare questo numero è possibile aumentando il prodotto $A\alpha$. Con un prodotto $A\alpha = 50$ i pixel che differiscono per più di un'unità sono 7, mentre con un prodotto $A\alpha = 100$ diventano 9.

5.5 Piccole oscillazioni con rumore.

La situazione analizzata in 5.3 rappresenta un caso ideale totalmente impossibile da riprodurre. La maggior parte dei pixel ha infatti una colorazione fissa. Come funziona il programma con casi più realistici? Cosa succede se si aggiunge un rumore su ogni pixel. Il fattore rappresentante il rumore è stato introdotto con l'utilizzo di una distribuzione normale centrata in zero.

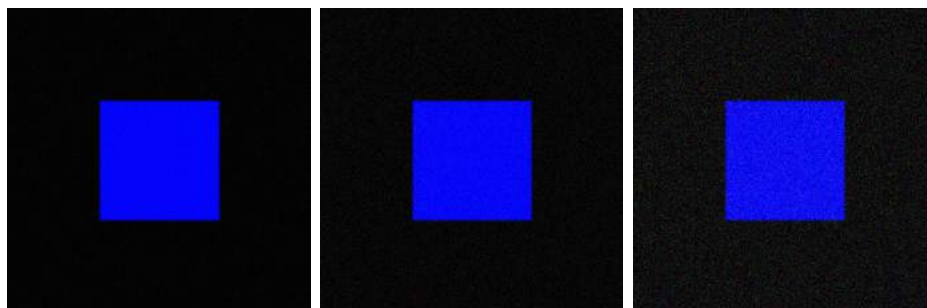


Figura 5.9: Tre fotogrammi uguali ma con rumori diversi. A sinistra il rumore è stato generato da una distribuzione normale con $\sigma = 10$, al centro con $\sigma = 20$ e a destra con $\sigma = 40$.

Il video originale in assenza di rumore presentava oscillazioni in prossimità dei bordi del quadrato di ampiezza uguale a 10. I grafici riportati nella colonna sinistra della figura 5.10 mostrano come il rumore renda complicato il riconoscimento di una periodicità nell'andamento del segnale. Ciononostante in tutti questi casi l'algoritmo riesce a ricostruire un segnale in cui

l'andamento sinusoidale è molto più evidente che in partenza. Il rumore non è dunque amplificato alla stessa maniera.

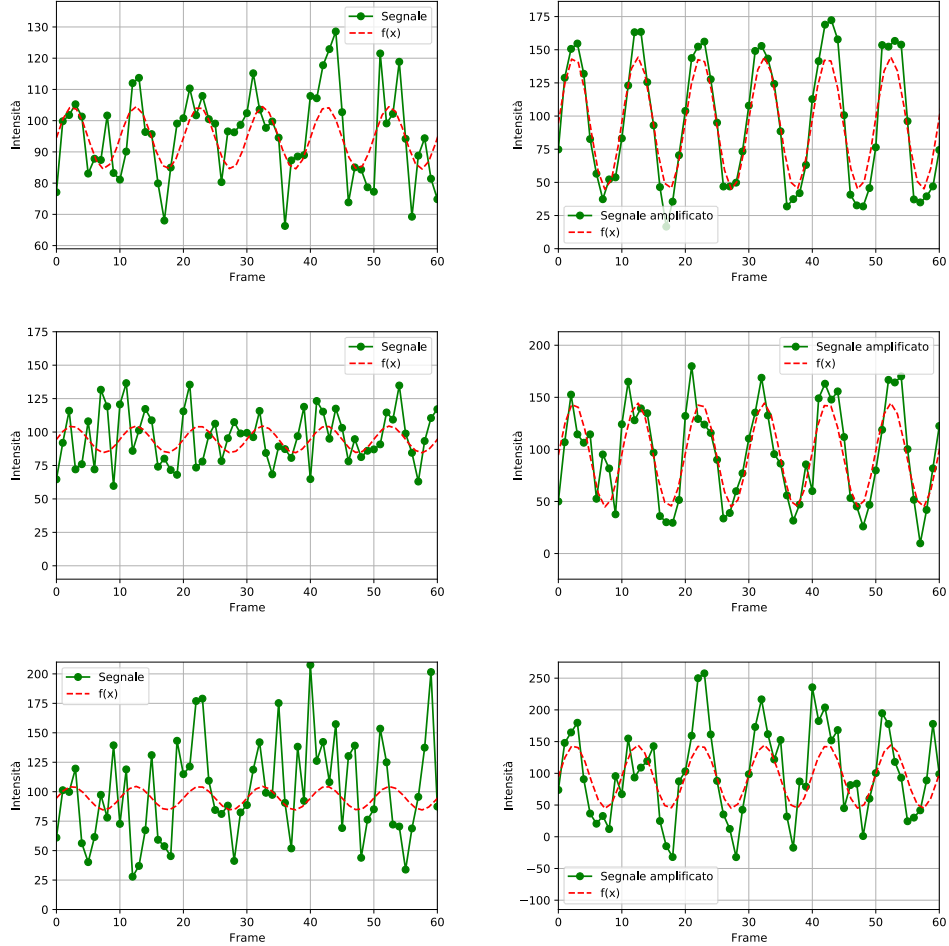


Figura 5.10: A sinistra in verde il segnale di un pixel originale generato dalla funzione $f(x)$ aggiungendo un rumore Gaussiano. A destra in verde è riportato l'andamento dello stesso pixel nel video finale, mentre in rosso è visibile il grafico di $\alpha f(x)$. Il valore di σ aumenta (10, 20 e 40) man mano che si scende.

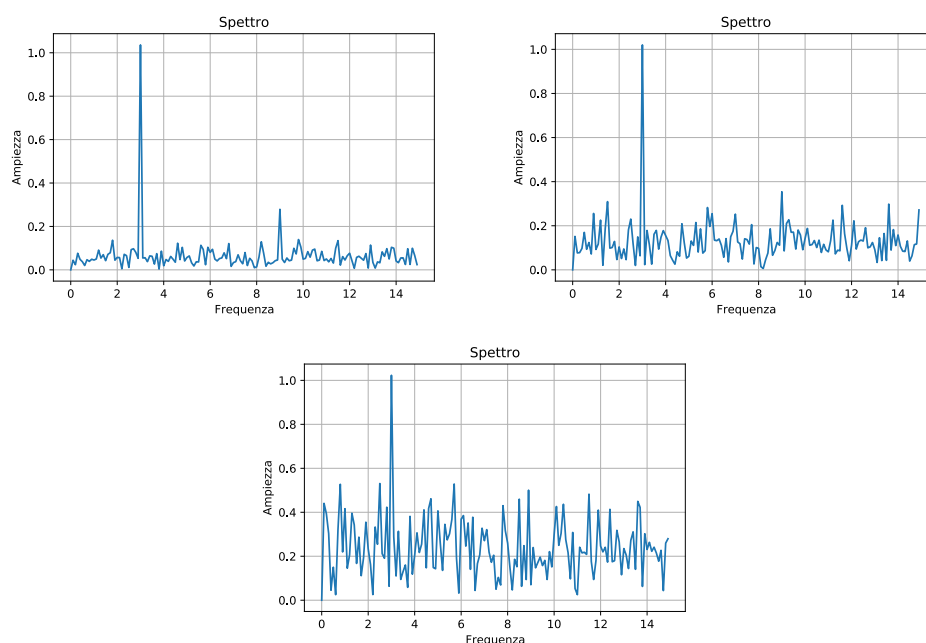


Figura 5.11: Spettro delle frequenze per i 3 pixel rappresentati in figura 5.10. Nonostante il rumore importante, è sempre possibile identificare la frequenza principale di oscillazione.

5.6 Amplificazione parziale o totale?

Come si è visto nel capitolo 4, l'algoritmo può essere utilizzato amplificando la totalità dei pixel, oppure solo una parte. Vediamo dunque nel dettaglio il metodo utilizzato per selezionare i pixel da amplificare e le differenze che si ottengono. I due grafici riportati mostrano la differenza tra lo spettro di un punto appartenente allo sfondo, non soggetto quindi a movimenti particolari, e un punto situato al confine di una regione oscillante.

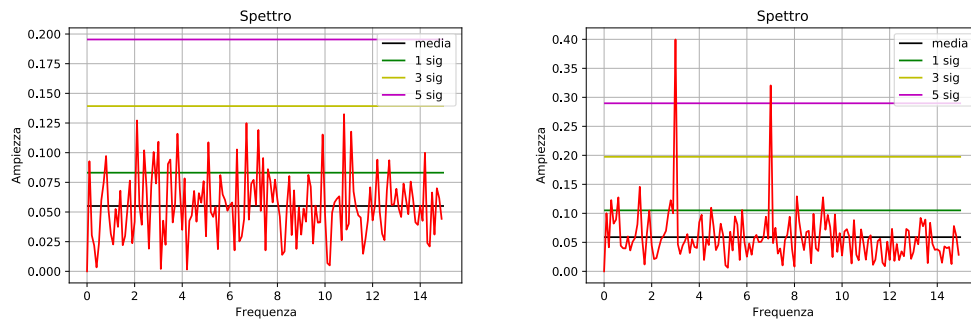


Figura 5.12: I punti sono stati selezionati dal primo livello della piramide di Laplace. Le linee continue orizzontali rappresentano il valore medio μ e i valori $\mu + \sigma$, $\mu + 3\sigma$ e $\mu + 5\sigma$

Lo spettro del punto appartenente allo sfondo non presenta nessun picco nel dominio delle frequenze. In figura 5.13 si vede la differenza tra due fotogrammi amplificati totalmente e parzialmente.

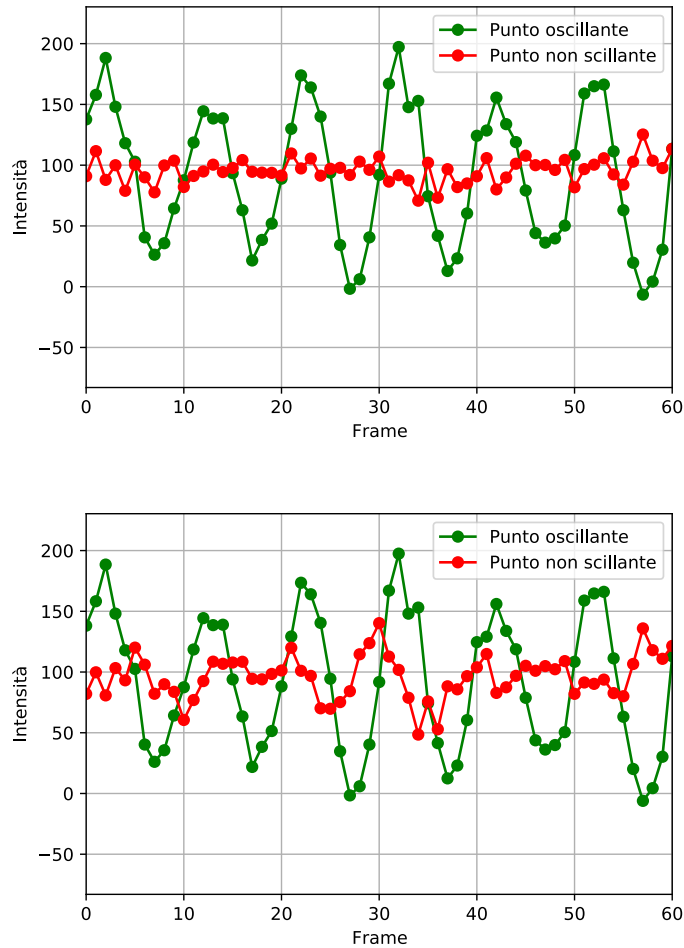


Figura 5.13: Nel primo grafico è possibile vedere in verde il segnale di un pixel amplificato, mentre in rosso vediamo un punto le cui oscillazioni non vengono modificate dall'algoritmo. Nel secondo grafico si vede invece l'effetto dell'amplificazione della totalità dei pixel. Anche il segnale del pixel non soggetto a oscillazioni viene alterato.

Ecco due fotogrammi diversi che evidenziano la differenza tra l'importanza del valore di soglia.



Figura 5.14: A sinistra l'algoritmo ha amplificato il segnale di tutti i pixel indiscriminatamente, a destra invece solo quelli che presentavano nello spettro un punto che differiva dalla media di almeno 7σ .

5.7 Oscillazioni a frequenze differenti

Il video studiato in questa sezione è stato generato in modo da avere regioni che presentano oscillazioni differenti tra loro. Si tratta sempre di un quadrato blu su sfondo nero ma in questo caso il bordo superiore e quello inferiore non vibrano con la stessa frequenza. Essi non hanno nessun tipo di correlazione e si muovono con frequenze totalmente differenti. L'unica cosa che condividono è l'ampiezza delle oscillazioni e il rumore. Questo tipo di materiale è finalizzato a comprendere quanto il programma sia preciso nell'amplificare esclusivamente le oscillazioni corrispondenti a determinate frequenze senza modificare il segnale dei pixel che rappresentano movimenti a frequenze differenti. Nel video sono dunque presenti due moti diversi con frequenze di 1.2Hz e 3.0Hz. Ecco una rappresentazione grafica dei risultati ottenuti amplificando le variazioni di intensità le cui frequenze caratteristiche sono comprese tra 2.5Hz e 3.5Hz con fattore $\alpha = 10$.

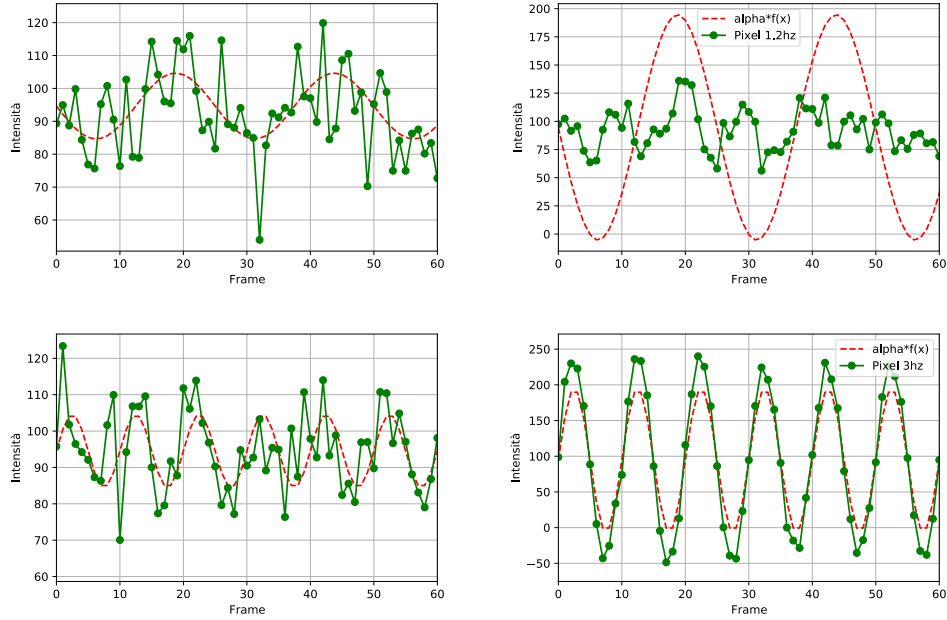


Figura 5.15: I primi due grafici mostrano l'andamento di un pixel oscillante a 1.2Hz prima e dopo l'esecuzione del codice. Il segnale non subisce grandi variazioni. In rosso è rappresentata la traiettoria che il segnale avrebbe seguito in assenza di rumore. Nel terzo e nel quarto grafico si vede invece come le oscillazioni a 3.0Hz siano effettivamente state amplificate. Risultati equivalenti si ottengono nel caso inverso.

Il programma è quindi in grado di selezionare e amplificare esclusivamente i movimenti dovuti a determinate frequenze.

Inoltre, come si è visto in (3.9) il programma dovrebbe consentire di amplificare distintamente i movimenti relativi a frequenze diverse (ν_1, ν_2). Dal punto di vista computazionale è sufficiente sostituire il filtro passa banda con un filtro a doppia banda. Questo permette di escludere tutte le oscillazioni indesiderate caratterizzate da una frequenza ν : $\nu_1 \leq \nu \leq \nu_2$ che sarebbero amplificate se si utilizzasse un unico filtro passa banda tra ν_1 e ν_2 .

5.8 Movimenti casuali

Supponiamo di voler utilizzare il programma su un sistema fisico caratterizzato da vibrazioni casuali. In questi casi è impossibile identificare una specifica frequenza attorno alla quale creare un filtro passa banda. L'amplificazione di tutto lo spettro delle frequenze può portare risultati soddisfacenti? Ovviamente l'assenza di filtraggio temporale implica che ogni perturbazione venga amplificata. Vediamo prima di tutto che effetti si ottengono in assenza di rumore. Il video raffigura un quadrato blu che questa volta si sposta in maniera casuale in tutte e quattro le direzioni. Il video prodotto dall'algoritmo sembra suggerire che l'algoritmo funzioni. Vediamo però nel dettaglio quanto i movimenti casuali finali rispecchino i movimenti originali.

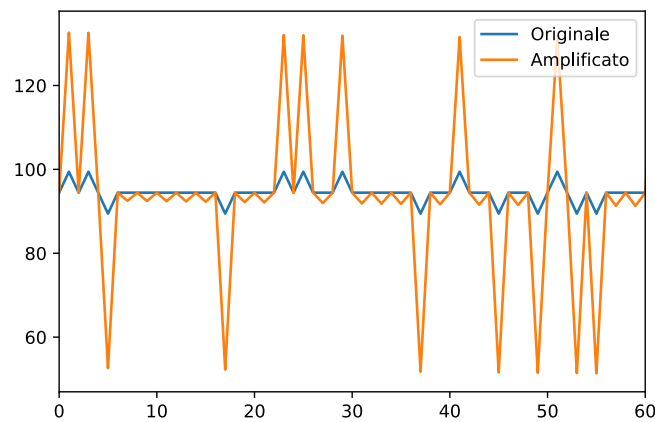


Figura 5.16: Un pixel al confine del quadrato soggetto a oscillazioni aleatorie.

In assenza di rumore l'andamento del video in output rispecchia perfettamente l'andamento del video originale. Come si comporta il programma con un rumore generato da distribuzione Gaussiana?

Le oscillazioni dovute ai movimenti casuali del quadrato sono caratterizzate da un aumento del colore blu di 5 unità. Vediamo i risultati ottenuti con 3 rumori diversi. Nel primo caso il rumore è costantemente inferiore all'oscillazione casuale. Nel secondo caso le oscillazioni e il rumore sono dello stesso ordine. Nel terzo caso il rumore è maggiore rispetto alle variazioni dovute al movimento.

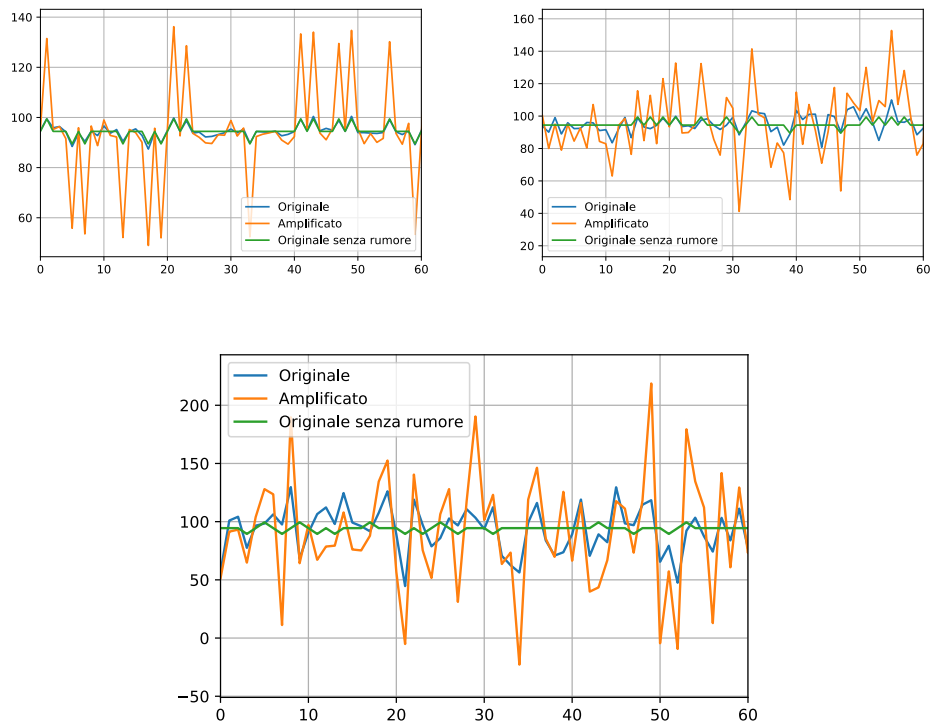


Figura 5.17: Tre situazioni con tre rumori diversi. Da sinistra a destra il rumore aumenta.

Nell'ultimo grafico in figura 5.17 il rumore è stato generato da una distribuzione $N(0, 20)$. Un rumore così grande altera significativamente il risultato generando un segnale in uscita che non rispecchia sempre la realtà come si vede al quarantaduesimo fotogramma. In quel caso il segnale originale au-

mentava l'intensità ma il rumore unito all'amplificazione creano un pixel in uscita dall'intensità ridotta. Ciononostante, le oscillazioni non sono puntuali e andando a mediare sulla totalità del bordo oscillante si ottiene un risultato finale che rispecchia l'originale. Il grafico in figura 5.18 mostra come il segnale amplificato medio sul bordo del quadrato rispecchi gli spostamenti generati prima dell'aggiunta del rumore. Nel video finale vi è un grande disturbo ma è possibile distinguere i movimenti casuali che nel video originale non erano riconoscibili a occhio nudo.

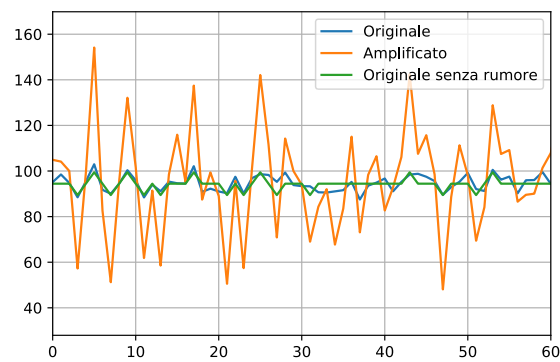


Figura 5.18: Andamento medio del segnale nella zona di confine del quadrato soggetto a movimenti casuali.

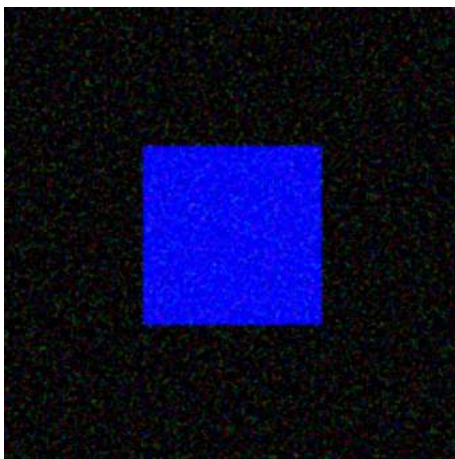


Figura 5.19: Un fotogramma del video rappresentante la situazione appena descritta.

Capitolo 6

Video

L'algoritmo è stato quindi testato con dei video reali. I primi due esempi fanno riferimento a due situazioni diverse.

- Un viso i cui lievi cambi di colore a livello cutaneo permettono di identificare il battito cardiaco del soggetto.
- Un bambino assopito apparentemente immobile. L'algoritmo permetterà di visualizzare i movimenti del petto dovuti alla respirazione.

Questi filmati sono gli stessi utilizzati da Wu et al. [17]. È quindi possibile confrontare i risultati ottenuti con quelli pubblicati in [17].

6.1 Volto e cambi colore.

Il video utilizzato è reperibile sul sito del gruppo di ricercatori che al MIT ha sviluppato l'Eulerian Video Magnification. Questo caso non è interessante per la rivelazione dei movimenti ma esclusivamente per i cambi di colore. Per questo motivo i risultati che vedremo sono tutti ottenuti con la piramide Gaussiana. Il video rappresenta un primo piano di un uomo di circa 50 anni. Il battito cardiaco è quindi verosimilmente compreso tra 45 e 120 b.p.m.



Figura 6.1: Un fotogramma del video analizzato.



Figura 6.2: In alto il susseguirsi dei fotogrammi dal video originale. In basso gli stessi fotogrammi a seguito dell'elaborazione del video. Le frequenze amplificate sono quelle comprese tra 0.7Hz e 2,0Hz. Il fattore $\alpha = 5.0$

Per verificare l'effettiva validità dell'algoritmo è utile confrontare il risultato con quelli ottenuti amplificando diversi range di frequenze. L'immagine successiva è stata creata prendendo la stessa colonna di pixel da ogni fotogramma.

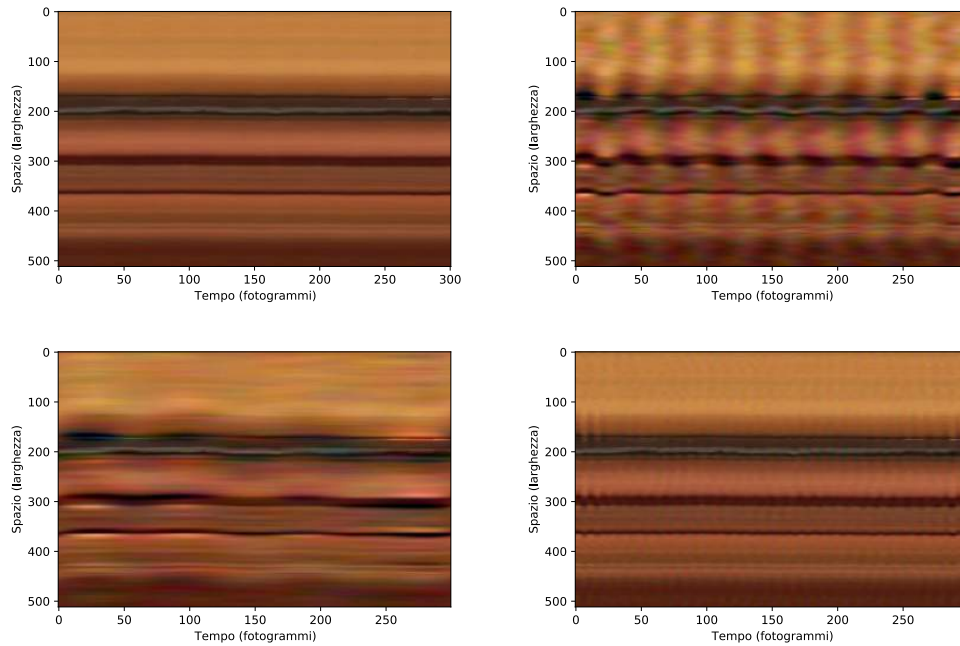


Figura 6.3: In alto a sinistra il video originale. In alto a destra l'amplificazione delle frequenze tra 0.7Hz e 2.0Hz. In basso il risultato ottenuto tagliando le frequenze tra 0.1Hz e 0.5Hz e tra 2,5Hz e 3,5Hz.

Se si guarda alla serie di colonne che compongono l'immagine in basso a destra in figura 6.3, si potrebbe pensare che anche a basse frequenze (0.1Hz - 0.5Hz) vi sia del moto caratteristico che possa essere soggetto a oscillazioni. In verità, è possibile vedere come questa evoluzione sia dovuta agli inevitabili movimenti del volto del protagonista del video. Per verificare questa affermazione riportiamo la media delle analisi spettrali effettuate in due diverse regioni.



Figura 6.4: Per ogni pixel contenuto nelle aree verdi è stata effettuata un'analisi spettrale. La regione più piccola in alto a destra non presenta punti della cute ma è comunque soggetta ai movimenti del soggetto. La regione più grande, invece, è composta quasi esclusivamente da pixel rappresentanti la pelle del soggetto.

È stata calcolata la media in modo da avere uno spettro quanto meno possibile influenzato dal rumore di fondo e da movimenti locali. In figura 6.5 è evidente il picco di frequenze in prossimità di 0.9Hz. Tale frequenza corrisponde a un ritmo cardiaco di 54 b.p.m..

Il picco a basse frequenze è quindi molto probabilmente dovuto ai movimenti involontari della telecamera e del soggetto. I risultati ottenuti amplificando il video sono assolutamente compatibili con quelli forniti dall'articolo originale sul Eulerian Video Magnification.

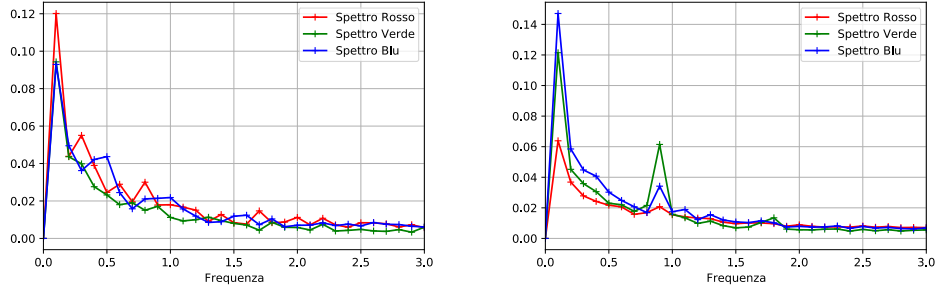


Figura 6.5: A sinistra lo spettro della zona esterna al volto. A destra invece lo spettro della regione centrale del viso.

6.2 Respirazione e movimenti

Un altro video che permette di confrontare i risultati del nostro algoritmo con quelli ottenuti in origine al MIT è il seguente: una scena raffigurante un bambino che dorme in cui tutto sembra apparentemente fermo. L'obiettivo è quindi quello di amplificare i movimenti del corpo dovuti all'espansione del torace durante l'alternarsi delle fasi respiratorie. Anche in questo caso il video è girato a 30 f.p.s. ed è stato ritagliato in modo da analizzarne una porzione quadrata ([512;512]).

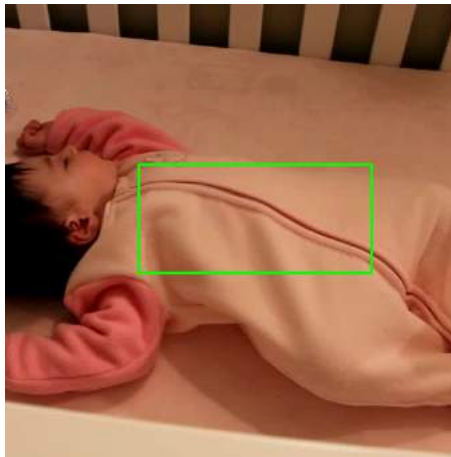


Figura 6.6: Un fotogramma del video. L'area evidenziata in verde è quella più sensibile ai movimenti respiratori.

Analizziamo quantitativamente i risultati. La frequenza respiratoria di un neonato è di 40 atti al minuto; nel sonno questa può scendere fino a un minimo di 20 atti al minuto. Sono dunque state amplificate le frequenze comprese tra 0.2Hz e 0.9Hz. Volendo evidenziare i movimenti è conveniente utilizzare la piramide di Laplace. Nel grafico in figura 6.7 sono riportati i valori delle frequenze mediati in una determinata regione di spazio.

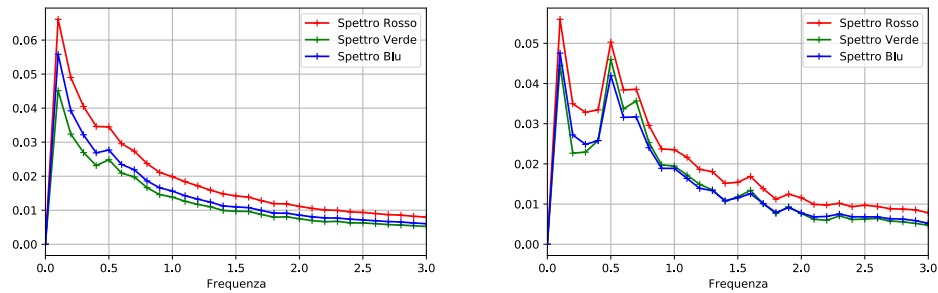


Figura 6.7: A sinistra lo spettro mediato su tutta l'immagine. A destra invece lo spettro medio della regione evidenziata in verde in 6.6.

Nella zona centrale vi è un picco delle frequenze in 0.5Hz. Questo valore corrisponde a una frequenza respiratoria di 30 atti al minuto, valore perfettamente compatibile con le informazioni mediche a disposizione. Di seguito riporto un'immagine ottenuta selezionando per ogni fotogramma la colonna centrale del riquadro evidenziato in verde in figura 6.6. La comparsa di un movimento precedentemente assente è evidente.

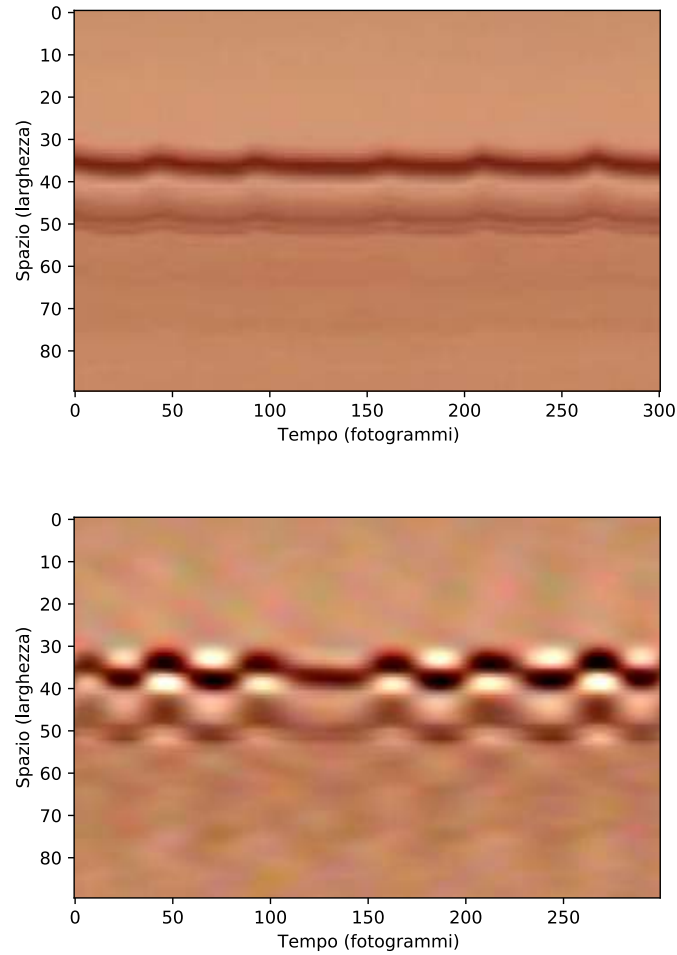


Figura 6.8: La prima immagine è stata generata partendo dal video originale, la seconda deriva dall'output generato con $\alpha = 15$

6.3 Pendolo

In questa parte si analizza il movimento di un pendolo. Questo sistema presenta alcuni vantaggi da sfruttare per analizzare al meglio i risultati dell'amplificazione in Python con l'EVM. Innanzitutto, la differenza tra il corpo in movimento e lo sfondo è ben definita. Inoltre, a differenza del torace del neonato analizzato precedentemente, il pendolo è un corpo rigido i cui punti si muovono all'unisono. Il video è stato realizzato con un telefono Xiaomi

Redmi Note 7 a 30 f.p.s. Il pendolo è stato perturbato in modo da innescare una lieve oscillazione nel piano perpendicolare alla direzione della telecamera. La presenza dell'attrito dell'aria tende a smorzare le oscillazioni del pendolo, che dopo pochi minuti sembra ritornato allo stato iniziale di equilibrio stabile. Le immagini ottenute rivelano le oscillazioni residue anche a diversi minuti dalla perturbazione.



Figura 6.9: Un fotogramma del pendolo in formato (512*512). Non sono riportati due fotogrammi rappresentanti le diverse fasi dell'oscillazione, dato che ad occhio nudo risultano praticamente identici.

Le riprese sono state effettuate a due istanti diversi. Saranno analizzate delle immagini girate circa 2 minuti dopo la perturbazione iniziale e altre girate a 5 minuti dalla perturbazione del pendolo. I video sono stati amplificati esattamente alla stessa maniera con un fattore $\alpha = 10$ e con un filtro passa banda compresa tra 0.7Hz e 1.3Hz.

6.3.1 2 minuti

Essendo impossibile visualizzare a occhio la differenza tra istanti diversi del video, è riportata un'immagine realizzata sottraendo due fotogrammi rappresentanti le fasi opposte dell'oscillazione. Per ogni pixel $[i; j]$ del fotogramma

n si indica l'intensità nel seguente modo:

$$I_{(n,i,j)}^k \quad k \in [0; 1; 2]$$

La lettera k indicizza i tre colori. Le immagini 6.10 sono create lasciando totalmente neri i pixel per cui la differenza (6.1) è minore di un valore di soglia.

$$\left| \sum_{k=0}^2 [I_{(n1,i,j)}^k - I_{(n2,i,j)}^k] \right| \quad (6.1)$$

I pixel in cui la differenza (6.1) supera la soglia stabilita vengono colorati di bianco. I fotogrammi $n1$ e $n2$ sono stati selezionati agli antipodi dell'oscillazione. Questa tecnica molto semplice rende visibile la differenza tra istanti diversi e permette di avere una prima stima sull'ampiezza in pixel dell'oscillazione del pendolo. Il valore di soglia è stato selezionato cercando di annullare la quasi totalità delle differenze dovute al rumore di fondo. La stessa procedura è stata effettuata con il filmato originale e con quello ottenuto a seguito dell'applicazione dell'EVM ($\alpha = 10$).

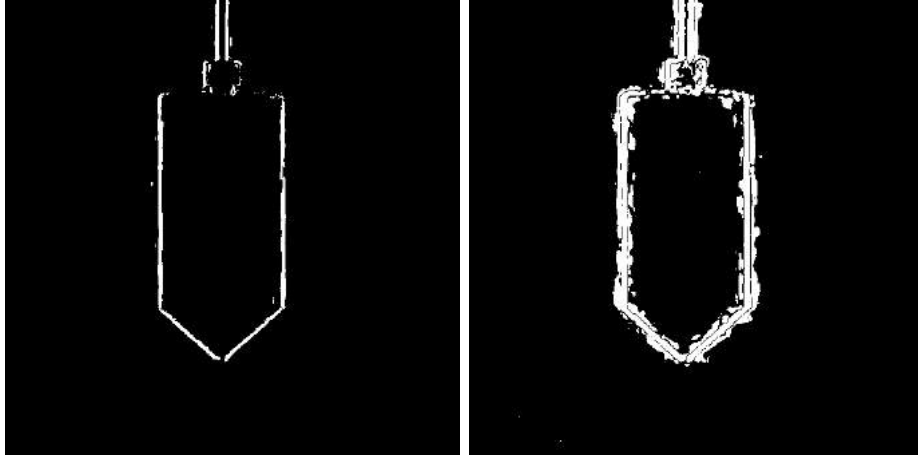


Figura 6.10: A sinistra il risultato ottenuto sottraendo due fotogrammi del video originale. A destra gli stessi fotogrammi dal video generato con l'EVM. Il valore di soglia è stato scelto a 25 in modo da eliminare la quasi ottalità delle differenze dovute al rumore. È evidente che l'amplificazione ha prodotto risultati in termini di differenze.

La cosa interessante che è possibile dedurre da questa immagine è che il rumore non è stato eccessivamente amplificato.

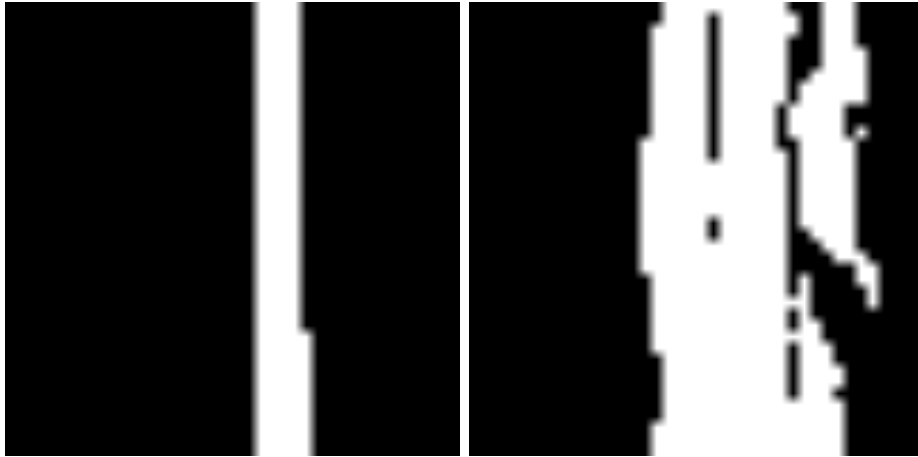


Figura 6.11: Due dettagli delle immagini 6.10 mostrano come prima dell'amplificazione l'oscillazione coinvolga circa 4 5 pixel. Dopo l'amplificazione la larghezza della banda bianca non è costante ma è uniformemente più spessa.

Per visualizzare le differenze tra fotogrammi opposti e come queste vengano amplificate, è riportato il grafico della differenza (6.1) lungo la direzione orizzontale. I tre colori sono riportati separatamente.

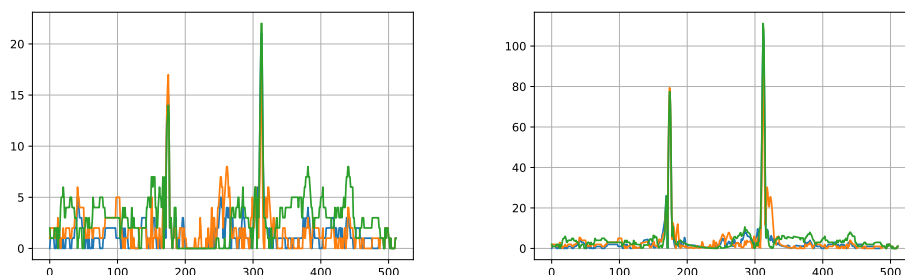


Figura 6.12: In questo caso la differenza è stata calcolata su un'unica linea orizzontale (256) di pixel. Sono evidenti i due picchi in corrispondenza delle estremità del pendolo sia prima che dopo l'ingrandimento. Inoltre, è possibile vedere come la differenza tra intensità a seguito dell'amplificazione sia di un ordine di grandezza superiore.

Lo stesso grafico è stato creato sommando il contributo di ogni singola linea. Il risultato mostra sia il contributo dell'oscillazione del pendolo sia i movimenti del filo e del sostegno.

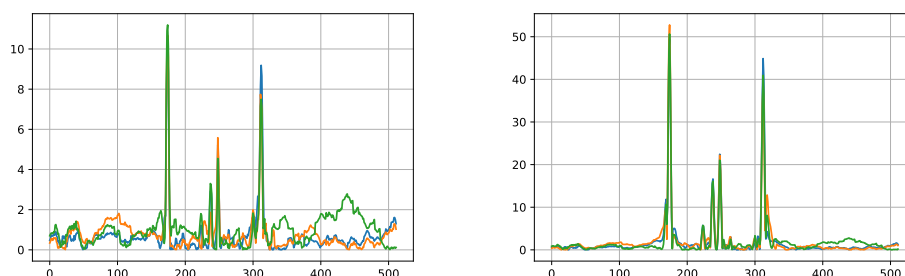


Figura 6.13: Mediare lungo la direzione verticale la differenza tra fasi opposte dell'oscillazione ha come risultato una notevole attenuazione del rumore rispetto alla situazione rappresentata in figura 6.12. In questi grafici compare un picco centrale dovuto alla presenza del filo e del sostegno.

Oscillazione dei pixel

Per individuare le oscillazioni è interessante anche studiare l'andamento dell'intensità dei pixel in funzione del tempo. Ricordiamo che l'algoritmo utilizza

esclusivamente questi dati per amplificare i filmati. I grafici riportati sono stati generati facendo le medie nelle regioni evidenziate in verde e in blu nella figura 6.14. L'andamento oscillatorio è evidente per i pixel che si trovano sul confine del corpo rigido (verde). I pixel immediatamente adiacenti (blu) non presentano nessuna oscillazione evidente.



Figura 6.14: I colori aiutano a identificare le zone analizzate.

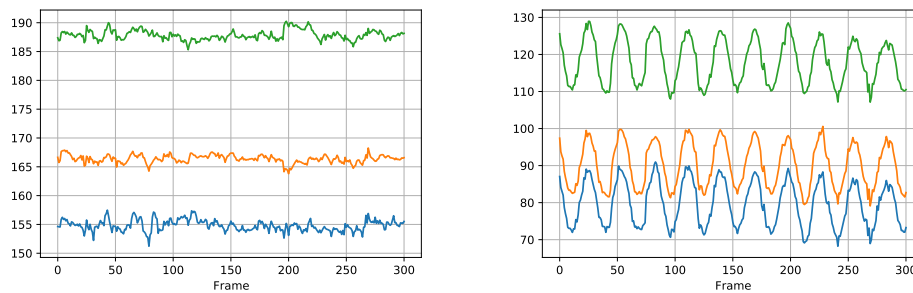


Figura 6.15: A sinistra l'andamento medio dei pixel nella zona blu. A destra quello della zona verde.

Questa differenza è molto visibile anche nello spettro delle frequenze che è stato calcolato per le stesse zone analizzate in figura 6.15. Questo permette di individuare con una certa precisione la frequenza delle oscillazioni.

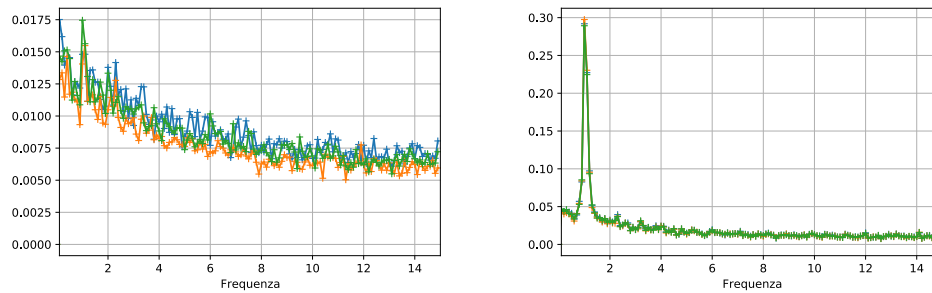


Figura 6.16: A sinistra lo spettro dei punti nella zona blu. A destra quello della zona verde. Il picco è in corrispondenza di $\omega = 0.9$ Hz.

La 6.15 conferma quanto detto nei paragrafi precedenti grazie alle immagini in 6.10: l'oscillazione è limitata a una piccola porzione di pixel. Non appena ci si sposta leggermente verso sinistra (regione blu) non vi è nessun moto oscillatorio.

Il programma non rende semplicemente più intensi i movimenti ma li espande su sezioni di pixel più ampie, grazie all'elaborazione spaziale iniziale. A dimostrazione di ciò, ecco l'andamento medio dell'intensità dei pixel nel video finale nelle 3 regioni esterne al pendolo evidenziate in azzurro, rosa e giallo in 6.14.

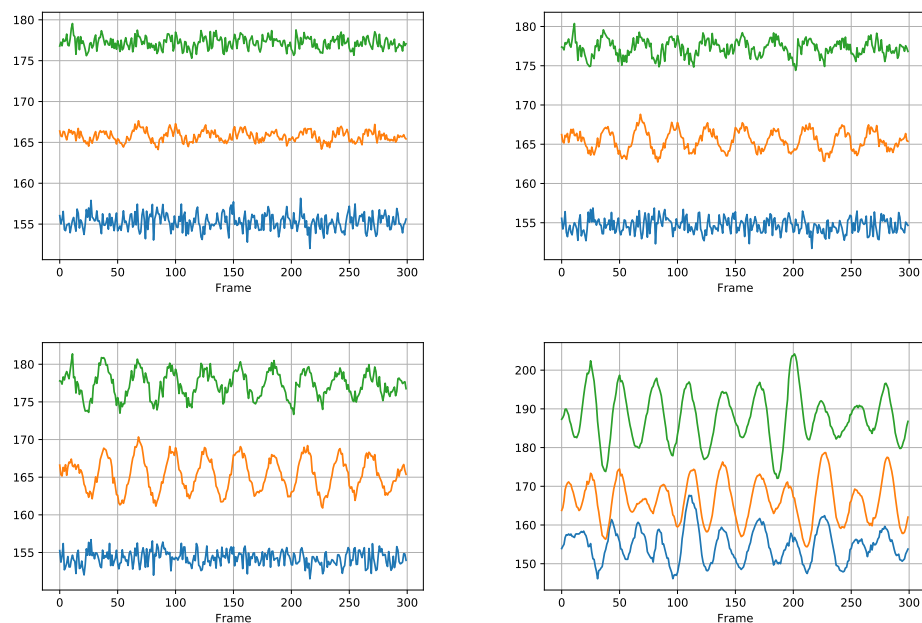


Figura 6.17: In ordine: andamento dei pixel a seguito dell'amplificazione nelle seguenti zone: azzurra, rosa, gialla e blu. Solo a 50 pixel di distanza dal pendolo l'oscillazione comincia a svanire.

Infine, è stata creata un'immagine raffigurante l'evoluzione dei movimenti del pendolo nel tempo nella direzione verticale. Questa è stata generata nello stesso modo della figura 6.8 e della 6.3.

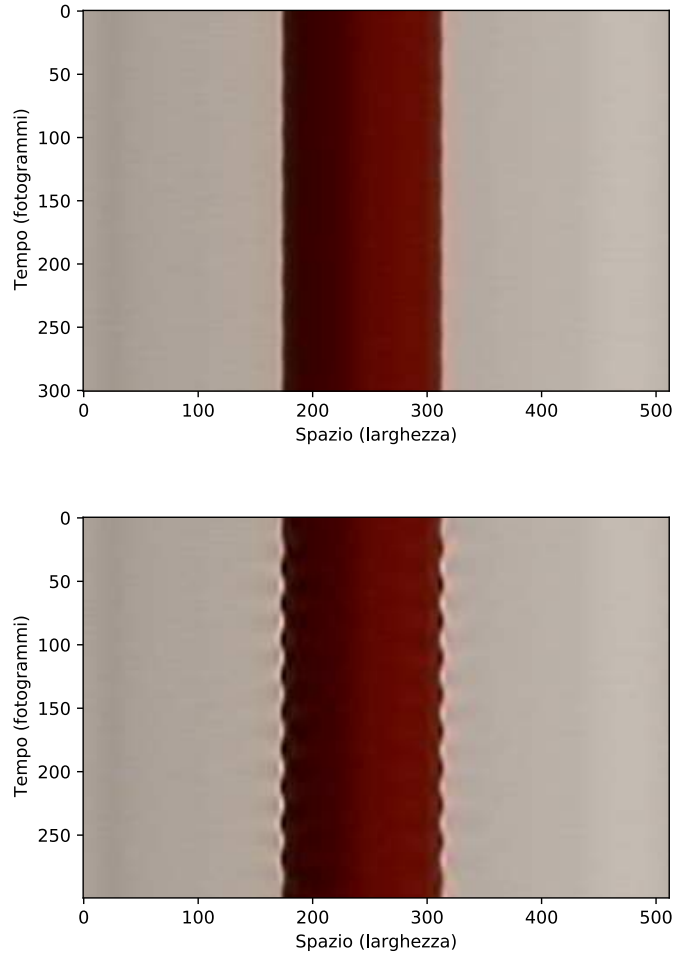


Figura 6.18: In alto una visualizzazione dell'evoluzione temporale prima dell'elaborazione. In basso la stessa immagine generata dal video ottenuto con l'EVM ($\alpha = 10$).

6.3.2 5 minuti

A 5 minuti di distanza dalla perturbazione del pendolo le oscillazione sembrano essere quasi del tutto svanite e il video originale sembra assolutamente statico. Di seguito sono riportati alcuni grafici equivalenti a quelli in 6.3.1 relativi alla fase distante 5 minuti dalla perturbazione del pendolo.

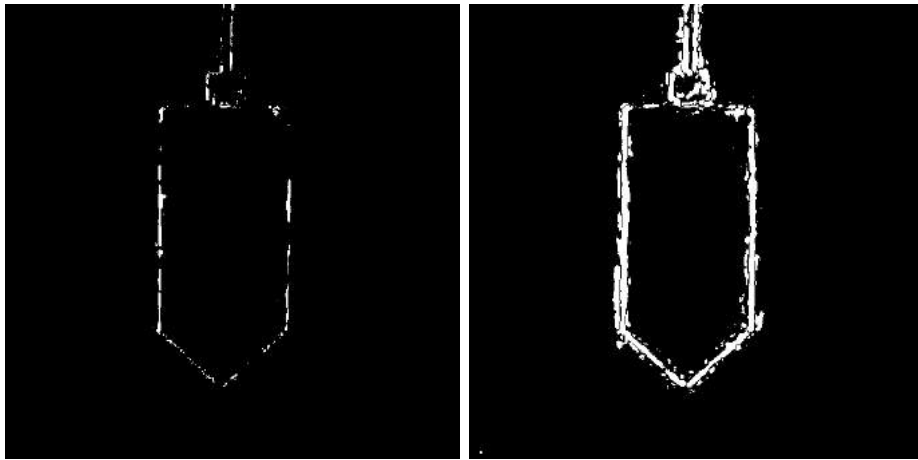


Figura 6.19: Queste due immagini sono ottenute esattamente nello stesso modo con cui sono generate quelle riportate in 6.10. La prima, in particolare, mostra come nel video originale si sia notevolmente attenuata l'ampiezza delle oscillazioni. La sagoma bianca del pendolo è molto meno definita.

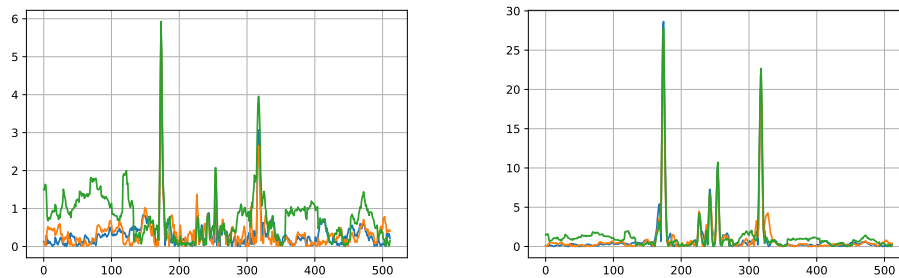


Figura 6.20: Questi grafici sono equivalenti a quelli riportati in 6.13 e mostrano la differenza lungo la direzione orizzontale media tra fotogrammi rappresentanti situazioni diverse. La differenza si è ridotta del 40%.

L'immagine più interessante è però quella che mostra l'evoluzione di una singola riga di pixel nel tempo. Mentre in figura 6.18 è possibile distinguere le lievi oscillazioni anche a occhio nudo, in figura 6.21 la zavorra sembra assolutamente statica nel tempo e solo grazie all'amplificazione è possibile visualizzare i movimenti.

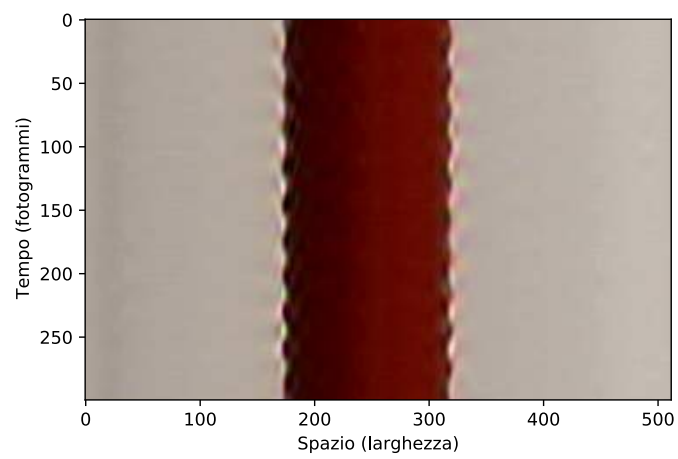
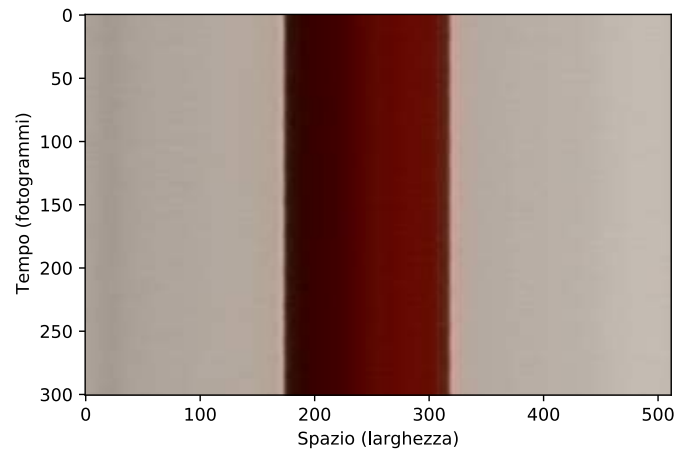


Figura 6.21: L'evoluzione temporale del pendolo a 5 minuti dalla perturbazione. In alto l'immagine creata dal video originale, in basso quella creata dall' output dell'EVM.

Capitolo 7

Conclusioni

Il programma che implementa in Python l'Eulerian Video Magnification funziona. Il programma è adatto all'amplificazione dei video e permette di visualizzare a occhio nudo le piccole oscillazioni presenti nelle sequenze di immagini. Il codice è stato caricato sulla piattaforma GitHub (<https://github.com/vincenzovitale95/EVM>) e è visualizzabile su Google Drive (<https://rb.gy/zlq3ra>).

Ho mostrato come l'opportuna scelta dei parametri per il filtraggio delle frequenze e per l'amplificazione produca diversi tipi di risultati. Questi parametri vanno scelti esclusivamente in funzione del video che si vuole analizzare. Ho inoltre dimostrato come la decomposizione dei fotogrammi con il metodo della piramide Gaussiana permetta di amplificare i colori, operazione per cui la piramide Laplaciana è risultata inadeguata. Ho quantificato l'amplificazione dell'ampiezza dei movimenti a seguito dell'amplificazione. Questa aumenta all'aumentare del numero dei livelli utilizzati nella scomposizione piramidale. Il programma è inoltre in grado di selezionare e di amplificare le oscillazioni anche in presenza di un rumore considerevole. Questa operazione è possibile, a condizione di conoscere le frequenze caratterizzanti i movimenti da rivelare. Il programma può amplificare anche le oscillazioni casuali; operazione che comporta un'inevitabile e considerevole amplificazione del rumore. Inoltre ho proposto una nuova tecnica per l'amplificazione del segnale che seleziona i pixel da amplificare in funzione del loro spettro delle frequenze. Questa tecnica di *amplificazione parziale* permette di utilizzare

un fattore di amplificazione α maggiore e attenuare l'amplificazione del rumore. L'algoritmo è quindi stato testato con gli stessi video utilizzati da [17]. Infine ho utilizzato il video di un pendolo apparentemente fermo per evidenziare le potenzialità dell'Eulerian Video Magnification anche nei casi in cui l'oscillazione coinvolge pochissimi pixel. I video con cui l'algoritmo è stato collaudato non sono stati girati con apparecchiature costose e/o sofisticate: quindi è possibile lavorare con quasi ogni tipo di video. L'algoritmo potrebbe inoltre essere unito a tecniche già esistenti per creare nuovi programmi. La tecnica di Viola e Jones [13] permette di selezionare porzioni di video in cui sono presenti volti umani. Potrebbe essere possibile applicare l'EVM in maniera indipendente a ogni porzione per rilevare il battito cardiaco dei soggetti inquadrati.

In conclusione, è possibile affermare che l'Eulerian Video Magnification può essere utilizzato in svariati ambiti con finalità diverse e che, nonostante le tecniche legate al *machine learning* siano al centro della maggior parte delle ricerche contemporanee sulla visione artificiale, i metodi numerici per l'analisi delle immagini e dei video continuano ad avere incredibili potenzialità.

Bibliografia

- [1] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden. 1984, Pyramid methods in image processing. *RCA Engineer*, 29 (6):33–41, 1984.
- [2] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986. ISSN 0162-8828. doi: 10.1109/TPAMI.1986.4767851. URL <https://doi.org/10.1109/TPAMI.1986.4767851>.
- [3] M. Elgharib, M. Hefeeda, F. Durand, and B. Freeman. Video magnification in presence of large motions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4119–4127, 2015.
- [4] B. K. Horn and B. G. Schunck. Determining optical flow. Technical report, USA, 1980. URL "<https://dspace.mit.edu/handle/1721.1/6337>".
- [5] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3):574–591, 1959. doi: <https://doi.org/10.1113/jphysiol.1959.sp006308>. URL <https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1959.sp006308>.
- [6] R. A. Kirsch, L. Cahn, C. Ray, and G. H. Urban. Experiments in processing pictorial information with a digital computer. In *Papers and Discussions Presented at the December 9-13, 1957, Eastern Joint Computer Conference: Computers with Deadlines to Meet*, IRE-ACM-AIEE

- '57 (Eastern), page 221–229, New York, NY, USA, 1957. Association for Computing Machinery. ISBN 9781450378628. doi: 10.1145/1457720.1457763. URL <https://doi.org/10.1145/1457720.1457763>.
- [7] J. S. Kulchandani and K. J. Dangarwala. Moving object detection: Review of recent research trends. In *2015 International Conference on Pervasive Computing (ICPC)*, pages 1–5, 2015. doi: 10.1109/PERVASIVE.2015.7087138.
 - [8] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision (ijcai). volume 81, 04 1981.
 - [9] S. Marie and J. Anudev. *Response Analysis of Eulerian Video Magnification*, pages 671–678. 01 2019. ISBN 978-3-030-00664-8. doi: 10.1007/978-3-030-00665-5_66.
 - [10] H. Nyquist. Certain topics in telegraph transmission theory. *Transactions of the American Institute of Electrical Engineers*, 47(2):617–644, 1928. doi: 10.1109/T-AIEE.1928.5055024.
 - [11] M. Rubinstein. *Analysis and Visualization of Temporal Variations in Video*. PhD thesis, Massachusetts Institute of Technology, Feb 2014.
 - [12] D. A. Ryan. Visible imaging of global mhd on mast. *IEEE Transactions on Plasma Science*, 42(10):2556–2557, 2014. doi: 10.1109/TPS.2014.2349931.
 - [13] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, 2001. doi: 10.1109/CVPR.2001.990517.
 - [14] N. Wadhwa, H.-Y. Wu, A. Davis, M. Rubinstein, E. Shih, G. J. Mysore, J. G. Chen, O. Buyukozturk, J. V. Guttag, W. T. Freeman, and F. Durand. Eulerian video magnification and analysis. *Commun. ACM*, 60(1):87–95, Dec. 2016. ISSN 0001-0782. doi: 10.1145/3015573. URL <https://doi.org/10.1145/3015573>.

- [15] Wikipedia contributors. Optical flow — Wikipedia, the free encyclopedia, 2020. URL https://en.wikipedia.org/w/index.php?title=Optical_flow&oldid=988229781. [Online; accessed 30-November-2020].
- [16] K.-W. Wong, K.-M. Lam, and W.-C. Siu. An efficient algorithm for human face detection and facial feature extraction under different conditions. *Pattern Recognition*, 34(10):1993 – 2004, 2001. ISSN 0031-3203. doi: [https://doi.org/10.1016/S0031-3203\(00\)00134-5](https://doi.org/10.1016/S0031-3203(00)00134-5). URL <http://www.sciencedirect.com/science/article/pii/S0031320300001345>.
- [17] H.-Y. Wu, M. Rubinstein, E. Shih, J. Guttag, F. Durand, and W. T. Freeman. Eulerian video magnification for revealing subtle changes in the world. *ACM Transactions on Graphics (Proc. SIGGRAPH 2012)*, 31(4), 2012.
- [18] Y. Zhang, S. L. Pinteá, and J. C. Van Gemert. Video acceleration magnification. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 502–510, 2017. doi: 10.1109/CVPR.2017.61.