

DDM: ANALISI DI SFOCatura IMMAGINI TRAMITE LINE DETECTION E ALLINEAMENTI TRA DATI

Marioemanuele Gianni, University of Florence

06/07/2019

Introduzione e obiettivi

Nel corso di questa relazione verrà esposto nel dettaglio il lavoro svolto, i concetti alla base e alcuni risultati rilevanti del progetto relativo al corso di Data and Document Mining del Professore Simone Marinai.

Il progetto ha lo scopo di effettuare un'analisi e uno sviluppo preliminare di un sistema per la rilevazione di frames sfocati in un flusso video e, eventualmente, segnalare in anticipo la problematica nel caso in cui si rilevi una progressiva perdita di fuoco.

Il caso di studio in questione non è particolarmente complesso; si prendono in esame, infatti, camere fisse in un contesto urbano dove vi sono molti piccoli elementi variabili quali passanti o veicoli ma anche elementi fissi di carattere architettonico e strutturale quali colonne, finestre o terrazzi di vari edifici.

Questo tipo di camere sono molto diffuse e hanno un impiego massiccio nell'ambito della videosorveglianza, del monitoraggio del traffico o dei controlli elettronici sui semafori e zone ad accesso limitato.

Tuttavia, esse presentano di rado problemi di sfocatura rilevabili solo a posteriori in una revisione video portando dunque ad una perdita di informazioni alla quale questo progetto tenta di sopperire.



Figure 1: Esempio di camera di sorveglianza in contesto urbano.

Idea generale

L'idea di fondo è stata dunque quella di concentrarsi sugli elementi fissi dell'immagine; andando infatti ad analizzare frame per frame le parti fisse potremo notare una perdita progressiva di informazioni per quanto riguarda i contorni nel caso di una progressiva sfocatura.

In tal senso, per la parte di analisi di immagini, è stato preso in considerazione il linguaggio Python e la libreria OpenCV.

OpenCV è una libreria software multiplattaforma e soprattutto, libera, nell'ambito della visione artificiale in tempo reale; verrà sfruttata principalmente per catturare e analizzare immagini tramite una LifeCam HD Microsoft.

Importate infatti le funzioni di cv2, sarà possibile configurare il fuoco e gli intervalli di cattura della camera USB collegata direttamente dal codice python permettendo dunque una cattura di frames automatica durante il giorno e la progressiva sfocatura delle immagini stesse.

A ogni cattura di frame segue una fase di analisi, che verrà illustrata successivamente nel dettaglio, attraverso alcuni dei principali algoritmi di openCV.

In fase di analisi si otterranno le linee principali dell'immagine e i punti che le determinano i quali saranno rielaborati e confrontati con frames precedenti.

Si procede dunque ad un allineamento e al calcolo di un indice che dipende dal numero di allineamenti e che ci offrirà una stima del grado di sfocatura dell'immagine catturata.

Una volta che verrà rilevata un decremento significativo di questo indice, si potrà segnalare che la camera sta iniziando a perdere il fuoco e, eventualmente, avviare una procedura di riavvio o di auto-focus.

Fase di cattura

Come già illustrato, la prima fase si occupa di catturare varie immagini tramite la camera USB. In tal senso ci viene subito in aiuto OpenCV che ci permette di accedere alla camera, settarne alcuni parametri e catturare frame di immagini.

OpenCV infatti, fornisce un'interfaccia molto semplice per accedere alle camere USB collegate e alle loro funzionalità; instanziando un oggetto di tipo VideoCapture (`cv2.VideoCapture()`), attraverso le funzioni di `set()` e `read()` è immediatamente possibile andare a cambiare il focus e leggere il frame corrente.

Il focus viene facilmente impostato passando un indice intero positivo; il valore 0 indica il focus ottimale mentre aumentandolo si ottiene un'immagine progressivamente sempre più sfocata (nel nostro caso, il valore oscillerà tra 0 e 25).

Inizialmente, attraverso un portatile e la camera in posizione fissata su una finestra sono stati fatti alcuni test per poi invece portare il codice python su Raspberry PI.

Con il raspberry, data la maggiore versatilità, con un while loop, sono state invece catturate

svariate immagini a intervalli regolari che costituiranno una sorta di "dataset" di confronto per le successive.

Le immagini, catturate ad orari differenti, presenteranno dunque dettagli e soprattutto luminosità diverse che influiranno sulla rilevazione dei contorni e servirà dunque, quando verranno catturate nuove immagini, un confronto diretto con immagini del dataset di orari circa corrispondenti.

Fase di elaborazione

Ogni volta che viene catturato e selezionato un frame per l'analisi, viene inizialmente convertito a scala di grigi.

Per scala di grigi, in analisi delle immagini e nella fotografia digitale, si intende un'immagine dove il valore di ogni singolo pixel è un singolo campione che rappresenta solo una quantità di luce/luminosità portando con se quindi solo informazioni di intensità.

Le immagini a scala di grigi ("grayscale"), come suggerisce il nome, sono costituite esclusivamente da sfumature del grigio e questo step di conversione a grayscale serve soprattutto per il passaggio successivo in quanto riduce la complessità dell' immagine e mantiene le caratteristiche importanti (la luminosità) per il successivo algoritmo di edge detection.

La fase successiva è costituita dal rilevamento degli edges dell'immagine grayscale tramite la funzione OpenCV "cv2.Canny()" che implementa l'algoritmo di Canny.

L'algoritmo di Canny è un'operatore per il riconoscimento dei contorni sviluppato nel 1986 da John F.Canny che utilizza un metodo di calcolo multi-stadio per individuare contorni di molti dei tipi normalmente presenti nelle immagini reali.

L'algoritmo è costituito da varie fasi con scopi distinti; una prima fase si occupa della riduzione del rumore attraverso la convoluzione con un filtro gaussiano: il risultato è un'immagine con una leggera "sfocatura" gaussiana, in cui nessun singolo pixel è affetto da disturbi di livello significativo.

L'equazione di un filtro gaussiano con kernel $(2k + 1)(2k + 1)$ è data da:

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}\right); 1 \leq i, j \leq (2k + 1)$$

La seconda fase si occupa della ricerca del gradiente della luminosità dell'immagine attraverso quattro filtri differenti; viene assunta come direzione del gradiente quella relativa al filtro che dà come risultato il valore maggiore.

Costruita dunque la mappa dei gradienti, si procede a sopprimere i non massimi, e quindi i falsi contorni, attraverso la derivata del gradiente per poi arrivare all'ultimo step che estrae i

contorni attraverso un procedimento di sogliatura con isteresi (riferito al fatto che si usano due soglie, una alta e una bassa) .

E dunque si ha che se il valore del gradiente risulta:

1. inferiore alla soglia bassa, il punto è scartato;
2. superiore alla soglia alta, il punto è accettato come parte di un contorno;
3. compreso fra le due soglie, il punto è accettato solamente se contiguo ad un punto già precedentemente accettato.

Come possiamo osservare in Figura 2, alla funzione cv2.Canny() viene passato in ingresso come primo parametro l'immagine grayscale e successivamente i due bounds per la sogliatura.

Questi due bounds vengono determinati calcolando prima la mediana delle intensità dei singoli pixel e poi attraverso un parametro $+\sigma$ ($\sigma = 0.33$, statisticamente buono per la maggior parte dei dataset su cui è stato applicato Canny).

Viene dunque generata infine l'immagine degli edges che sarà il punto di partenza per la successiva elaborazione che ci porterà ad identificare le linee principali dell'immagine.

Si passerà infatti per la funzione "cv2.HoughLinesP()" che implementa l'algoritmo basato sulla trasformata di Hough probabilistica.

Quest' immagine viene poi successivamente passata alla funzione "cv2.HoughLinesP()" che implementa la trasformata di Hough probabilistica per rilevare le varie linee presenti nell'immagine.

Come sappiamo, una linea può essere espressa nel sistema di coordinate cartesiane e nel sistema di coordinate polari; per la trasformata di Hough verrà fatto riferimento alle coordinate polari per esprimere una linea.

```
#convert to grayscale
string = "frame{}_{}gray.jpg".format(counter, now.strftime("%Y-%m-%d %H:%M:%S"))
img = frame
img = cv2.resize(img, (1107, 623))
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imwrite(os.path.join(dirname1, string), gray)
cv2.imshow('Original image', img)
cv2.imshow('Gray image', gray)
cv2.waitKey(0)
cv2.destroyAllWindows()

#Canny edge detection
v = np.median(gray)
lower = int(max(0, (1.0 - 0.33) * v))
upper = int(min(255, (1.0 + 0.33) * v))
string = "frame{}_{}edge.jpg".format(counter, now.strftime("%Y-%m-%d %H:%M:%S"))
edges = cv2.Canny(gray, lower, upper, None, 3)
cv2.imwrite(os.path.join(dirname1, string), edges)
cv2.imshow('Edges', edges)
cv2.waitKey(0)
```

Figure 2: Breve porzione di codice che mostra la semplicità di utilizzo di openCV per la conversione a grayscale e l'algoritmo di Canny.

Dunque, l'equazione di una retta si può esprimere come:

$$y = \left(-\frac{\cos \theta}{\sin \theta}\right)x + \left(\frac{r}{\sin \theta}\right)$$

E, arrangiando i termini e generalizzando per ogni punto (x_0, y_0) , possiamo definire la famiglia delle linee che passano attraverso quel punto come:

$$r_\theta = x_0 \cdot \cos \theta + y \cdot \sin \theta$$

individuata da i due parametri (r_θ, θ) .

Se per ogni punto rappresentiamo sul piano la famiglia delle linee che lo attraversano, otteniamo una sinusoida.

La parte interessante è che, eseguendo questa operazione per tutti i punti dell'immagine, se due curve di punti differenti si intersecano, significa che entrambi i punti appartengono alla solita linea.

Questo significa che, in generale, una linea può essere rilevata trovando il numero di intersezioni tra curve e possiamo definire una soglia del numero minimo di intersezioni necessarie per definire una linea.

Questo è proprio ciò che si occupa di fare la trasformata di Hough tenendo traccia delle intersezioni tra le curve di ogni punto dell'immagine.

La variante di Hough probabilistica è un'ottimizzazione di questa trasformata in quanto di base risulta pesante computazionalmente.

Per alleggerire il processo, si prendono dunque in considerazione solo un sottoinsieme randomico di punti su cui fare la detection delle linee.

A seguito di questa operazione, viene generata un'immagine identica al frame catturato ma che evidenzia le linee rilevate (Vedesi figura 4).

Le linee saranno dunque identificate da due punti e dalle loro coordinate da cui sarà calcolata l'equazione delle rette tramite la formula solita:

$$\frac{x-x_1}{x_2-x_1} = \frac{y-y_1}{y_2-y_1}$$

Per poi essere ricondotta nella forma implicita " $ax + by + c = 0$ " da cui verranno estratti i parametri a, b e c e salvati su file.

L'intero processo di elaborazione è descritto visivamente in Figura 3.

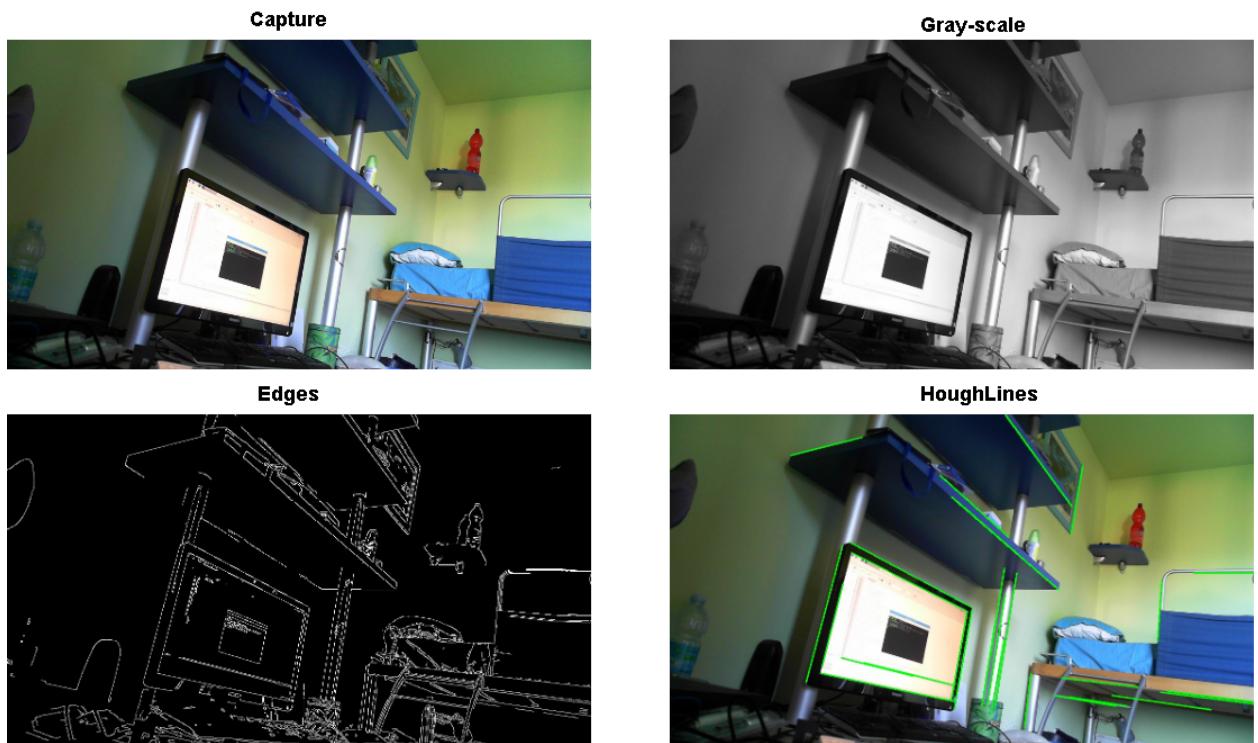


Figure 3: Fase di elaborazione in un contesto ambientale interno. Dall'alto verso il basso e da sinistra verso destra possiamo osservare i vari steps descritti e le loro immagini output.



Figure 4: Esempio di contesto ambientale esterno con l'immagine originale a sinistra e il risultato finale della fase di elaborazione a destra.

Allineamento e detection

A questo punto, quando viene catturata una nuova immagine, si cerca nel "dataset" una cattura avvenuta in orario simile e si va a prendere in esame il file contenente i dati delle rette rilevate.

Si cerca dunque un modo per confrontare le rette rilevate nell'immagine corrente con quelle rilevate nell'immagine selezionata per poi andare a conteggiare quelle uguali o comunque molto simili.

A tal proposito, ci viene in aiuto l' "Hungarian matching algorithm" che è fondamentalmente un metodo di ottimizzazione combinatoria che risolve in tempo polinomiale il problema dell'assegnamento.

I problema dell'assegnamento richiede di associare coppie di elementi presi da due insiemi differenti. Tenendo conto che a ogni coppia è associato un costo, si vuole minimizzare il costo totale.

Nel nostro caso, abbiamo un insieme di rette dell'immagine corrente e un'insieme di rette dell'immagine presa dal dataset; abbiamo inoltre che per ogni associazione di retta corrisponde un "costo" dato dalla loro "distanza".

Si vuole dunque associare ad ogni retta dell'immagine corrente una retta dell'immagine dataset minimizzando il costo.

E' stata scelta come misura della "distanza" tra due rette la variazione percentuale in quanto i parametri a,b e c della retta (soprattutto il parametro c) variano su range diversi ed, ad esempio, una variazione di 1/2 del parametro a corrisponde ad una variazione di qualche centinaio del parametro c.

Nell' hungarian matching algorithm, il problema è codificato tramite un grafo bipartito, in cui i vertici sono gli elementi da associare e gli archi rappresentano le possibili scelte di coppie.

A ogni arco è associato un costo.

L'algoritmo inizia da una qualsiasi soluzione, anche parziale, e tenta di migliorarla a ogni iterazione cercando un percorso che aumenti il numero di elementi presenti nella soluzione.

Vi è anche un'interpretazione matriciale dell'algoritmo che, data una matrice quadrata di ordine n rappresentante la matrice dei costi del problema, si definisce in vari steps:

1. Per ogni riga, individuare il minimo e sottrarlo a tutti gli elementi della riga;
2. Per ogni colonna, individuare il minimo e sottrarlo a tutti gli elementi della colonna;
3. Coprire con il minor numero di linee tutti gli zeri che si sono formati con le precedenti sottrazioni;
4. Se il numero minimo di linee necessarie è pari a n è possibile determinare un assegnamento ottimo altrimenti procedere con lo step 5;
5. Individuare il minimo tra gli elementi non coperti da linee, sottrarlo agli elementi non coperti e sommarlo agli elementi che sono incrocio di due linee. Tornare allo step 3.

Vedesi figura 5 per maggiori dettagli.

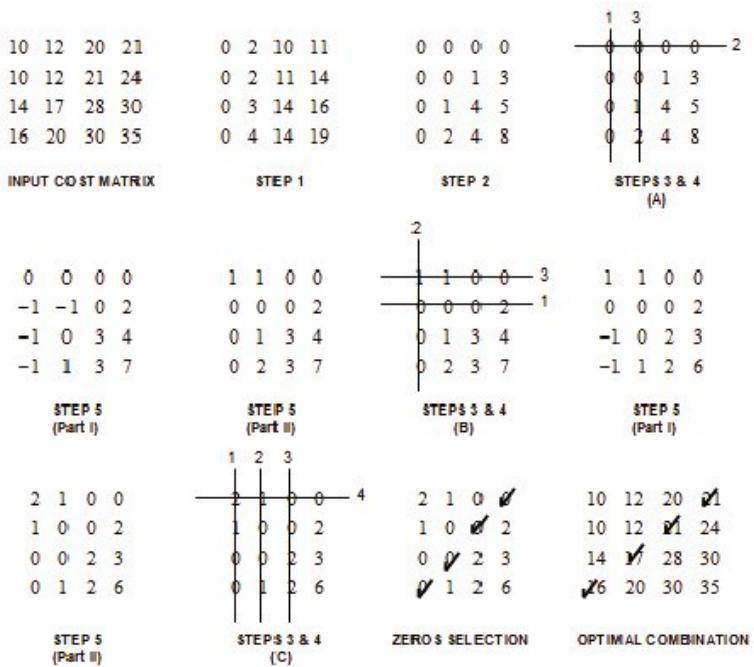


Figure 5: Esempio di procedura step-by-step dell'hungarian matching algorithm.

Source: Researchgate, Gerardo Agni Medina-AcostaJosé A. Delgado-PenínJosé A. Delgado-Penín

A questo punto, applicando quest'algoritmo, abbiamo assegnato ad ogni retta del frame corrente una retta del frame selezionato (Figura 7) ma, in aggiunta, viene impostata una soglia sulla variazione percentuale per arrivare ad ottenere e conteggiare quelle effettivamente corrispondenti o comunque molto simili (Vedesi figura 6 e figura 8).

Ottenuto dunque il numero degli allineamenti (intersezioni tra i due insiemi) e tenendo presente il numero di linee rilevate sul frame corrente e su quello selezionato (con cui si può calcolare l'unione), possiamo andare a calcolarci l'indice di Jaccard:

$$Jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Questo indice, calcolato su frame corrispondenti in termini di orario, dipendente dal numero degli allineamenti e dal numero di linee rilevate, è decrescente al progressivo sfocarsi dell'immagine; infatti si andranno a diminuire gli allineamenti e, essendo "pesato" sul numero di linee rilevate, sarà poco dipendente dai momenti della giornata con diverso tipo di illuminazione.

Per concludere, una volta che viene effettivamente rilevato un andamento medio decrescente dell' indice di Jaccard, si può determinare una soglia sotto la quale segnalare la sfocatura o il progressivo sfocarsi dell'immagine.

```

161 -198 61585
ALIGNED TO: ['163,', '-201,', '62113']

140 -109 93131
ALIGNED TO: ['140,', '-109,', '93240']

99 -118 40799
ALIGNED TO: ['99,', '-118,', '40799']

90 -107 37022
ALIGNED TO: ['93,', '-110,', '38366']

48 -209 -75095
ALIGNED TO: ['47,', '-207,', '-74470']

```

Figure 6: Esempio di un risultato di allineamento tramite parametri delle linee. Possiamo vedere come la terza sia stata trovata perfettamente identica mentre le altre leggermente scostanti ma comunque molto simili.

```

([11, 269, 948, 462, 589, 424],
 [47, 265, 907, 451, 567, 413],
 [512, 94, 1026, 500, 568, 417],
 [568, 477, 7, 108, 100, 101],
 [495, 567, 197, 2, 93, 53],
 [737, 620, 156, 111, 5, 79],
 [625, 643, 251, 72, 93, 6],
 [906, 719, 226, 173, 41, 105])

[(0, 0), (2, 1), (3, 2), (4, 3), (5, 4), (6, 5)]

```

Figure 7: Esempio di una matrice dei costi. In fondo gli indici delle coppie di linee associate in base al costo minimo secondo l'Hungarian Matching.

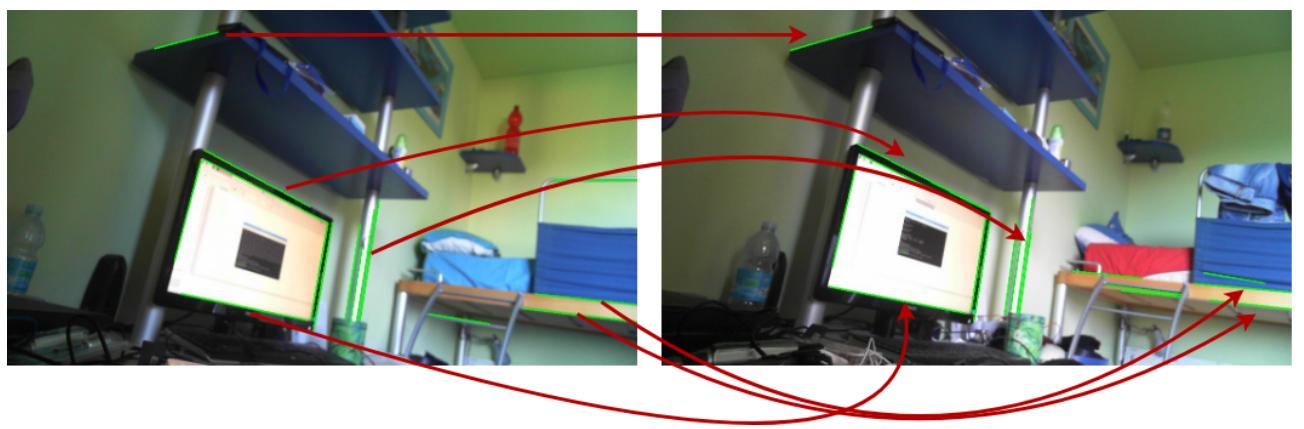


Figure 8: Esempio visivo di un risultato di allineamento tramite parametri delle linee.

Risultati sperimentalni

Il codice python è stato testato infine svariate volte in varie situazioni e giorni; come già accennato, sono stati presi dei frames precedenti con cui confrontare le immagini catturate e sono stati fatti rilevamenti sia la mattina alle 7.00 sia alle 12.00 che il pomeriggio alle 17.00 e la sera alle 20.30.

E' stato dunque inizialmente impostato un indice di focus a 0 per poi aumentare gradualmente in vari step fino a raggiungere una cattura quasi completamente sfocata. (Vedesi esempio in Figura 9)

Per ogni step, tenuto conto del fatto che l' HoughLines probabilistico non riporta valori sempre uguali ma oscillanti (seppur in maniera contenuta) , sono stati rilevati 10 frames e calcolata la media del loro indice di Jaccard.

Un' altro parametro importante che è stato testato è la soglia con cui vengono fatti gli allineamenti; come esposto in precedenza, due rette si considerano allineate se la variazione percentuale dei loro parametri è al di sotto di una data soglia.

Ovviamente, impostando una soglia alta, si vanno ad incrementare il numero degli allineamenti e viceversa; la soglia ottimale è quella tale per cui si ha il minor numero di allineamenti in caso di immagine totalmente sfocata e un' indice di Jaccard nettamente decrescente nel caso di progressivo sfocamento.

Sono state fatte varie prove; di seguito, a titolo di esempio, vi sono due esempi grafici dell'andamento medio dell'indice di Jaccard con una soglia del 10% e una soglia del 5% (Figura 10).

Tenendo presente che un valore del parametro focus pari a 20 corrisponde ad un'immagine quasi totalmente sfocata e un valore 25 ad una totalmente sfocata notiamo bene come l'andamento dell'indice nella soglia a 10% non sia effettivamente tanto decrescente quanto nella soglia a 5% e, addirittura, vengono rilevati allineamenti anche in caso di immagine totalmente sfocata.

D'altra parte, il grafico della soglia a 5%, non solo non ci mostra allineamenti al di sopra del valore 20 di focus ma ha un andamento molto decrescente che facilita anche la detection di una progressiva sfocatura.

Infatti, tenendo presente che con un valore di focus pari a 15 l'immagine si presenta già leggermente sfocata, possiamo, nel secondo grafico, impostare un'ulteriore soglia per l'indice di Jaccard a 0.1 ed è poi facilmente riconoscibile se stiamo andando verso la sfocatura o meno in quanto i vari steps hanno risultati dell'indice più distanti.

Successivamente sono state fatte prove, con l'aiuto del dottorando Francesco Lombardi, anche in un contesto più realistico con camera puntata da una finestra del laboratorio A.I. di S.Marta e attiva 24h/24 (vedesi Figure 11 e 12).

L'andamento dei risultati è molto simile anche se tendenzialmente vengono rilevate molte meno linee all'aumentare della sfocatura in quanto gli elementi sono a distanza maggiore.

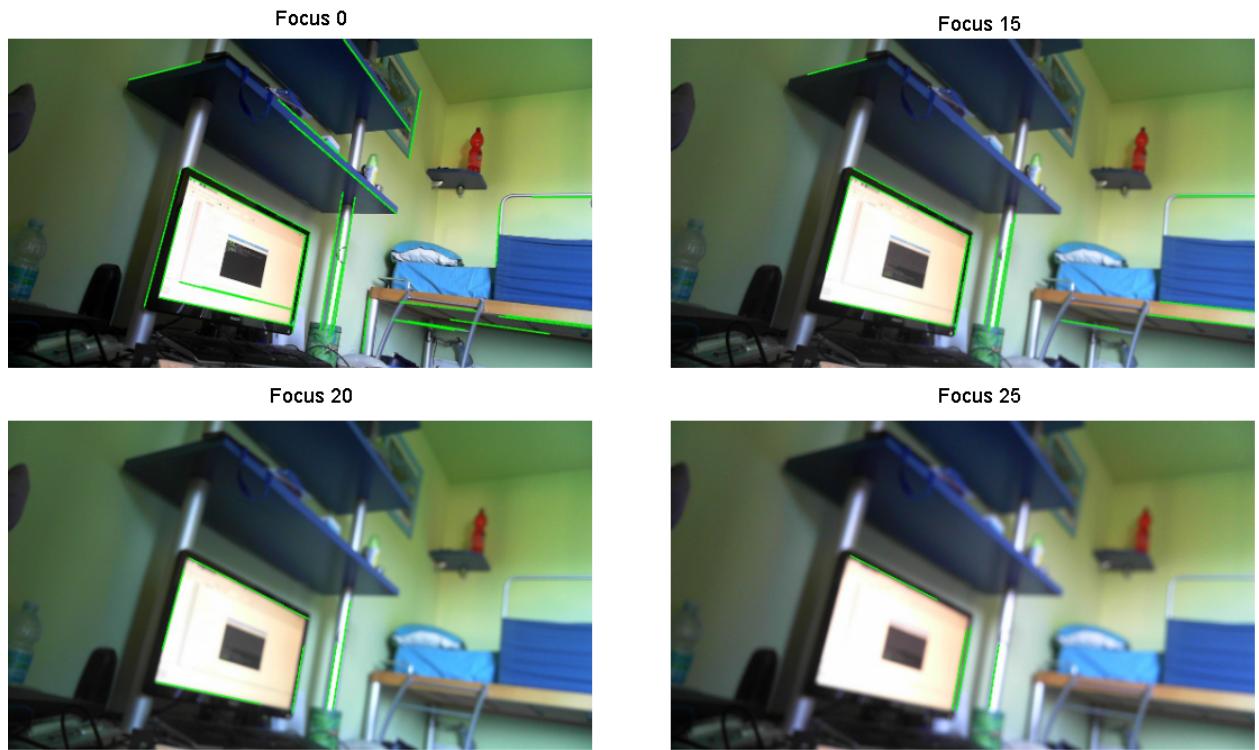


Figure 9: Esempio di sfocatura progressiva tramite il parametro del focus. Si osserva inoltre la perdita di informazioni e il conseguente numero minore di linee rilevate.

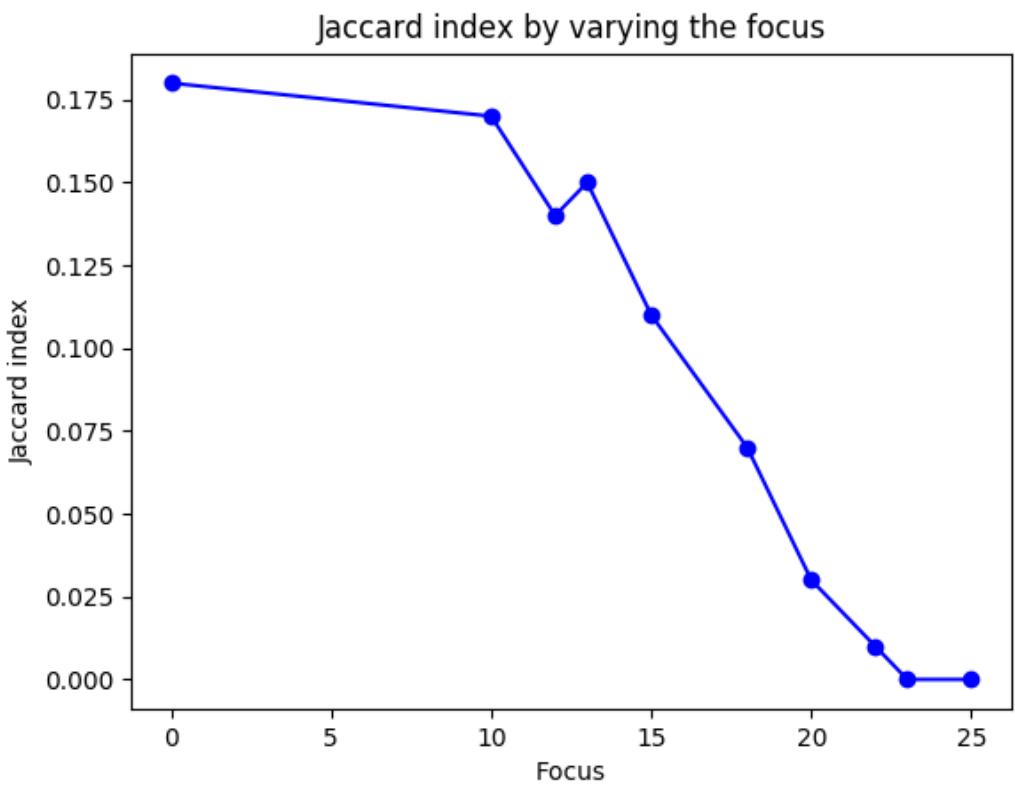
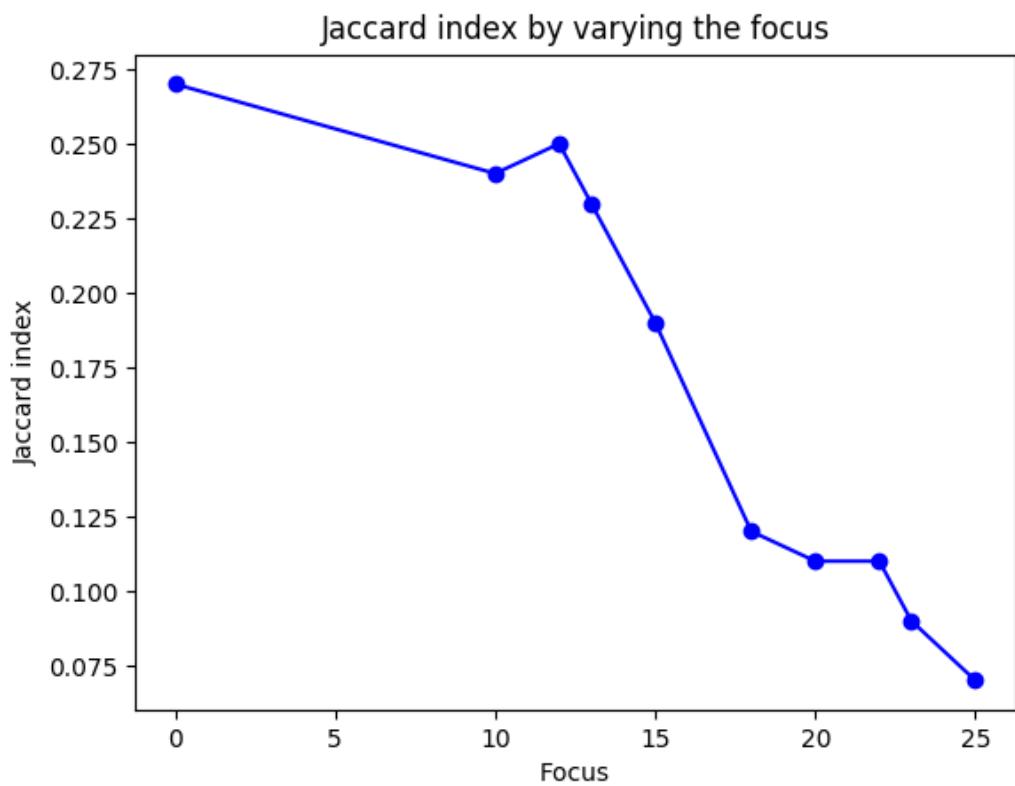


Figure 10: Andamento del valore medio del coefficiente di Jaccard a seconda dei vari valori di focus. In alto, i risultati con allineamenti di variazione percentuale inferiore al 10%, in basso con soglia a 5%.



Figure 11: Cattura verso l'esterno in contesto mattutino dal laboratorio di A.I. di S.Marta. A scalare, dall'alto verso il basso abbiamo una cattura con focus 0, una con focus 15 e una con focus 20.

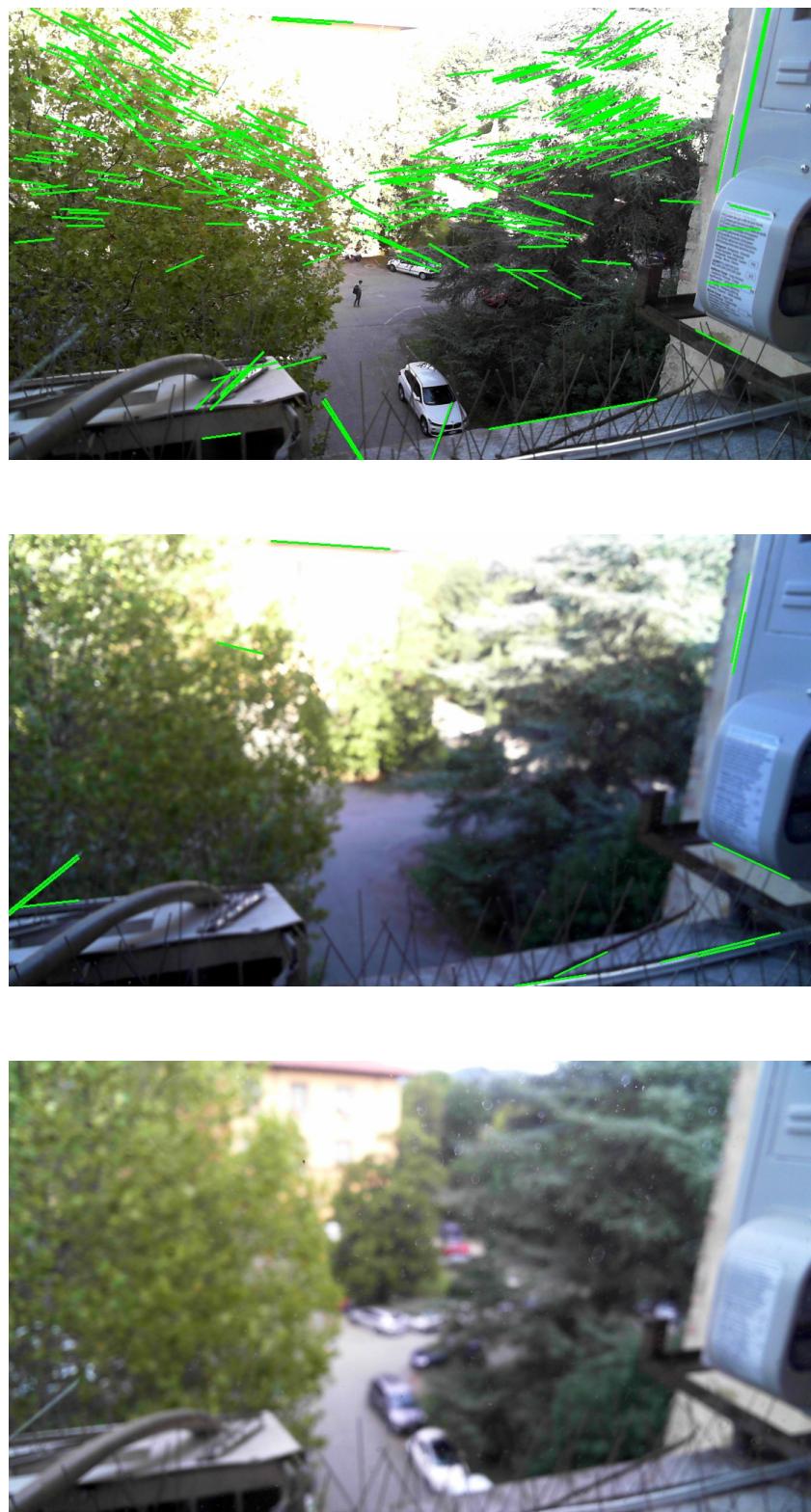


Figure 12: Cattura verso l'esterno in contesto pomeridiano dal laboratorio di A.I. di S.Marta. A scalare, dall'alto verso il basso abbiamo una cattura con focus 0, una con focus 15 e una con focus 20.

Fonti e links utili

- <https://opencv.org/>
- https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html
- https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html
- <https://brilliant.org/wiki/hungarian-matching/>
- <http://software.clapper.org/munkres/>
- <https://www.pyimagesearch.com/2015/04/06/zero-parameter-automatic-canny-edge-detection-with-python-and-opencv/>
- https://www.researchgate.net/publication/257877477_On_the_feasibility_of_a_channel-dependent_scheduling_for_the_SC-FDMA_in_3GPP-LTE_mobile_environment_based_on_a_prioritized-bifacet_Hungarian_method