UNIVERSITA' DEGLI STUDI DI PADOVA

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

RILEVAMENTO DELLA POSIZIONE DELLA MANO TRAMITE IMMAGINI E DATI 3D

HAND POSITION'S DETECTION USING IMAGES AND 3D DATA

Laureando: Alessio Marzo

Relatore: Prof. Guido Maria Cortelazzo

Correlatori: Prof. Pietro Zanuttigh, Ing. Carlo Dal Mutto

Anno Accademico 2010-2011

Sommario

Il riconoscimento dei movimenti tramite elaborazione di immagini e video sta diventando sempre più una necessità in molti ambiti diversi e le applicazioni che lo sfruttano si sono molto diffuse recentemente (un esempio concreto è Kinect© della Microsoft, usato in ambito videoludico). In questo campo, molto importante è la branca della Gesture Recognition, ossia lo studio, attraverso l'uso di un'opportuna strumentazione, delle tecniche di riconoscimento dei gesti delle mani catturate da immagini e video. Questa tesi di laurea vuole mostrare come sia stato raggiunto uno dei principali obbiettivi di questo ramo del riconoscimento di movimenti: l'individuazione della posizione della mano a partire da uno stream di immagini e di informazioni sulla profondità. L'attività di progetto si è articolata in passi successivi che hanno portato allo sviluppo di un algoritmo finale che integra tutti gli aspetti analizzati e studiati nel corso del lavoro di tesi.

La relazione comincerà con il descrivere una fase di studio preliminare sul colore di una serie di immagini acquisite, al fine di isolare i pixel della pelle dai restanti (fase di *Skin Detection*). Seguirà, quindi, una parte relativa all'analisi dell'informazione sulla profondità degli oggetti acquisiti con un sensore a tempo di volo (*Time Of Flight*) per capire come sfruttare questo nuovo dato al fine di individuare più precisamente la posizione della mano rispetto ad altri elementi dell'immagine che il solo colore non riesce a rimuovere. Infine, verrà presentato l'algoritmo finale ottenuto, il quale fonde le informazioni relative al colore e alla profondità riuscendo ad isolare la mano dal resto della scena inquadrata.

Ogni passo enunciato nella relazione prevede, nel seguente ordine, la descrizione del problema e le modalità di acquisizione dei dati, la soluzione proposta ed i risultati ottenuti. La tesi si concluderà con una disamina degli obbiettivi raggiunti e con la proposta di una serie di spunti per implementazioni e scopi futuri dei quali questo progetto può considerarsi il punto di partenza.

Indice

1 Utilizzo dell'informazione colore e Skin Detection **1**
1.1 Acquisizione dei dati e analisi degli spazi di colore 1
1.2 Algoritmo proposto e sua implementazione 2

2 Uso dell'informazione di profondità **7**
2.1 Acquisizione dei dati 7
2.2 Algoritmo proposto e sua implementazione 8

3 Soluzione finale: fusione di colore e profondità **11**
3.1 Acquisizione dei dati 11
3.2 Algoritmo proposto e sua implementazione 13
3.2.1 Creazione delle corrispondenze sensore-camera 14
3.2.2 Creazione della maschera basata sul colore 15
3.2.2.1 Uso di soglie fisse 15
3.2.2.2 Uso di soglie dinamiche 17
3.2.3 Individuazione del pixel della pelle a minima profondità 19
3.2.4 Creazione della maschera di profondità tramite procedura di espansione 20
3.2.5 Visualizzazione sull'immagine a colori e tracciamento dei contorni . . 22

4 Risultati finali e analisi computazionale **25**
4.1 Filtraggio tramite Skin Detector 25
4.2 Filtraggio basato sulla profondità 27
4.3 Filtraggio tramite depthColorFilter 28
4.4 Confronto tra gli algoritmi proposti 30
4.5 Tempi e performance dell'algoritmo finale 33

5 Possibili spunti e sviluppi futuri **37**
5.1 Possibili estensioni e raffinamenti dell'algoritmo sviluppato 37
5.2 Morfologia e modello 3D 38
5.3 Individuazione di entrambe le mani 38
5.4 Riconoscimento di gesti, motion tracking e Microsoft Kinect© 39

6 Conclusioni **41**

A Procedura di acquisizione tramite trigger hardware **43**

Capitolo 1

Utilizzo dell'informazione colore e Skin Detection

In questa sezione verrà trattato il primo passo del progetto svolto, riguardante l'individuazione, a partire da una serie di immagini acquisite con una fotocamera (risoluzione 1024x768), dei pixel della pelle ed il loro isolamento rispetto ai pixel restanti. L'obbiettivo principale consiste nel ridurre al massimo i cosiddetti "falsi negativi", ossia pixel che sembrano appartenere allo sfondo, ma che in realtà sono pixel della pelle, mantenendo bassi, al tempo stesso, i "falsi positivi", ossia pixel dello sfondo erroneamente riconosciuti come pixel della pelle. Per raggiungere questo scopo, è stato necessario uno studio sui diversi spazi di colore esistenti, al fine di individuarne uno adatto al tipo di elaborazione necessaria e che garantisse la miglior precisione nell'individuazione dei pixel. A partire da questo, sono state ricavate delle soglie di colore modellate su una serie di immagini di prova (immagini di *training*) e testate su altre immagini generiche acquisite (immagini di *test*). Il lavoro svolto in questa fase è stato realizzato con la collaborazione del dott. Marco Fraccaro, laureando triennale in Ingegneria dell'Informazione, il cui argomento di tesina è lo Skin Detection. Pertanto, in questa sezione verranno riportati solo gli aspetti più importanti dell'implementazione dell'algoritmo proposto ed i risultati più significativi[1].

1.1 Acquisizione dei dati e analisi degli spazi di colore

La scelta dello spazio di colore con cui operare risulta fondamentale sia in termini di prestazioni dell'algoritmo che dovrà usufruirne, sia in termini di accuratezza nell'individuazione dei pixel di interesse in diversi ambienti e con diverse illuminazioni. Per questo motivo, è stato necessario acquisire un certo numero di foto iniziali in diverse situazioni ambientali sulle quali effettuare uno studio approfondito. Le immagini impiegate raffigurano delle mani in diverse condizioni di ambiente e di illuminazione e per ciascuna di queste sono stati ricavati degli istogrammi che mostrano la distribuzione dei pixel con i diversi spazi di colore. Per lo studio in questione sono state acquisite 39 foto di training, usate anche per la successiva modellazione delle soglie di colore, e 125 foto di test, sulle quali verificare i risultati e testare l'algoritmo proposto.

[1] Per una dettagliata panoramica sul lavoro svolto in questa fase è possibile consultare la tesi del dott. Marco Fraccaro "Analisi del colore per il riconoscimento della mano".

Figura 1.1: Alcuni esempi di immagini di training usate per lo studio.

Sono stati studiati i seguenti spazi di colore:

- RGB
- XYZ
- CIELAB
- CIELUV
- HSV
- YCbCr

Come detto, per ciascuno di questi sono stati ricavati degli istogrammi (figura 1.2), i quali riportano in ascissa i valori possibili di ogni componente dello spazio di colore, e in ordinata il numero di pixel dell'immagine per ogni valore della particolare componente. Sulla base dei confronti tra i grafici, è stato possibile individuare uno spazio di colore che permettesse di ricavare in maniera abbastanza semplice delle soglie accurate per il colore della pelle.

Dopo un'attenta analisi, la scelta è ricaduta sullo spazio CIELAB: la distribuzione mostrata dagli istogrammi ha evidenziato una maggiore adattabilità alle diverse situazioni di illuminazione e una maggiore facilità nell'isolare le porzioni di pelle nelle immagini grazie ad una concentrazione di pixel in insiemi piuttosto ristretti. L'uniformità dello spazio, inoltre, permette di utilizzare al meglio anche il concetto di *distanza euclidea tra i colori*, utile all'algoritmo proposto per escludere pixel con colore diverso da quello della pelle. Tuttavia, quest'ultimo aspetto comporta un più alto costo computazionale, compensato dall'importanza dei risultati ottenuti. Una volta determinato lo spazio di colore su cui lavorare, le immagini di training sono state sfruttate per ricavare delle soglie di colore per le tre componenti L, a, b (delle quali le ultime due sono le più significative) derivanti dagli intervalli di pixel evidenziati dagli istogrammi. In seguito, verrà mostrato come tali soglie risultino fondamentali per la determinazione delle maschere di colore con cui filtrare le immagini acquisite.

1.2 Algoritmo proposto e sua implementazione

L'algoritmo di identificazione dei pixel della pelle si basa sulla tecnica dell'*hysteresis thresholding* (sogliatura a isteresi) ed è stato realizzato in Matlab, con l'ausilio dei suoi metodi di filtraggio; una sua versione ridotta è stata, poi, realizzata in C++ sfruttando le librerie OpenCv. A partire dallo spazio di colore scelto, che nel nostro caso è il CIELAB, sono stati definiti due diversi insiemi di pixel, ottenibili con soglie di colore più o meno restrittive:

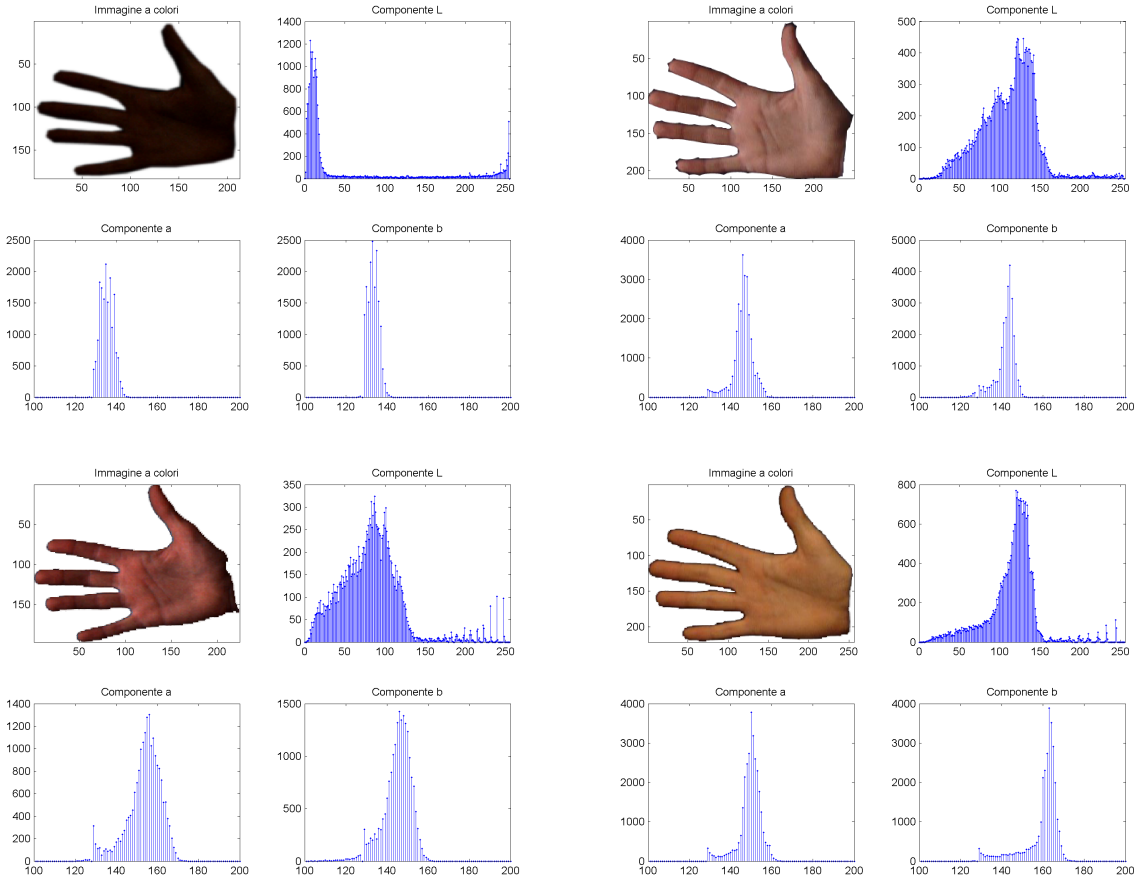


Figura 1.2: Alcuni esempi di istogrammi usati per ricavare le soglie di colore.

- **Specific Set of Values** (*SSV*), che contiene i pixel che quasi certamente sono pixel della pelle. I valori che lo compongono sono scelti sulla base degli intervalli evidenziati dagli istogrammi in modo da costituire vincoli più restrittivi e maggiormente legati al colore della pelle. Per adattarsi al meglio alle diverse situazioni, lo SSV è un insieme derivante dall'unione di intersezioni di piccoli sottoinsiemi, dovuti alle soglie ricavate.

- **Extended Set of Values** (*ESV*), che contiene pixel che appartengono alla pelle con buona probabilità. I vincoli imposti da questo insieme sono meno restrittivi di quelli dello SSV e per questo possono includere anche un numero (comunque non troppo elevato) di falsi positivi. La condizione ideale per ottenere l'ESV è quella di considerare l'unione di tutti i valori presenti in tutti gli istogrammi ricavati dalle immagini di training. Ovviamente, per come è definito, l'ESV contiene lo SSV dal punto di vista insiemistico.

I due insiemi definiti costituiscono il fulcro dell'operazione di filtraggio dell'algoritmo, suddivisa in molteplici passi che nel complesso portano a rimuovere quasi tutte le porzioni dell'immagine che non fanno parte della pelle (salvo rari casi in cui le condizioni dell'ambiente e l'illuminazione non permettono un adeguato filtraggio). Innanzitutto, lo SSV viene ridotto il più possibile per fare in modo che il numero di pixel contenuti sia molto basso (figura 1.4); per questo motivo, all'immagine viene applicato un iniziale filtraggio gaussiano combinato con una successiva sogliatura, teso ad eliminare quelle regioni di pixel che non contengono un

numero sufficiente di pixel appartenenti allo SSV. La funzione `fspecial` di Matlab permette di definire un filtro gaussiano basato su una matrice quadrata di coefficienti (maschera) che viene mossa su tutti i punti presenti nell'immagine e i cui valori vengono moltiplicati per quelli racchiusi nel riquadro su cui ci si è mossi. L'immagine rappresentante i pixel dello SSV viene considerata come una matrice di valori 0 (se il pixel non è dello SSV) e 1 (se il pixel è dello SSV), in modo tale che, alla fine dell'operazione di filtraggio, questa possa essere moltiplicata per l'immagine di partenza e usata in maniera più semplice. Lo stesso discorso vale per la matrice dell'ESV.

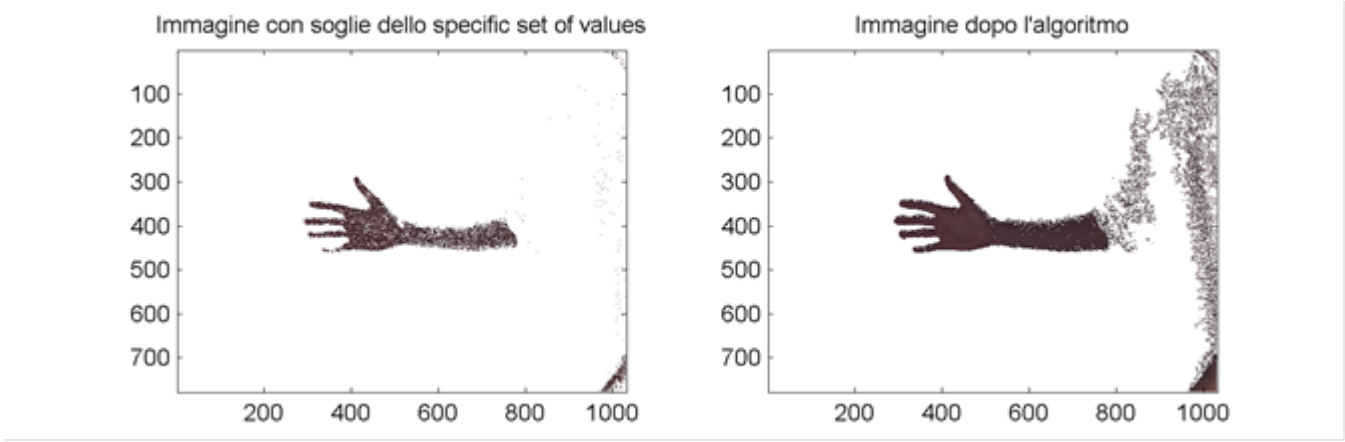


Figura 1.3: Come si vede, il mancato utilizzo del filtro gaussiano porta lo SSV a contenere pixel non appartenenti alla pelle che verranno erroneamente presi in esame dall'algoritmo.

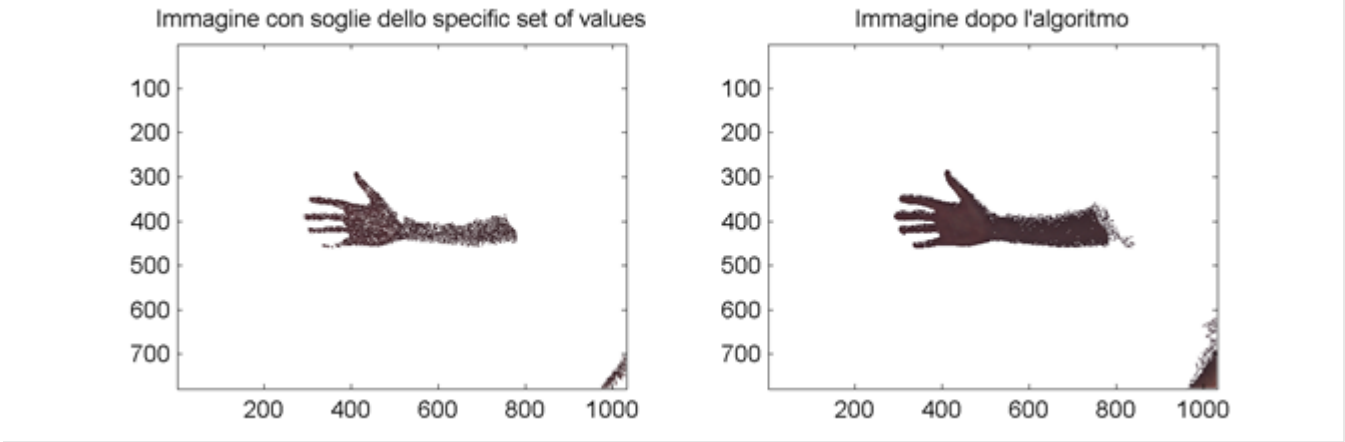


Figura 1.4: La figura mostra l'effetto del filtro gaussiano sui pixel sparsi dello SSV; come si nota, rispetto alla figura 1.3 il risultato finale è decisamente migliore.

Terminata l'eliminazione dei pixel sparsi, l'algoritmo procede con la fase principale di espansione. La procedura in questione prevede, ad ogni iterazione, di considerare l'i-esimo pixel dello SSV, del quale viene memorizzato il colore per i confronti futuri, e di espandersi da questo andando ad includere eventuali pixel adiacenti che appartengono a porzioni di pelle. L'inclusione di un pixel avviene sulla base di considerazioni quali l'appartenenza del nuovo valore allo SSV ed il suo colore (utile al calcolo di un colore di riferimento facendo la media

tra i colori dei pixel a confronto), oppure la sua appartenenza all'ESV e la distanza di questo dal valore preso in esame nell'iterazione corrente. L'aggiunta di nuovi pixel dovuta all'espansione porta a ripetere lo stesso procedimento su questi appena inseriti, finché non si hanno più pixel dell'ESV da analizzare. Un controllo finale viene, dunque, fatto sui pixel distanti che l'espansione non è riuscita a raggiungere, di modo da inserirli nell'immagine finale se necessario, e su quelli non ancora analizzati.

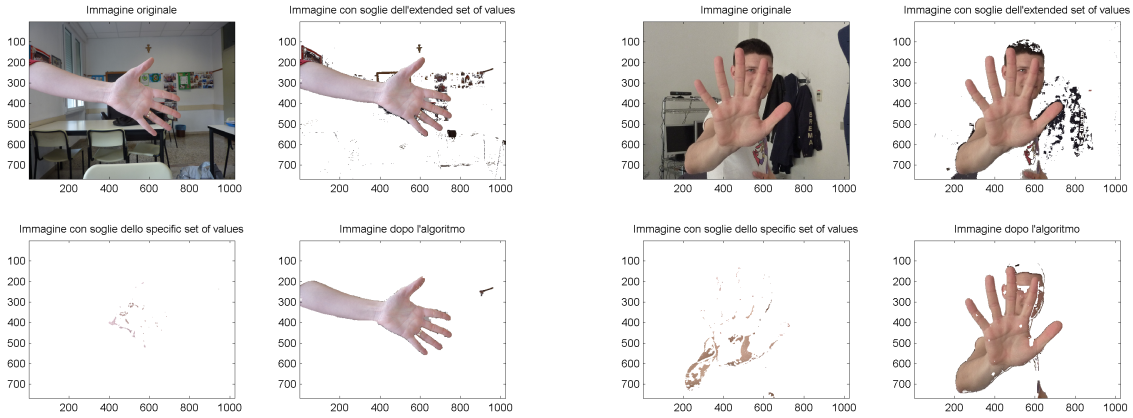
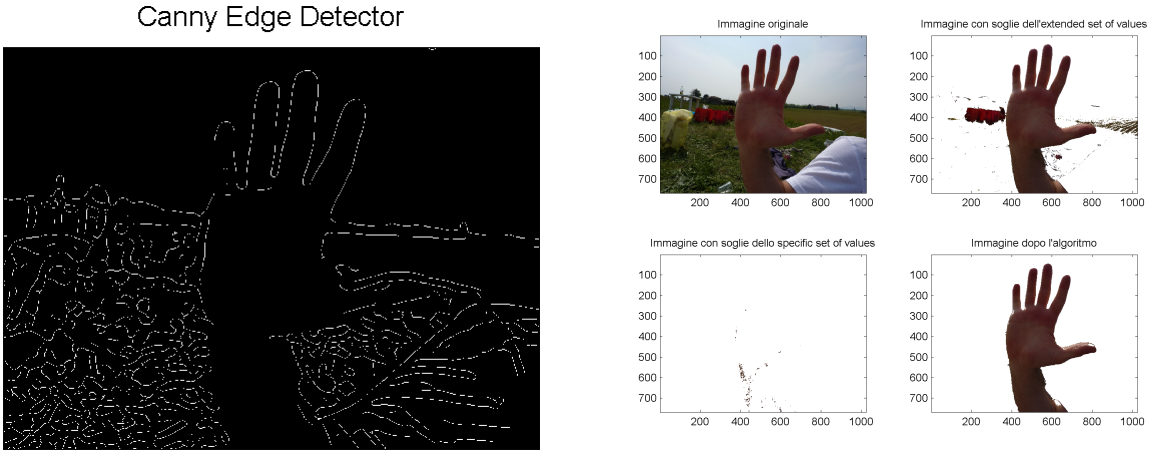


Figura 1.5: Alcuni risultati ottenuti usando la tecnica dell'espansione dei pixel in base alla distanza tra i colori.

Al fine di raffinare ulteriormente l'algoritmo e migliorarne i risultati, si è scelto di implementare anche un controllo relativo a quei pixel che appartengono ai contorni di porzioni di pelle (*edge*), in modo da fermarne l'espansione in caso si ecceda da questi. Anche in questo caso si è deciso di sfruttare una funzione di Matlab per l'individuazione di questi edge: si è,

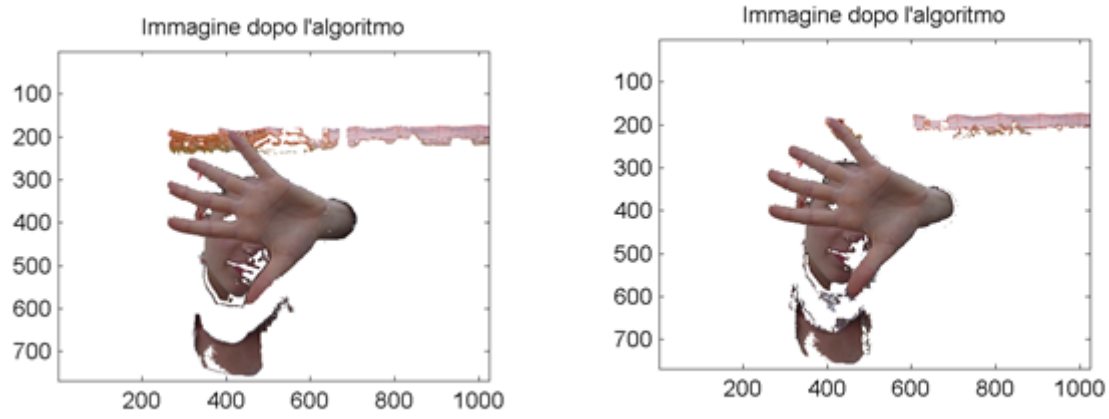


(a) (b)

Figura 1.6: L'immagine in figura 1.6a rappresenta il risultato dell'algoritmo di Canny per l'individuazione dei contorni. Questo viene impiegato nello Skin Detector per la rimozione dei pixel appartenenti agli edge, come si vede in figura 1.6b, nonché per fermare il processo di espansione se si superano i contorni di una porzione di pelle.

infatti, scelto di usare il metodo `edge` specificando come algoritmo quello di **Canny** (figura 1.6), che, a partire da un'immagine RGB, ne restituisce una in bianco e nero evidenziando i contorni delle figure rappresentate. In base ad alcuni parametri impostabili dall'utente (soglie inferiore e superiore e deviazione standard) è possibile rendere il calcolo degli edge più o meno restrittivo, a seconda degli scopi. L'adozione dei contorni ha permesso di ottenere risultati più precisi, grazie all'esclusione di pixel non della pelle difficili da rimuovere con la sola procedura di espansione (in particolare, quelli aventi un colore simile a quello della pelle), ma eliminabili se appartenenti ad un edge o se situati al di là di questo.

Tuttavia, l'utilizzo di questa nuova informazione ha portato all'instaurarsi di un nuovo problema legato ai collegamenti mancanti tra segmenti di contorni: i buchi creati da questi possono portare l'espansione a continuare anche oltre il limite e ad includere pixel non appartenenti alla pelle. Questo problema, detto di *Edge Linking*, è stato in parte risolto andando in un primo momento ad ispessire gli edge calcolati (soluzione poco apprezzabile), metodo che però è stato sostituito dall'utilizzo della funzione `imclose` di Matlab andando ad aggiungere segmenti rettilinei mancanti tra zone di punti ad una distanza stabilita dal programmatore. Quest'ultima implementazione, più apprezzabile rispetto ad un ispessimento dei contorni che potrebbe alterare le forme degli oggetti rappresentati, ha portato a risultati leggermente migliori in alcuni casi, mentre in altri la situazione è rimasta pressoché invariata.



(a) (b)

Figura 1.7: Rispetto alla figura 1.7a in cui sono presenti porzioni di immagine non appartenenti alla pelle, la figura 1.7b mostra un risultato leggermente migliore grazie all'implementazione dell'Edge Linking.

Capitolo 2

Uso dell'informazione di profondità

In questa sezione verrà trattato il secondo passo dell'attività di progetto, riguardante lo studio del fattore profondità negli oggetti ripresi da un sensore a tempo di volo (*Time Of Flight*) presente in laboratorio. Essendo l'obbiettivo finale quello di creare un'applicazione che individui il più precisamente possibile la posizione della mano di fronte ad una sorgente di acquisizione, l'ipotesi da cui si è partiti è che questa fosse l'elemento più vicino al sensore tra tutti quelli ripresi. Tale assunzione risulta piuttosto verosimile per gli scopi del progetto, anche se in seguito, sfruttando il colore, verrà mostrato come eliminare questo vincolo che, seppur corretto, è in alcuni casi troppo stringente. L'idea è, dunque, quella di individuare il punto a minore profondità ripreso dal sensore TOF e costruire, a partire da questo, una regione tridimensionale (che funge da maschera) comprendente solo la mano, in modo da tracciarne un contorno rettangolare che ne stabilisca la posizione.

2.1 Acquisizione dei dati

Come detto in precedenza, per l'acquisizione dei dati relativi alla profondità ci si è serviti di un sensore TOF e di un tool software ufficiale che, a partire dalla ripresa della scena, restituisce in output una serie di file .dat contenenti diverse matrici di valori riguardanti, tra l'altro, profondità e distanze x e y rispetto al centro ottico. Le immagini acquisite sono frame di uno stream di dati in uscita dal tool di acquisizione e rappresentano un soggetto con una mano più vicina al sensore rispetto al corpo in diverse posizioni e che compie gesti differenti; la risoluzione delle immagini catturate dal dispositivo è di 176x144.

Figura 2.1: Alcuni esempi di frame acquisiti dal sensore di profondità.

La matrice restituita contiene i valori sulle distanze, compresi tra 0 e 5 e misurati in metri; questa può essere convertita in un'immagine in scala di grigi, nella quale gli elementi più

vicini al sensore risultano essere più scuri rispetto al resto. In seguito, verrà illustrato come questo dato, assieme alle distanze x e y, sarà fondamentale per il filtraggio delle immagini al fine di isolare la mano. Complessivamente, è stato acquisito uno stream di 1000 frame del quale è stato fatto un parsing per creare le singole matrici di valori per ciascuna immagine.

2.2 Algoritmo proposto e sua implementazione

Come nel caso dello Skin Detector, anche per questa fase si è scelto di lavorare in Matlab inizialmente, in quanto è risultato più semplice convertire le matrici in immagini e lavorare con le maschere. Successivamente, l'algoritmo è stato tradotto in C++ con l'ausilio delle librerie OpenCv ed il suo funzionamento risiede in un metodo chiamato `depthFilter`. Partendo dal presupposto, come anticipato, che la mano sia la cosa più vicina al sensore, l'obbiettivo è quello di riuscire ad individuare le coordinate del punto nella matrice di profondità (che chiameremo *depth map*) con il minor valore possibile. L'algoritmo proposto, quindi, fa una scansione dell'intera immagine e salva il pixel con la minore profondità; i valori uguali a 0 nella depth map, corrispondenti a punti saturi dovuti alla troppa vicinanza di oggetti al sensore, vengono eliminati perché non significativi. Ottenuto questo, viene creata una prima maschera (di valori 0 e 1) contenente tutti quei punti la cui profondità rientra entro due soglie, una massima e una minima, scelte staticamente sulla base delle reali misure dello spessore della mano (nel caso peggiore, del pugno chiuso). Successivamente, al fine di eliminare eventuali pixel che rientrano nell'intervallo fissato, ma che non appartengono alla mano, si individua il baricentro dei punti fin qui isolati (semplicemente facendo la media aritmetica delle coordinate x e y) e da questo si ricavano due ulteriori maschere riguardanti le componenti x e y delle distanze. Queste ultime definiscono rispettivamente una fascia verticale e una orizzontale che, intersecate, generano una regione bidimensionale che racchiude la mano; questa, unita alla componente z ricavata dalla prima maschera, forma una regione tridimensionale che racchiude la zona interessata. Per completezza, l'algoritmo crea, infine, un riquadro rettangolare di colore rosso intorno alla mano che, dinamicamente, adatta la sua dimensione in ogni frame a seconda dei pixel racchiusi nella maschera di profondità.

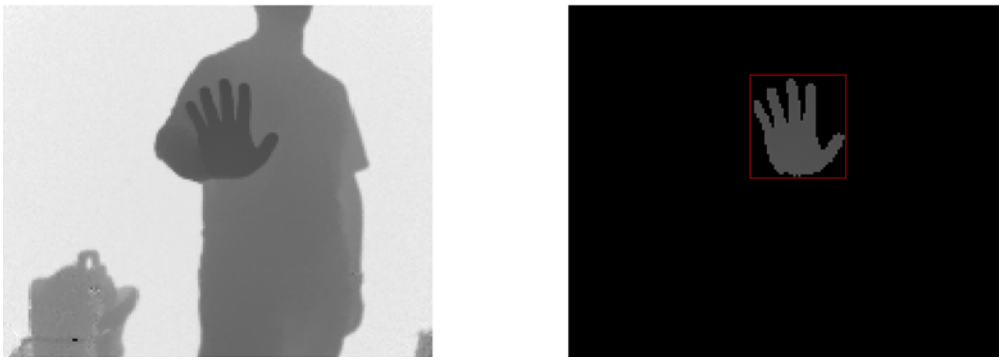


Figura 2.2: Immagine che mostra il risultato dell'algoritmo agente sui singoli frame.

Va riportato il fatto che il metodo depthFilter realizzato in C++ ha un funzionamento leggermente diverso, in quanto fa riferimento alla sola depth map senza alcuna informazione sulle distanze x e y. In tal caso, la creazione della regione tridimensionale che racchiude la mano viene fatta sulla base di prove di inclusione di un numero sempre maggiore di pixel

nella maschera di profondità finché non si ottiene il risultato sperato. Pertanto, la versione in C++ risulta meno precisa di quella realizzata in Matlab, ma comunque utile allo studio.

Capitolo 3

Soluzione finale: fusione di colore e profondità

L'ultimo passo dell'attività di progetto consiste nell'utilizzare congiuntamente i risultati ottenuti nei passi precedenti per isolare ancora più precisamente la mano dal resto dell'immagine. In questa sezione verrà spiegato come è stato possibile integrare l'informazione sul colore, analizzata con lo Skin Detector realizzato, all'interno dell'algoritmo che riconosce l'oggetto a minore profondità, integrazione soprattutto utile per eliminare eventuali ostacoli che si frappongono tra mano e sorgente di acquisizione. La soluzione proposta ha permesso, inoltre, di tracciare i contorni della mano individuata, nella maggior parte dei casi, in modo da visualizzarne al meglio la posizione all'interno della scena.

3.1 Acquisizione dei dati

Per quest'ultima fase di progetto, l'acquisizione dei dati ha richiesto una maggiore attenzione rispetto ai casi precedenti, in quanto sono necessarie due informazioni (colore e profondità) e non più una sola delle due. Tali informazioni derivano da due fonti di acquisizione diverse (rispettivamente videocamera e sensore TOF), perciò è richiesto che vi sia una corrispondenza di valori tra ciò che viene catturato dalla videocamera e ciò che deriva, invece, dal sensore di profondità. Per questo motivo, è stata richiesta una procedura di calibrazione iniziale della strumentazione presente in laboratorio (mostrata in figura 3.1), seguita dall'acquisizione vera e propria delle immagini in diverse situazioni di posizione della mano e degli oggetti che la circondano.

Figura 3.1: Foto del sistema di acquisizione usato in laboratorio.

Utilizzando un apposito tool software di acquisizione, sono stati catturati circa 500 frame sui quali testare l'algoritmo sviluppato; la scena inquadrata vede il soggetto con la mano distante o vicina al corpo, in diverse posizioni, con o senza oggetti che si frappongono tra di essa e la sorgente di acquisizione (figura 3.2). Nonostante, inizialmente, l'obbiettivo del progetto fosse quello di determinare la posizione della mano assumendo che non ci fossero oggetti di fronte a lei, l'integrazione del colore ed il filtraggio per ottenere i pixel della pelle permettono di analizzare anche alcuni casi limite che, nella maggior parte dei casi, non influenzano più di tanto il risultato. Le condizioni di illuminazione sono pressoché le stesse, anche se, come verrà spiegato, ciò non rappresenta un problema per il modo con cui le immagini vengono elaborate.

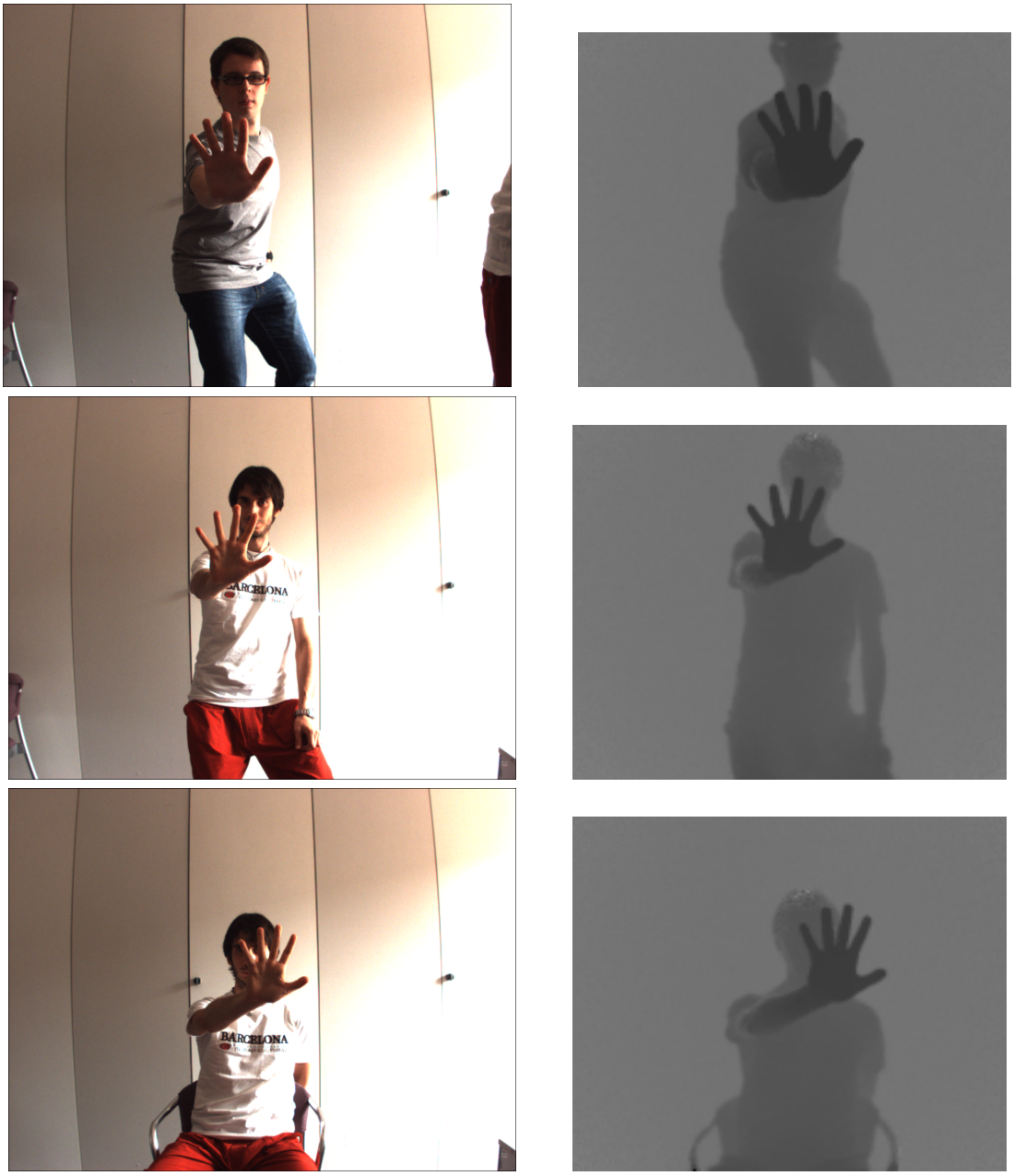


Figura 3.2: Alcuni esempi di frame acquisiti.

Le immagini riprese dalle videocamere hanno una risoluzione di 1032x778, molto superiore a

quella del sensore TOF (che, come detto, è di 176x144). Tuttavia, questo non rappresenta un problema, in quanto la procedura di calibrazione fornisce appositi parametri utili al passaggio da una sorgente all'altra. Nel paragrafo 3.2 verrà mostrato come l'operazione di conversione dei punti del sensore in punti della videocamera sia resa possibile da appositi metodi in C++.

3.2 Algoritmo proposto e sua implementazione

Per quest'ultimo passo di progetto si è scelto di realizzare l'algoritmo direttamente in C++ per poter lavorare con più facilità e con l'ausilio delle librerie OpenCv. Il fulcro della soluzione proposta è rappresentato dal metodo `depthColorFilter`, che effettua un filtraggio dell'immagine passatagli come parametro esplicito avendo informazioni sul colore e sulla profondità, quest'ultima in uscita dal sensore TOF. Il sistema di acquisizione prevede, come detto, l'impiego di un sensore di profondità e di due videocamere (situate a destra e a sinistra del sensore); tuttavia, per gli obbiettivi prefissati è stato sufficiente concentrarsi sulle immagini provenienti da una sola delle due camere, quella destra nel nostro caso.

L'algoritmo sviluppato effettua, nell'ordine, le seguenti operazioni, che verranno spiegate in dettaglio nel corso di questo paragrafo:

1. Usando due opportuni metodi ausiliari, viene creata una corrispondenza tra i punti acquisiti dal sensore TOF e quelli della videocamera (procedura ottenuta grazie ai parametri di calibrazione). Questo passo viene fatto al di fuori del nucleo del metodo depthColorFilter per non peggiorarne le prestazioni complessive.

2. Viene creata una maschera basata sul colore che isola i pixel appartenenti alla pelle dal resto dell'immagine, servendosi di opportune soglie di colore. Questa maschera è una *CvMat* di dimensioni pari a quelle delle immagini della videocamera e costituita da valori 0 e 1.

3. Attraverso una scansione completa dell'immagine del sensore TOF, viene individuato il punto di minima profondità che, grazie all'aiuto contemporaneo della maschera sul colore ricavata al punto 1, con buona probabilità appartiene alla mano. In questo caso, si sfruttano le corrispondenze tra punti del sensore TOF e punti della videocamera.

4. Ottenuto il punto di minima profondità, a partire da questo, tramite una procedura di espansione di punti, viene creata una maschera di profondità che, con buona approssimazione, isola la mano dal resto della scena inquadrata dal sensore. Anche in questo caso si ottiene una CvMat di valori 0 e 1 di dimensioni 176x144.

5. Infine, utilizzando nuovamente le informazioni relative alle corrispondenze di punti, dalla maschera di profondità ottenuta al punto 4 si ricava una nuvola di punti, nell'immagine a colori, che ricopre la mano, e di questa si traccia il contorno.

Il metodo realizzato ha subito diversi raffinamenti e modifiche nel corso dell'attività di progetto; in particolare, la maschera sul colore è ottenuta con due diversi procedimenti: il primo prevede l'utilizzo delle soglie ricavate per lo Skin Detector, descritto al paragrafo 1.2; il secondo, più apprezzabile, estende il primo grazie allo sfruttamento di un primo frame per la "calibrazione del colore" direttamente a partire dalle condizioni di illuminazione della scena inquadrata. Nel corso di questo paragrafo verranno illustrate in dettaglio entrambe le versioni mettendo in evidenza i benefici delle modifiche apportate nel corso del lavoro.

Figura 3.3: Schema del procedimento seguito dall'algoritmo.

3.2.1 Creazione delle corrispondenze sensore-camera

Come accennato nel paragrafo 3.1, per poter lavorare con entrambe le informazioni di colore e profondità è necessario che ci sia una corretta corrispondenza tra le coordinate dei punti catturati dal sensore TOF e quelle dei punti della videocamera, in quanto lo sfruttamento delle due informazioni in maniera congiunta richiede un continuo passaggio da un sistema all'altro per identificare i pixel della mano che sono anche alla minima profondità. Per tale motivo, depthColorFilter riceve una serie di parametri in ingresso relativi ad una preliminare fase di calibrazione, i quali vengono passati al metodo `tof2camera` che, a sua volta, li passa a due funzioni ausiliarie che restituiscono i dati necessari a passare dal sistema di riferimento del sensore a quello della videocamera. Di seguito vengono riportate le firme dei due metodi in questione utili alla calibrazione iniziale, ciascuno con i rispettivi parametri:

- `tofCreate3D(CvMat* tof_dist_, IplImage* tof_amp_, IplImage* tof_conf_, CvMat* intrTof_, CvMat* distTof_, vector<CvPoint3D32f>** tof3D_v_)`

  Questo metodo restituisce nel vettore *tof3D_v_* (il cui indirizzo è passato come parametro) i punti dello spazio della scena catturati dal sensore TOF ciascuno con le proprie coordinate tridimensionali (x,y,z) a partire dalle immagini delle distanze (*tof_dist_*), dell'ampiezza (*tof_amp_*), della confidenza (*tof_conf_*), dalla matrice dei parametri intrinseci del sensore (*intrTof_*) e da quella di distorsione del sensore (*distTof_*). L'obbiettivo di questo metodo è, quindi, quello di creare una prima corrispondenza tra i punti cartesiani delle immagini riprese dal sensore e i punti reali tridimensionali dello spazio della scena. Non volendo entrare troppo nel dettaglio dei parametri utili al metodo per ottenere il risultato, che vanno al di là dell'obbiettivo di questa relazione, basti sapere che sono informazioni che il tool di acquisizione restituisce a partire dalla sorgente di cattura basata su sensore e videocamere destra e sinistra e che sono utili a calibrare tale sistema trinoculare per individuare le corrette corrispondenze di punti.

- `projection(vector<CvPoint3D32f> points3D_v_, CvMat* intrinsic_, CvMat* estrinsic_, vector<CvPoint2D32f>& points2D_v_)`

  Come suggerisce il nome stesso del metodo, esso serve a proiettare i punti in uscita dal metodo tofCreate3D precedente sul sistema di riferimento della videocamera, ottenendo punti con coordinate cartesiane (u,v) nello spazio dell'immagine(questi punti vengono

salvati nel vettore *points2D_v_*). I parametri passati al metodo per ottenere la corrispondenza finale sono relativi a matrici di valori rappresentanti ulteriori informazioni necessarie alla calibrazione e derivanti dalla teoria della conversione di punti da uno spazio 3D ad uno 2D. Anche in questo caso, evitiamo di entrare nel dettaglio di tali parametri e ci limitiamo a riportare il significato dei nomi: la matrice *estrinsic_* contiene i valori estrinseci per passare dal sistema di riferimento del sensore TOF a quello della camera, mentre la matrice *intrinsic_* contiene i parametri intrinseci del sensore di profondità, utili ad ottenere le coordinate cartesiane finali nello spazio dell'immagine.

Tutte le CvMat di parametri delle quali questi metodi si servono sono state ricavate da un apposito tool software di calibrazione che è stato impiegato prima della vera e propria acquisizione; le *IplImage* in uscita dal tool di acquisizione e le matrici adoperate dai due metodi ausiliari di calibrazione sono passati, tra gli altri, come parametri al metodo depthColorFilter. I metodi descritti integrano, inoltre, alcuni passi ulteriori legati all'antidistorsione e alla rettificazione delle immagini acquisite, in modo che la corrispondenza di punti sia la più precisa possibile. Il vettore in uscita dalla funzione tof2camera, contenente le corrispondenze tra punti del sensore e punti della camera, viene memorizzato in una variabile statica in modo da poter essere usato in punti diversi della classe creata e calcolato una sola volta per ogni frame.

3.2.2 Creazione della maschera basata sul colore

Il passo successivo si ottiene con l'ausilio di una versione ridotta dello Skin Detector, illustrato nella sezione 1.2, che è stata appositamente scritta in C++. La creazione della maschera sul colore viene permessa dal metodo `colorFilter`, il quale si serve di soglie di colore relative allo spazio CIELAB che permettono di individuare, abbastanza precisamente, pixel della pelle in situazioni ambientali diverse. Inizialmente, tali soglie erano fisse e indipendenti dal tipo di immagine passata come parametro; successivamente, si è però scelto di fare in modo che queste si adattassero alla scena grazie all'utilizzo di un primo frame di "calibrazione del colore" dal quale ricavare le informazioni sulle componenti di colore necessarie. Di seguito verranno illustrati entrambi i procedimenti.

3.2.2.1 Uso di soglie fisse

Il calcolo della maschera sul colore segue un procedimento analogo a quello dello Skin Detector realizzato in Matlab, con l'unica differenza che le soglie ottenute sono state leggermente rilassate e riadattate per soddisfare al meglio i requisiti di illuminazione e di ambiente; quello che si ha è, quindi, una CvMat delle dimensioni dell'immagine della videocamera (1032x778) con valori 1 in corrispondenza dei pixel con colore simile alla pelle e 0 altrimenti. Tale maschera si ottiene dal prodotto delle maschere create per ciascuna componente di colore (L, a, b), ottenute in base alle soglie fissate. I valori scanditi nell'immagine in CIELAB vengono confrontati con le soglie (meno selettive di quelle dello SSV descritto nel paragrafo 1.2); se questi rientrano negli intervalli predisposti, il corrispettivo punto della maschera viene posto a 1, mentre assume valore 0 negli altri casi.

Algorithm 3.1 *Procedura di creazione della maschera relativamente ad una generica soglia di colore.*

Siano M_L, M_a, M_b le maschere rispettivamente delle componenti L,a,b inizializzate a 0.
Sia M la maschera finale.
Sia p_i un generico pixel dell'immagineI in CIELAB di coordinate (x,y).
Siano $p_i.L$, $p_i.a$, $p_i.b$ le componenti L,a,b di p_i.
Siano l^j_min, l^j_max, a^j_min, a^j_max, b^j_min, b^j_max gli estremi di una soglia di colore j per le componenti L,a,b.
for each $p_i \in I$ do
  if ($p_i.L \geq l^j\_min$ AND $p_i.L \leq l^j\_max$) then
    $M_L(x,y) \leftarrow 1$
  if ($p_i.a \geq a^j\_min$ AND $p_i.a \leq a^j\_max$) then
    $M_a(x,y) \leftarrow 1$
  if ($p_i.b \geq b^j\_min$ AND $p_i.b \leq b^j\_max$) then
    $M_b(x,y) \leftarrow 1$
end for
$M \leftarrow M_L \times M_a \times M_b$
Rimozione pixel sparsi in M.
Rimozione pixel che appartengono agli edge in M.

Anche in questo caso, vengono impiegati un filtro gaussiano per la rimozione dei pixel sparsi (correttamente importato da Matlab) e l'algoritmo di Canny per la rimozione dei pixel che appartengono agli edge.

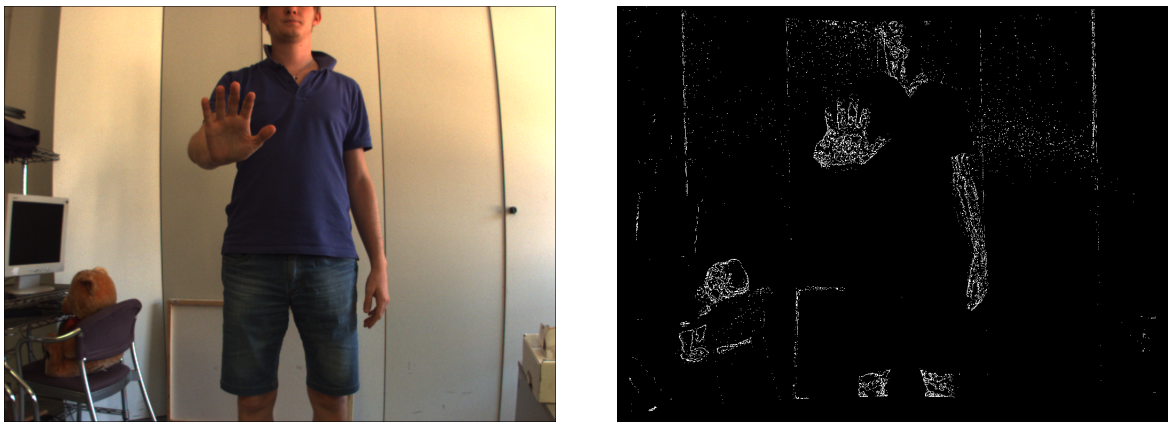


Figura 3.4: Esempio di maschera ottenuta usando il metodo colorFilter e soglie fisse.

Questa forma ridotta dell'algoritmo permette di ottenere il risultato sperato in minor tempo e con prestazioni migliori rispetto alla funzione in Matlab, in quanto si limita a fornire solo una parte dell'output complessivo che lo Skin Detector forniva. Tuttavia, va osservato che la maschera fornisce anche una serie di falsi positivi che una sua più completa versione in Matlab non avrebbe probabilmente considerato, ma ciò non comporta un grosso problema ai fini dell'algoritmo finale.

3.2.2.2 Uso di soglie dinamiche

L'utilizzo di soglie fisse ricavate dallo studio iniziale sul colore permette di creare una maschera sufficientemente precisa, ma a volte troppo selettiva; infatti, può accadere che la zona di punti presenti sulla mano risulti eccessivamente poco densa, tanto da compromettere il calcolo corretto del punto a minima profondità (che può ricadere in una zona dell'immagine lontana, per esempio il viso). Per ovviare a questo problema, si è scelto di implementare un secondo metodo di creazione della maschera a partire dall'idea di utilizzare il primo frame dello stream per modellare le soglie relative alle componenti a e b dello spazio di colore direttamente in base alle condizioni di illuminazione della scena inquadrata. Per tale motivo, si fa in modo che non vi siano oggetti frapposti tra la mano e la sorgente di acquisizione (figura 3.5), in modo da catturare con facilità il punto a minima profondità. La maschera che risulterà da questo procedimento, illuminazione permettendo, sarà sufficientemente precisa e, allo stesso tempo, evidenzierà una maggiore densità di pixel in corrispondenza della mano.

Figura 3.5: Esempi di frame usati per la calibrazione delle soglie di colore.

In termini di codice, questa scelta ha portato alla sovrascrittura del metodo colorFilter, al quale viene segnalata la volontà di impiegare il primo frame dello stream come frame di calibrazione del colore. Il nuovo metodo effettua le seguenti operazioni: sfrutta il metodo depthFilter (descritto al paragrafo 2.2) per ottenere una maschera di profondità iniziale sulla base del punto a minima profondità; quindi, sfruttando le corrispondenze di punti tra le immagini del sensore e della videocamera, analizza le componenti di colore dei pixel della mano isolata e ne calcola due valori medi (uno per la componente a e uno per la componente b, la componente L viene trascurata perché troppo vincolante) sulla base dei

quali modellare le due soglie finali. Quindi, la maschera sul colore viene relizzata in modo analogo al procedimento spiegato nel paragrafo precedente, evitando però di impiegare il filtro gaussiano e l'Edge Detection.

Una volta ottenuta la maschera di profondità dal metodo depthFilter (di dimensione 176x144), i valori medi delle componenti dei pixel nell'immagine in CIELAB, ottenuti grazie alle corrispondenze e ad un'interpolazione bilineare, vengono calcolati secondo il seguente procedimento, che si rifà alla *stima robusta.*

- Vengono creati due vettori **aComp** e **bComp** che, rispettivamente, contengono oggetti di tipo **structComp** che a loro volta contengono le coordinate cartesiane e il valore della componente a del pixel in esame (inserito in aComp) e della componente b del pixel in esame (inserito in bComp).

- I due vettori così ottenuti vengono, quindi, ordinati in maniera crescente in base ai valori delle componenti inserite e di questi vengono salvati i valori mediani **aMed** e **bMed**, corrispondenti agli oggetti presenti a metà di ciascun vettore.

- A partire dai valori mediani, si stimano in maniera robusta i valori medi **aMean** e **bMean** delle componenti di colore della mano a partire da quei pixel, appartenenti ai due vettori, che di poco si discostano dai valori mediani.

  Per ogni $p_i \in aComp$ e $q_i \in bComp$,

  $$\begin{cases} aMean = aMean + p_i.value & |p_i.value - aMed| < \delta_A \\ \\ bMean = bMean + q_i.value & |q_i.value - bMed| < \delta_B \end{cases}$$

  Al termine del ciclo,

  $$aMean = \frac{aMean}{countPix} \quad ; \quad bMean = \frac{bMean}{countPix}$$

  δ_A e δ_B sono due soglie fisse scelte in base ad una serie di prove e $countPix$ contiene il numero di pixel totali presenti nei due vettori.

- Ottenuti i valori medi, si passa infine alla creazione della maschera vera e propria, in cui viene fatta la differenza tra il valore della componente di ciascun pixel dell'immagine ed il corrispondente valore medio; se tale differenza non va oltre una certa soglia, il punto della maschera con le stesse coordinate del pixel scelto viene posto uguale a 1, 0 altrimenti. Anche in questo caso, vengono create due maschere separate, una per la componente a e una per la componente b, il cui prodotto è la maschera sul colore finale.

  Siano c_i un generico pixel dell'immagine in CIELAB e m_i^A e m_i^B i suoi corrispondenti rispettivamente nelle maschere per a e b. Allora

  $$\begin{cases} m_i^A = 1 & |c_i.a - aMean| < \delta_A^{'} \\ \\ m_i^A = 0 & altrimenti \end{cases}$$

$$\begin{cases} m_i^B = 1 & |c_i.b - bMean| < \delta_B' \\ \\ m_i^B = 0 & altrimenti \end{cases}$$

dove δ_A' e δ_B' sono due ulteriori soglie fisse ricavate da una serie di prove.

Le soglie fisse descritte nel paragrafo precedente vengono, dunque, aggiornate con quelle nuove, in modo tale da poter riutilizzare il primo colorFilter per i frame successivi, ma con le nuove soglie ottenute.

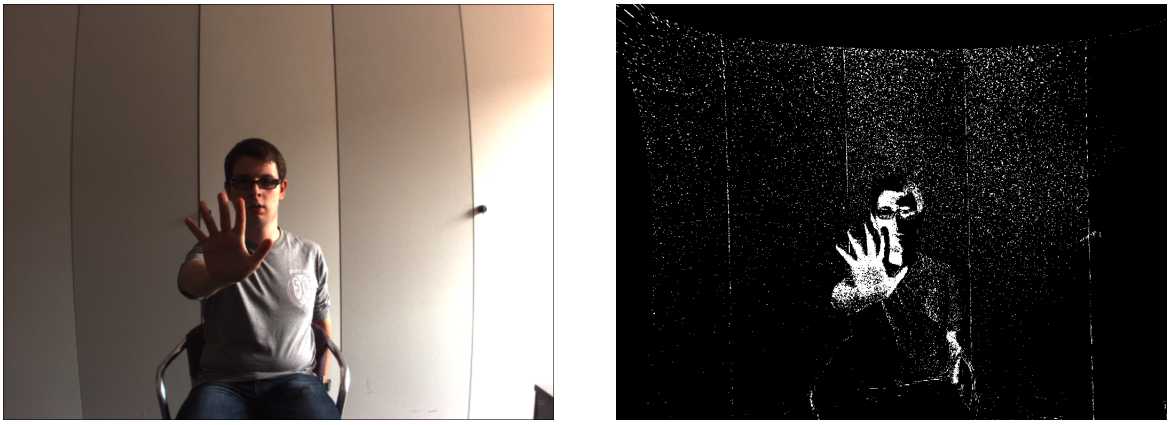


Figura 3.6: Maschera ottenuta con la stima robusta.

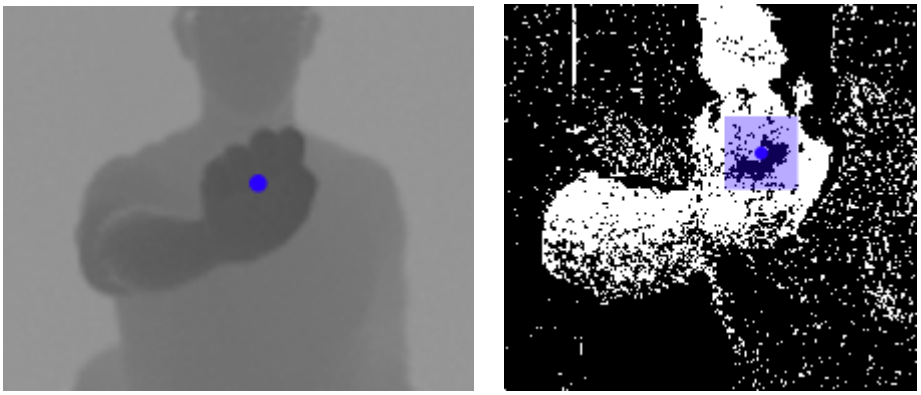
L'utilizzo di due sole soglie permette di ricavare una maschera più densa, ma meno selettiva rispetto al primo metodo, quindi in situazioni di illuminazione critiche si rischia di ottenere anche punti appartenenti allo sfondo. Tuttavia, le soglie fisse possono essere modificate per rendere la maschera più o meno selettiva a seconda delle esigenze.

3.2.3 Individuazione del pixel della pelle a minima profondità

In una situazione realistica, durante un'acquisizione alcuni oggetti tra la mano e le videocamere possono essere ripresi e finire all'interno dell'inquadratura. Escludendo il primo frame, utilizzato per la calibrazione del colore e nel quale si assume che non ci siano oggetti frapposti tra mano e sorgente di acquisizione, si vuole che l'algoritmo sia in grado di gestire situazioni più realistiche riuscendo comunque ad individuare la posizione della mano anche in caso di occlusioni in primo piano.

Per risolvere questo problema, il terzo passo consiste nella scansione dell'immagine del sensore TOF alla ricerca del valore minimo di profondità. La soluzione proposta prevede l'utilizzo di un ciclo for che prenda in esame ciascun pixel della depth map (di dimensione 176x144) e, individuando il suo corrispondente nella maschera sul colore, verifichi se questo appartiene o meno alla pelle; in caso affermativo, viene controllato se questo si trovi effettivamente in una zona piuttosto densa di pixel, per evitare di andare a considerare dei punti sparsi. Per far ciò, è stata fissata una soglia relativa al numero minimo di valori uguali a 1 (memorizzata nella variabile *pixTresh*) che devono essere contenuti in un quadrato di dimensione 30x30 che si sposta sulla maschera di colore: se tale quadrato, il cui centro è rappresentato dall'i-esimo punto dell'immagine della videocamera corrispondente al punto

corrente dell'immagine del sensore, racchiude un numero di valori 1 superiore alla soglia fissata, allora con buona probabilità si assume che esso faccia parte della pelle (figura 3.7). Se il punto del sensore TOF viene riconosciuto come pixel appartenente alla pelle, la sua profondità viene memorizzata e questa verrà confrontata con quelle analizzate nelle iterazioni successive, finché alla fine resterà il pixel che appartiene alla pelle e con la minore profondità rispetto agli altri (figura 3.8).



(a) (b)

Figura 3.7: La figura 3.7a mostra un generico punto della depth map scandito dal ciclo; intorno al suo corrispettivo nella maschera sul colore (figura 3.7b) viene tracciato il rettangolo all'interno del quale contare i valori 1.

Il ciclo si preoccupa anche di effettuare una serie di confronti tesi a ridurre i tempi di scansione: se un pixel con una certa profondità viene scartato, plausibilmente eventuali pixel adiacenti e con una profondità simile non dovranno essere considerati, in quanto con buona probabilità anche loro non apparterranno alla pelle. Alla fine del ciclo, vengono restituite le coordinate cartesiane del punto della depth map con profondità minima e il cui corrispondente nell'immagine della videocamera appartiene alla mano con buona approssimazione.

3.2.4 Creazione della maschera di profondità tramite procedura di espansione

Ottenuto il punto a minima profondità e che con buona probabilità appartiene alla pelle, ne vengono memorizzate le coordinate cartesiane nell'immagine del sensore TOF e viene creata una maschera di profondità, a partire dal valore del pixel riconosciuto, secondo un approccio a *clustering*.

I punti che costituiranno la maschera finale, tramite un ciclo for, vengono inseriti in una struttura a coda (denominata **depthPoints**) seguendo una tecnica di espansione dei pixel verso quelli confinanti, purché questi ultimi possiedano un valore di profondità rientrante in due soglie fisse, una massima (**maxDep**) e una minima (**minDep**) scelte sulla base di prove empiriche. Se un generico punto rientra in tali soglie, esso viene posto a 1 nella maschera di profondità finale e inserito nella coda, per poi essere esaminato nuovamente all'iterazione successiva; i nuovi vicini del punto preso in esame (rimosso dalla coda) verranno aggiunti se anch'essi rientreranno nell'intervallo di profondità fissato. L'utilizzo di una coda garantisce un'espansione in tutte le direzioni in maniera uniforme, preferibile alla struttura a stack che,

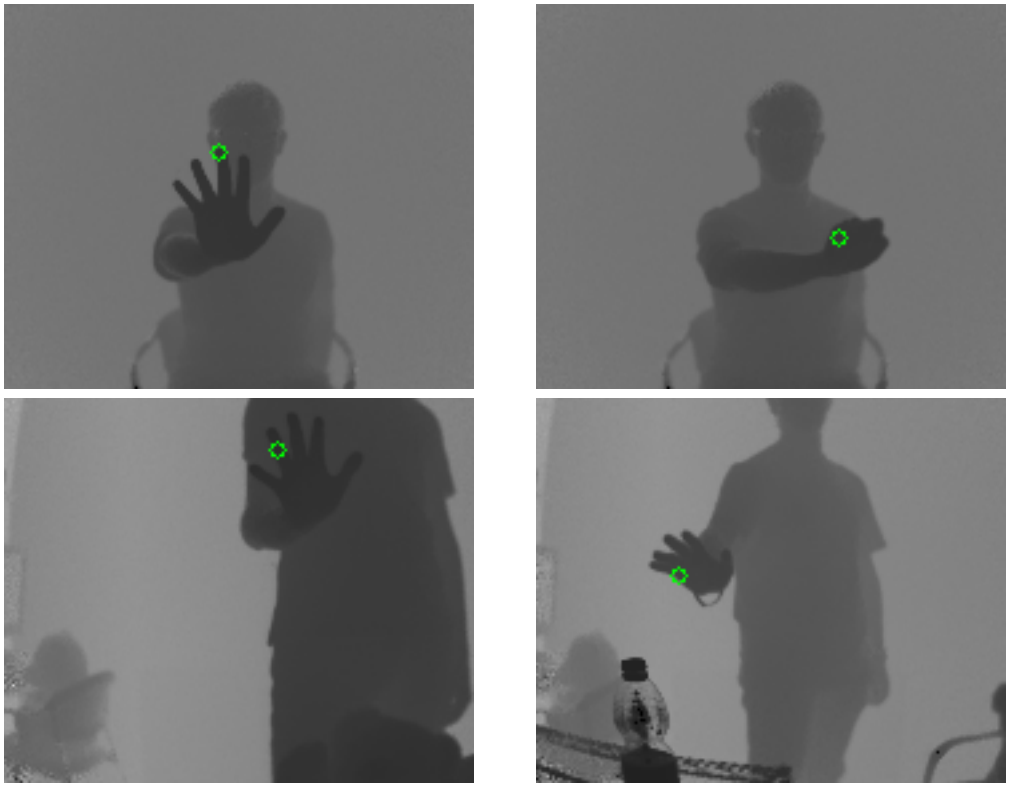


Figura 3.8: Le immagini mostrano il riconoscimento del punto a minima profondità (pallino verde) il cui corrispondente pixel nell'immagine a colori appartiene con buona probabilità alla pelle.

per come sono inseriti i punti, tendeva ad espandere i pixel prima lungo un'unica direzione. Il ciclo for che porta all'espansione della maschera termina quando la coda viene svuotata del tutto, il che equivale ad aver analizzato tutti i punti che soddisfano i vincoli imposti e aver scartato tutti quei vicini che non li soddisfano.

	0.485	0.495	0.49	
	0.51	0.50	0.492	
	0.505	0.5013	0.507	

(a)

	NO	0.495	0.49	
	NO	0.50	0.492	
	0.505	NO	0.507	**0.567**
		0.508	**0.515**	**0.499**

(b)

Figura 3.9: Esempio di espansione di un punto ai suoi vicini. La figura 3.9a mostra come, a partire dal punto corrente analizzato dal ciclo (in rosso), vengano inclusi (in senso orario e a partire dal vicino in basso a destra) solo quei vicini (in grassetto) i cui valori rientrano nel range di profondità fissato (in giallo). In figura 3.9b, invece, i punti già analizzati (in grigio) non vengono considerati e viene analizzato il primo vicino in senso orario (in verde in figura 3.9b) inserito nella coda all'iterazione precedente. Anche di questo vengono analizzati i vicini, purché non già visitati in precedenza.

Algorithm 3.2 *Algoritmo di espansione dei punti della maschera di profondità.*

while (depthPoints non è vuota) do
  for each $p_i \in depthPoints$
    for each q_i *vicino di* p_i
      Sia m_i *il corrispondente punto di* q_i *nella maschera finale.*
      if $(m_i = 1)$ *then*
        Continua perché il punto q_i *è già stato analizzato.*
      else
        Aggiungi il punto q_i *alla coda se il suo valore di profondità soddisfa le soglie e non appartiene a un edge e poni* $m_i \leftarrow 1$.
    end for
  end for
Ogni numIterations iterazioni controlla che depthPoints.size() sia cresciuta del 30%, altrimenti ferma.
end while

Oltre alla terminazione naturale dovuta allo svuotamento della coda, si è scelto di bloccare l'espansione di un generico punto anche in altri tre casi:

- avendo come informazione aggiuntiva quella sugli edge della depth map (grazie all'algoritmo di Canny), se il punto corrente appartiene ad un bordo di una figura non viene inserito nella coda, così da non analizzarne i vicini;
- oltre alle soglie sulla profondità, si verifica anche che un generico punto abbia un valore non troppo distante dal punto di minima profondità, per evitare di espandersi in zone troppo lontane;
- periodicamente, dopo un certo numero fisso di iterazioni (memorizzato nella variabile **numIterations**), viene controllata la dimensione, in termini di numero di punti aggiunti alla maschera, della coda depthPoints: se rispetto all'ultimo valore analizzato (**lastSize**) la coda non è cresciuta di almeno il 30%, l'espansione viene bloccata.

Questi ultimi tre accorgimenti non solo permettono di evitare che l'espansione continui oltre i confini della mano, ma anche che venga presa una porzione eccessiva del braccio nel caso in cui questo sia piuttosto in linea (verosimilmente, ad ogni iterazione i punti da espandere sul braccio sono in numero inferiore rispetto a quelli presenti sul palmo della mano). I valori utilizzati per bloccare l'espansione in tali casi, pur non essendo dinamici, permettono un corretto isolamento della mano rispetto al resto del corpo e della scena; inoltre, essi consentono di rilassare i vincoli sulla profondità degli oggetti, pur mantenendo l'espansione dei punti entro certi limiti. Tuttavia, inevitabilmente, in alcuni casi, la maschera conterrà comunque piccole porzioni del braccio.

3.2.5 Visualizzazione sull'immagine a colori e tracciamento dei contorni

La maschera di profondità ottenuta nel passo precedente costituisce un insieme di punti dei quali è possibile, utilizzando le corrispondenze sensore-camera, trovare i pixel relativi nell'immagine a colori ripresa dalla videocamera. Tuttavia, la risoluzione dell'immagine

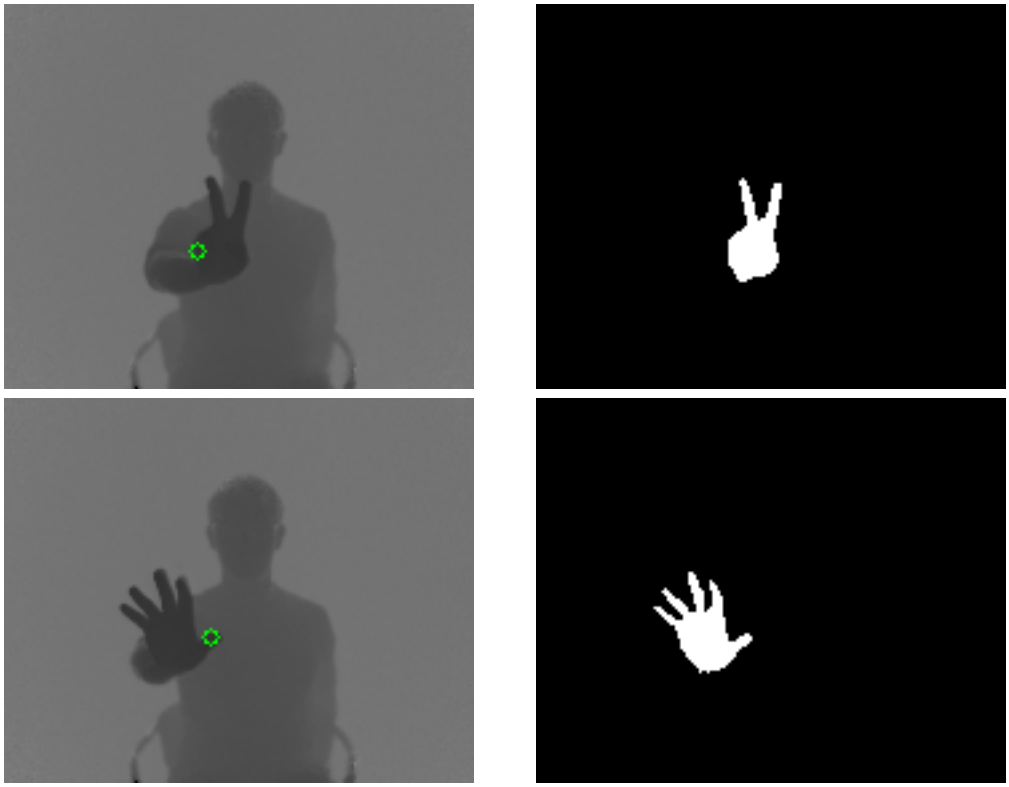


Figura 3.10: Risultati della procedura di espansione per la maschera di profondità.

acquisita dal sensore di profondità risulta di gran lunga inferiore rispetto a quella della camera, perciò i punti della maschera di profondità costituirebbero solo una parte di quelli realmente giacenti sulla mano nell'immagine a colori.

Sfruttando la possibilità di tracciare dei punti su quest'ultima (mediante la funzione *Circle* di OpenCv), si vuole quindi creare una nuvola di punti che approssimi al meglio la forma della mano; viene, dunque, creata un'immagine temporanea di dimensioni pari a quelle dell'immagine della camera, la quale raffigura la sola mano, e questa viene passata all'algoritmo di Canny che ne ricava i contorni. Infine, scandendo tale immagine contenente

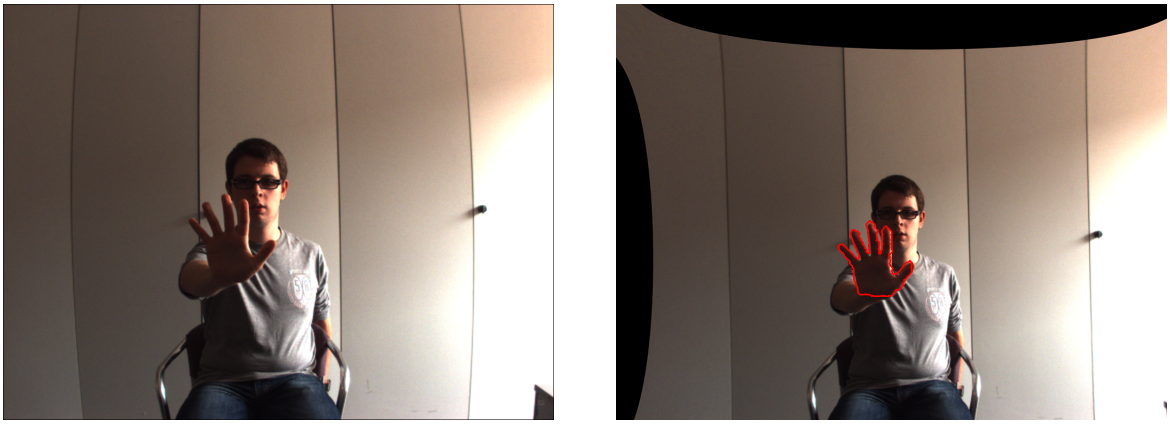


Figura 3.11: Esempio di risultato finale che evidenzia i contorni della mano. L'immagine finale è rettificata e anti-distorta.

i contorni, vengono riportati sull'immagine finale solo quei pixel che Canny ha isolato, in modo da ottenere nella posizione precisa tutti i pixel che definiscono la silhouette della mano (figura 3.11).

Una soluzione più apprezzabile per tracciare in maniera più precisa i contorni consisterebbe in un'interpolazione dei punti della maschera di profondità una volta proiettati sull'immagine a colori; tale soluzione risulta, però, più complessa e non è stata riportata nel progetto, anche se potrebbe essere implementata in futuro.

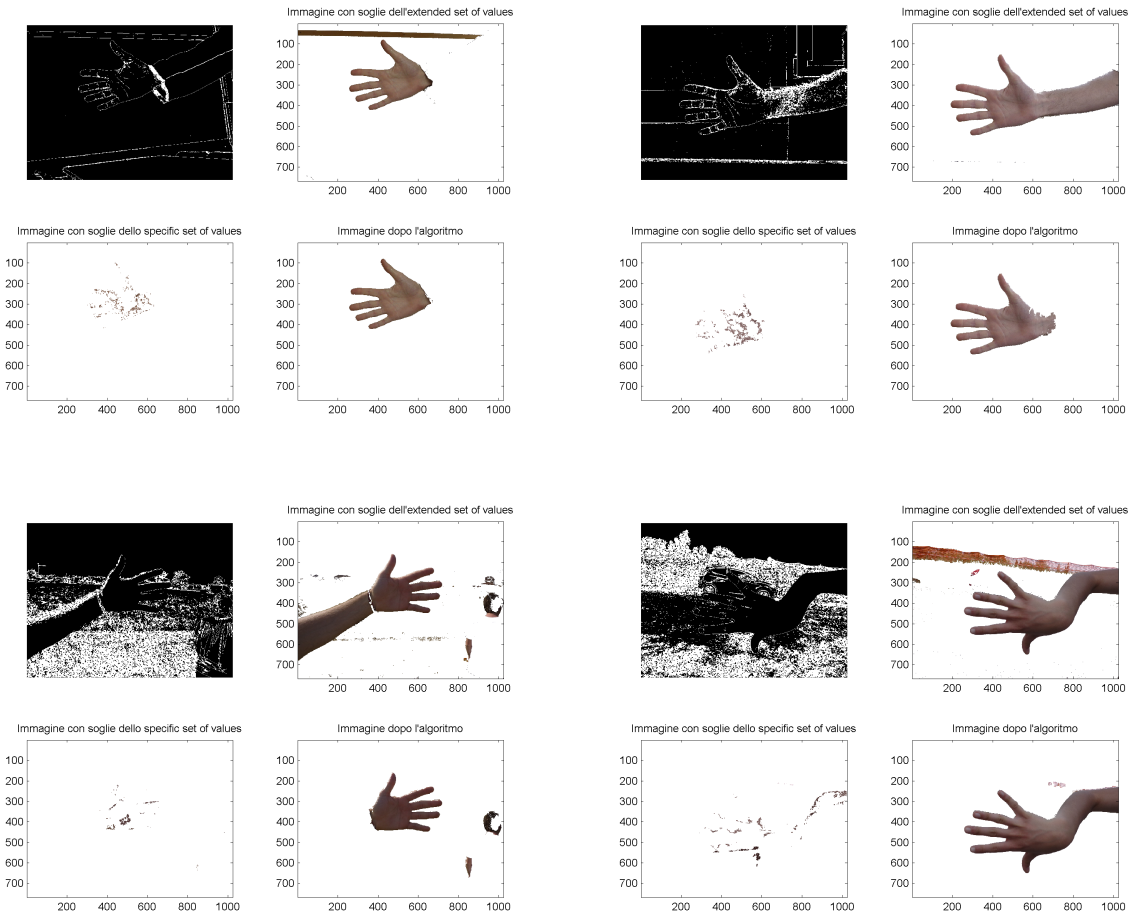
Capitolo 4

Risultati finali e analisi computazionale

In questo capitolo vengono riportati i risultati ottenuti dall'applicazione delle diverse soluzioni proposte, i confronti tra quelli relativi ad uno stesso stream di dati al quale sono stati applicati tutti gli algoritmi realizzati e l'analisi computazionale della soluzione finale proposta.

4.1 Filtraggio tramite Skin Detector

L'algoritmo di skin detection è stato testato su 125 immagini diverse catturate con una fotocamera e aventi risoluzione 1024x768.



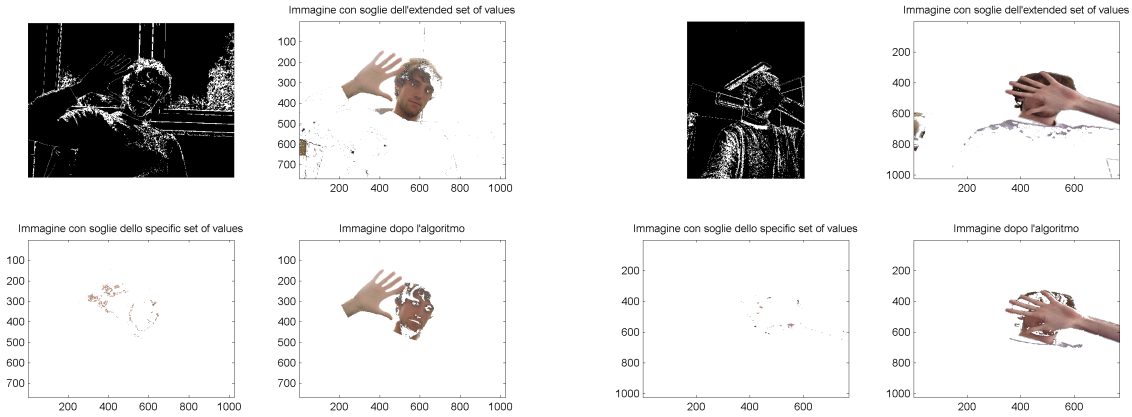
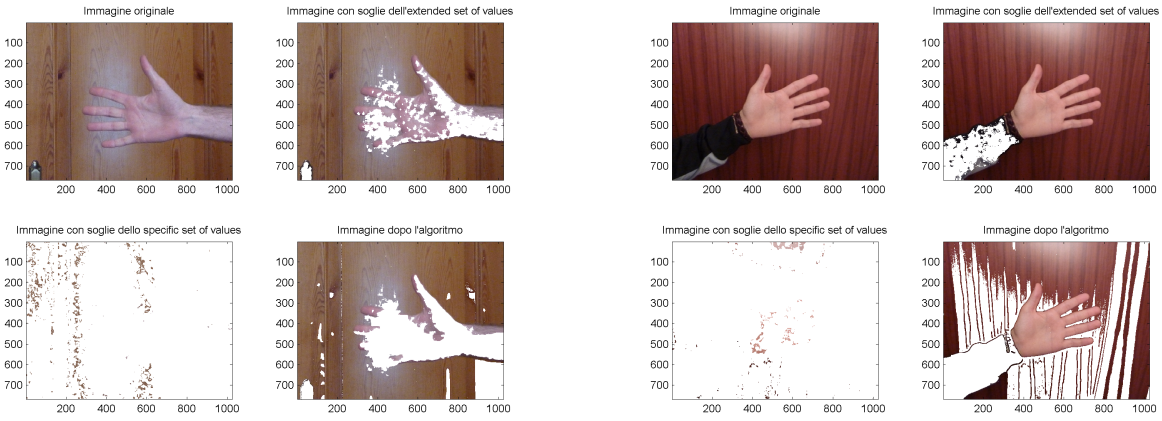


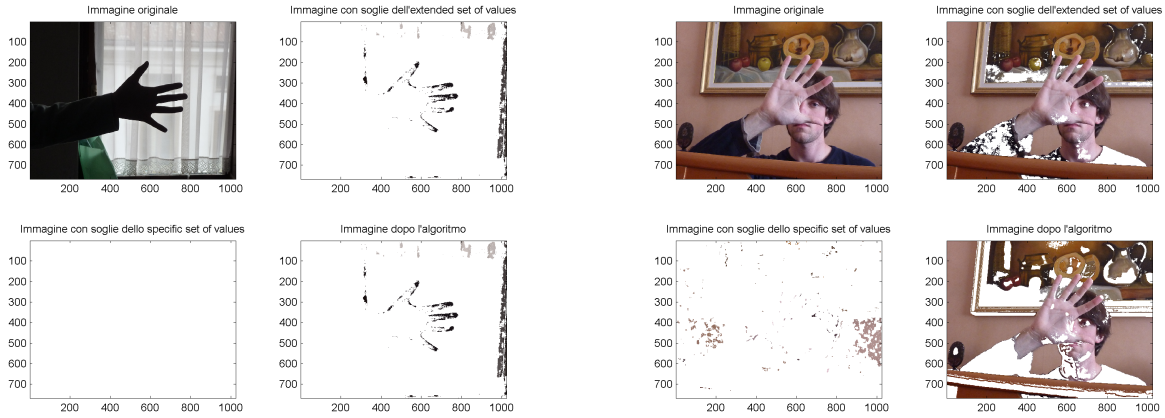
Figura 4.1: Esempi di applicazione dell'algoritmo.

Le soglie ricavate dalla fase di apprendimento tramite le immagini di training sono risultate verisimili nella maggior parte dei casi, fatta eccezione per situazioni piuttosto estreme quali posizionamento della mano di fronte ad oggetti con colori simili a quello della pelle (in particolare il legno, come mostrano le figure 4.2a e 4.2b) o situazioni in cui questa è molto scura a causa dell'ombra eccessiva. Comunque, tali casi sono particolari e per gli obbiettivi prefissati possono considerarsi trascurabili.



(a) (b)

Figura 4.2: Alcuni casi in cui l'algoritmo di Skin Dectection non riesce ad isolare a dovere i pixel della pelle perché non distinguibile dal legno.



(a) (b)

Figura 4.3: Altri casi in cui l'algoritmo non funziona a dovere: la figura 4.3a mostra il caso in cui l'ombra eccessiva non permette un corretto riconoscimento, mentre la figura 4.3b evidenzia le difficoltà dell'algoritmo in presenza di uno sfondo (muro) di colore non molto dissimile dalla pelle.

Dal punto di vista delle prestazioni dell'algoritmo, la sua realizzazione in Matlab ha evidenziato un tempo di esecuzione abbastanza alto (circa 15 secondi impiegati per l'elaborazione di ciascuna immagine), fatto che lo rende difficile da usare in caso di applicazioni real-time. Tuttavia, la sua versione in C++ si serve solo di alcuni aspetti dell'intero algoritmo, per cui i tempi sono notevolmente ridotti.

4.2 Filtraggio basato sulla profondità

Di seguito vengono riportati alcuni esempi di frame filtrati dall'algoritmo in Matlab che si basa interamente sulla profondità delle figure. Come detto, la risoluzione delle immagini è di 176x144.

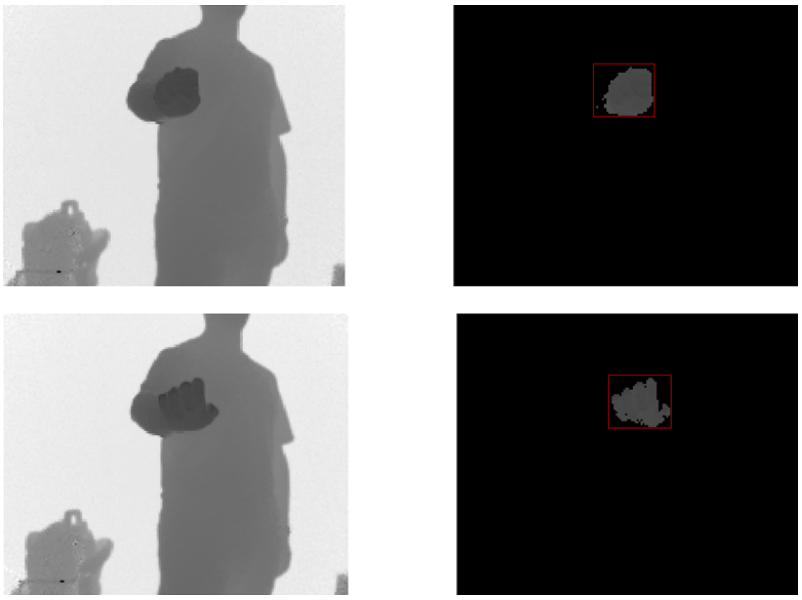


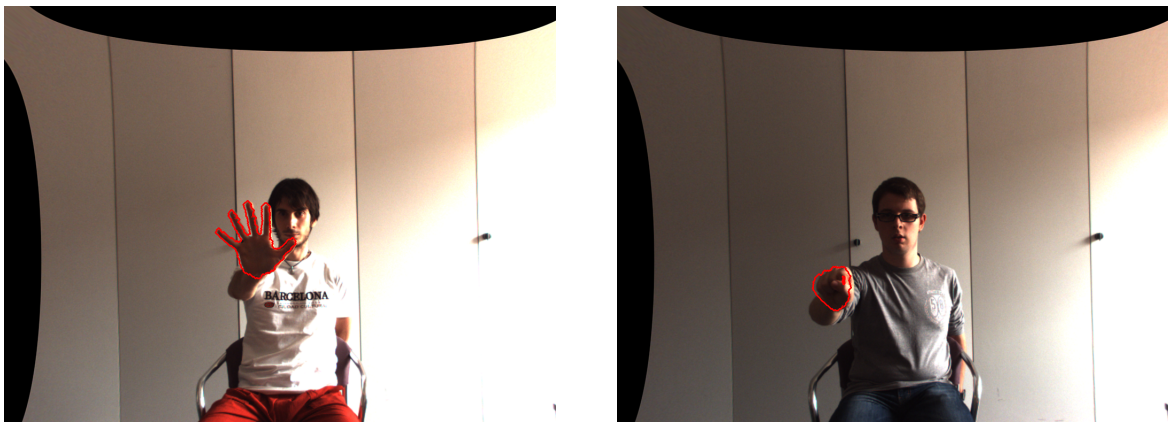
Figura 4.4: Le figure mostrano i risultati dell'algoritmo realizzato; a sinistra sono riportati frame di partenza e a destra il risultato della creazione della maschera di profondità.

Pur non essendo un algoritmo elaborato, la sua realizzazione ha permesso di familiarizzare con la strumentazione presente in laboratorio e, in particolare, capire come lavorare con il sensore TOF di profondità, indispensabile per l'ultimo passo del progetto. L'isolamento della mano in questa fase di progetto impone la forte limitazione di non avere oggetti di fronte alla mano, in qualunque posizione essi siano. In questi casi, infatti, l'algoritmo non funzionerebbe ed inevitabilmente verrebbe riconosciuto l'oggetto più vicino al sensore e che non rappresenta la mano.

4.3 Filtraggio tramite depthColorFilter

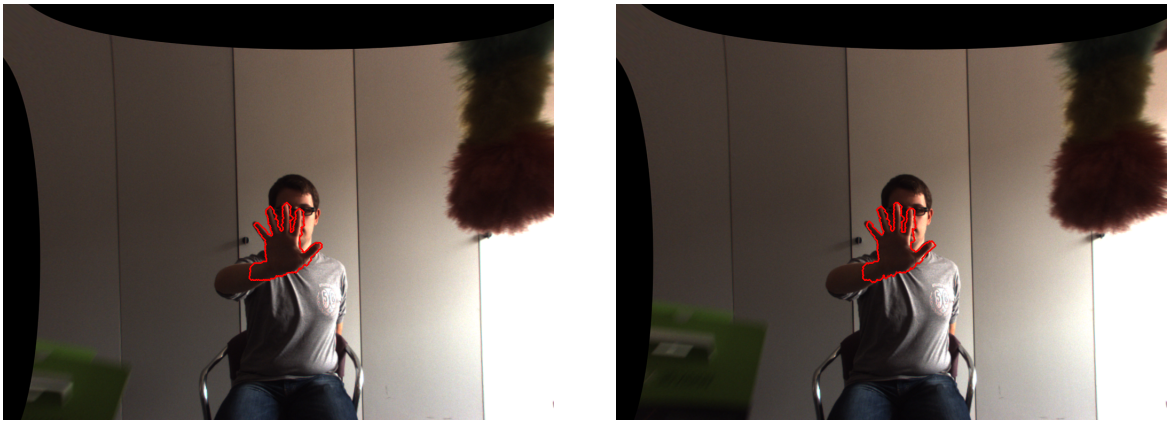
L'algoritmo finale sviluppato è stato testato su diverse serie di frame che ritraggono soggetti in condizioni ambientali diverse. Nella maggior parte dei casi, le immagini ritraggono una persona che compie movimenti con la propria mano eseguendo anche gesti diversi tra loro; si è poi voluto verificare che l'individuazione della mano avvennisse con successo anche in casi più estremi, che prevedono la presenza di oggetti nello spazio tra la mano e la sorgente di acquisizione, al fine di testare il corretto impiego congiunto delle informazioni sul colore e sulla profondità.

Il metodo depthColorFilter è stato applicato su stream di dati composti di 100 frame ciascuno; quello che si evince dai risultati è una corretta individuazione della mano nella maggior parte dei casi, nonostante non sempre l'illuminazione permetta una distinzione precisa di colore tra le figure riprese.

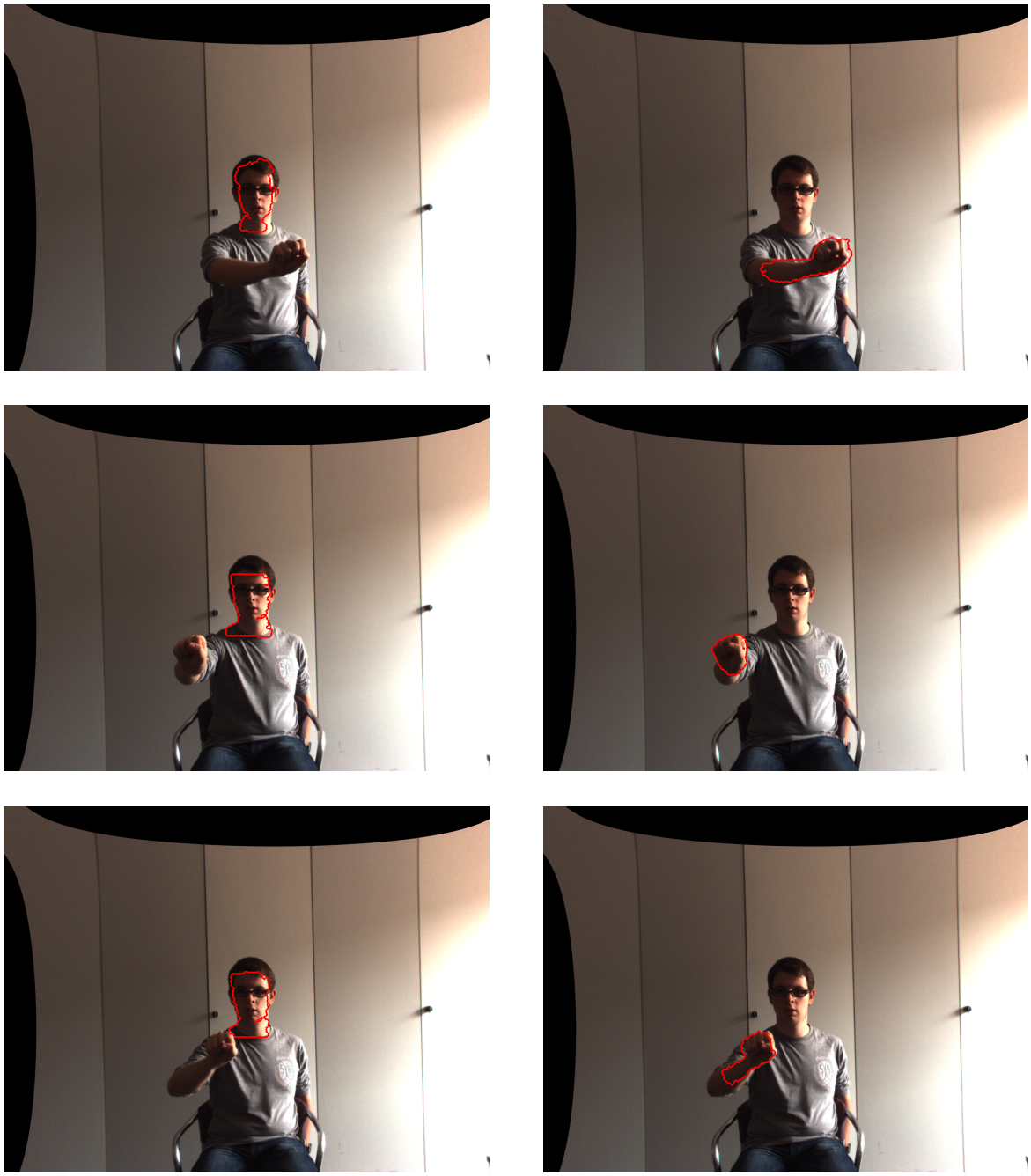


Tuttavia, l'espansione dei punti della maschera di profondità (illustrata dall'algoritmo 3.2) spesso porta ad includere porzioni piuttosto consistenti del braccio, in quanto non sempre i valori scelti per le soglie rappresentano un giusto compresso tra tutte le immagini riprese. Questo accade soprattutto quando il soggetto ha il polso in linea con la mano che si sposta, portando inevitabilmente il punto a minima profondità rilevato a giacere sul braccio o vicino ad esso.

Un altro aspetto positivo dell'algoritmo è che esso si comporta ragionevolmente bene anche nei casi in cui sono frapposti degli oggetti tra la mano e la sorgente di acquisizione. In questo caso viene in aiuto la procedura di calibrazione del colore (spiegata nel paragrafo 3.2.2.2), che permette di escludere tutto ciò che possiede un colore diverso da quello che si evince dal primo frame usato per la calibrazione. La maschera sul colore è modellata sulla mano e, per tale motivo, tutto ciò che è in primo piano viene escluso dall'algoritmo.



La calibrazione del colore, implementata nella seconda versione del metodo colorFilter, ha permesso anche di isolare la mano in casi in cui la prima versione, che si serve di soglie fisse (come spiegato nel paragrafo 3.2.2.1) fallisce a causa della scarsa densità di pixel in corrispondenza della mano. Le seguenti figure mostrano, a sinistra, l'immagine risultante dall'algoritmo che si serve della prima versione del metodo colorFilter, mentre a destra quella che risulta usando la seconda versione:

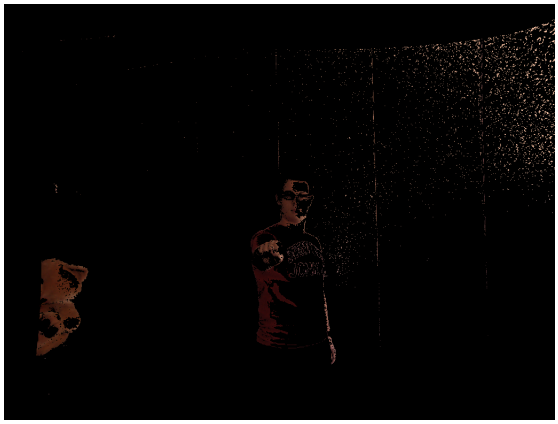


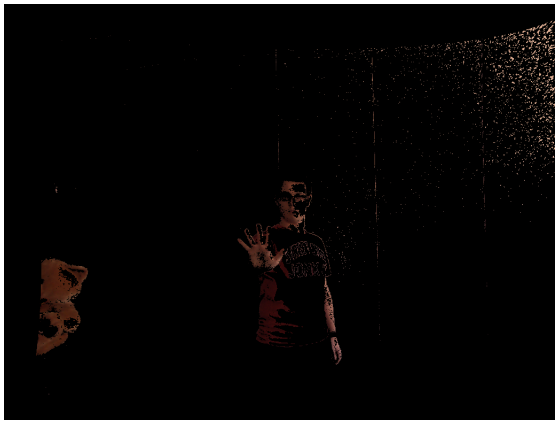
4.4 Confronto tra gli algoritmi proposti

Come spiegato ampiamente nel paragrafo 3.2, il metodo depthColorFilter si serve di due metodi ausiliari (colorFilter e depthFilter) per l'individuazione finale della mano. In questo

paragrafo si vogliono mettere a confronto i risultati di questi tre algoritmi per sottolineare come il loro uso congiunto sia di gran lunga più efficace del loro utilizzo separato. Per farlo, ci si è serviti di uno stream di 30 frame sincronizzati del quale consideriamo i campioni seguenti (già anti-distorti):


Applicando il solo metodo colorFilter, si ottiene una maschere a valori 0 e 1 basata sul colore (eventualmente calibrata in base alla scena) che se moltiplicata per l'immagine originale dà luogo ai seguenti risultati:



Come si può notare, la maschera esclude un notevole numero di pixel (in nero); tuttavia, alcuni pixel dello sfondo vengono erroneamente catturati, così come alcuni oggetti distanti dalla mano.

Applicando il solo metodo depthFilter, invece, si ottiene una maschera basata sulla profondità che contiene solo la mano e la sua applicazione, tramite le corrispondenze tra sensore TOF e camera, porta ai seguenti risultati:

I punti sull'immagine a colori corrispondenti a quelli della maschera di profondità comprendono, correttamente, la mano, ma, come si nota, la soglia di profondità non è sufficientemente selettiva e porta ad includere anche porzioni notevoli del braccio e persino del busto o delle gambe. Inoltre, l'eventuale presenza di oggetti in primo piano porterebbe il metodo a catturare, inevitabilmente, questi ultimi, in quanto incapace di distinguerli da oggetti più in profondità.

Infine, l'uso congiunto delle due informazioni permette di ottenere i risultati anticipati, andando a rilevare correttamente la posizione della mano e a tracciarne il contorno, come si nota dalle figure seguenti:

Questo testimonia ancora una volta che l'uso coordinato delle due informazioni può dar luogo a risultati di gran lunga migliori rispetto a quelli che si avrebbero sfruttandole in maniera separata.

4.5 Tempi e performance dell'algoritmo finale

Dal punto di vista dei tempi di elaborazione, i risultati ottenuti con il metodo depthColorFilter sono soddisfacenti: l'algoritmo proposto processa i singoli frame impiegando tempi relativamente brevi, anche se ancora non molto adatti ad un suo futuro impiego real-time.

Al fine di analizzare al meglio le prestazioni del metodo, sono state svolte diverse misure coinvolgenti stream diversi di dati; inoltre, si è voluto distinguere il caso in cui sia coinvolta la prima versione del metodo colorFilter e quello in cui sia coinvolta la seconda, che prevede una calibrazione sul colore. I principali tempi impiegati dall'algoritmo sono riassunti nella tabella seguente:

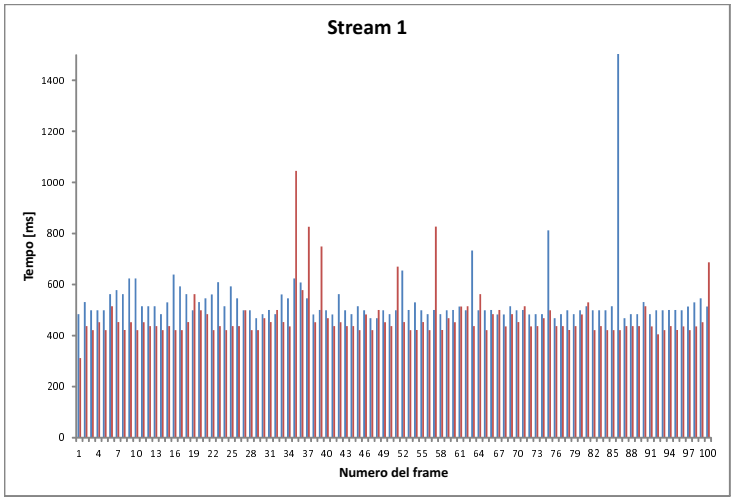
Indice prova	Numero di frame	Calibrazione colore	Tempo impiegato [ms]
1	100	No	91174
2	100	Sì	77142
3	100	No	76659
4	100	Sì	91682
5	100	No	76878
6	100	Sì	68110
7	100	No	63976

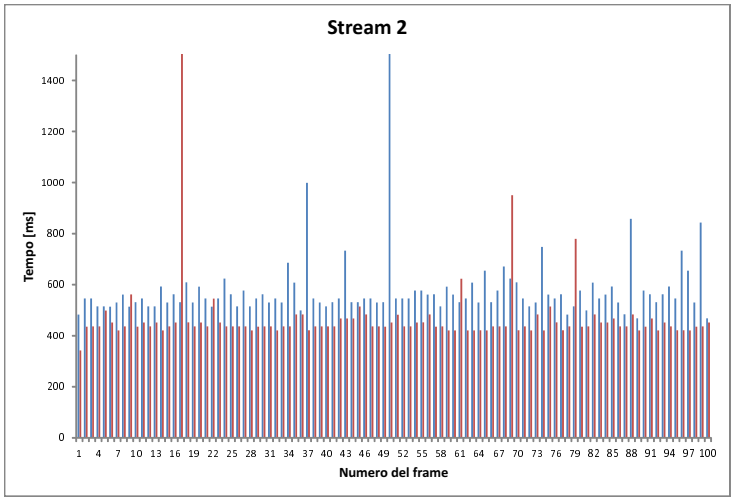
Tabella 4.1: La tabella riporta i tempi medi impiegati dall'algoritmo per processare una serie di stream di 100 frame ciascuno.

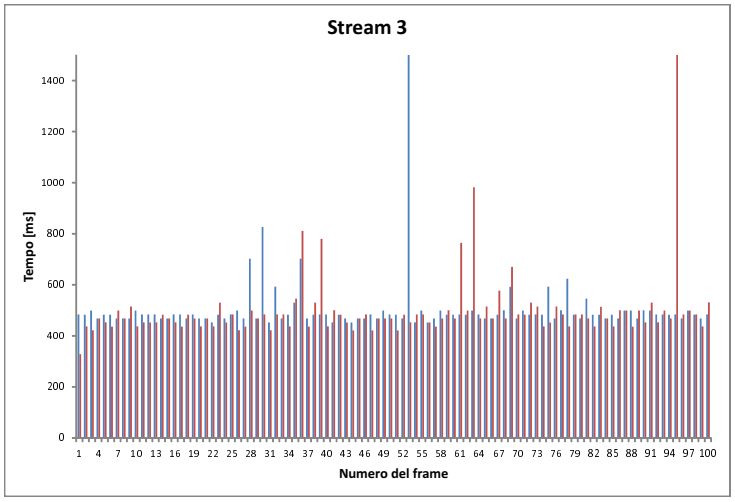
Come si può notare, l'algoritmo impiega, in media, circa 75s per elaborare 100 immagini, restituendo in uscita i frame risultanti e le maschere di profondità create.

Per ciascun frame sono stati analizzati la durata di ciascuna operazione, al fine di capire quale di queste richiedesse il maggior tempo; ciò che si evince dall'analisi è che la creazione della maschera sul colore, che prevede una scansione completa dell'immagine di partenza, è

quella che impone il maggior costo. Ovviamente, più la risoluzione delle immagini è alta, più tempo l'algoritmo impiegherà per processarle. Di seguito sono riportati dei grafici che mostrano l'andamento dei tempi impiegati dall'algoritmo per ciascun frame elaborato; i dati in blu si riferiscono all'applicazione della prima versione del metodo colorFilter, quelli in rosso all'applicazione della seconda versione (con calibrazione del colore).

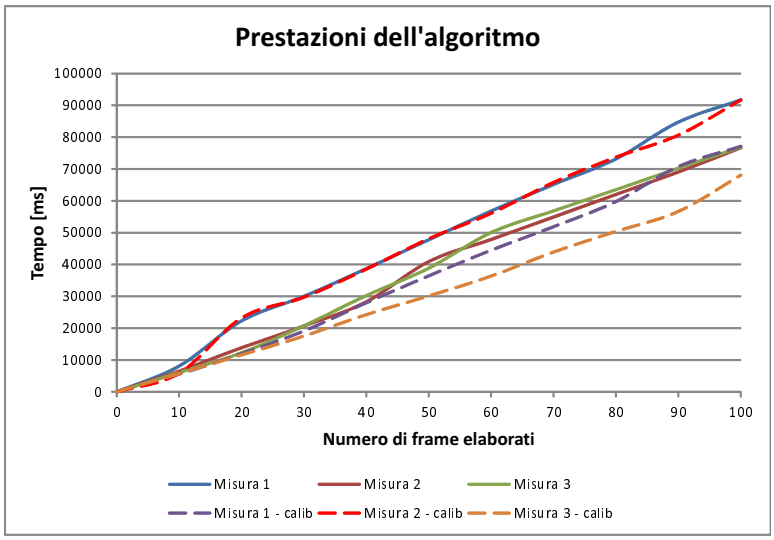






I grafici mostrano un andamento piuttosto uniforme dell'algoritmo, il quale necessita di circa 500 ms per elaborare ciascun frame che gli viene passato. Quello che si nota è che utilizzando una maschera creata sulla base di una calibrazione sul colore, i tempi sono leggermente inferiori, anche se non di molto. Il metodo colorFilter richiede circa 2/3 del tempo complessivo impiegato dall'algoritmo, mentre le operazioni di calibrazione degli strumenti di acquisizione e di creazione delle maschere di profondità risultano piuttosto irrisori (si parla di circa 30-40 ms per ciascun frame).

L'ebalorazione delle immagini è, dunque, fortemente dipendente dalla risoluzione dei frame acquisiti. Le prestazioni complessive dell'algoritmo sono riassunte nel grafico seguente, che ne analizza l'andamento in relazione ad un numero crescente di frame elaborati:



Le linee tratteggiate rappresentano l'andamento in presenza della calibrazione sul colore. Quello che si nota è che le prestazioni della soluzione realizzata sono lineari nel numero di frame elaborati e ciascuno di essi in media richiede meno di 1 s per essere processato. In alcuni casi, la calibrazione sul colore viene in aiuto in quanto permette di evitare operazione ulteriori di filtraggio e di Edge Detection, risparmiando tempo per il resto dell'elaborazione. In definitiva, lo sfruttamento della seconda versione del metodo colorFilter garantisce una maggiore efficienza dell'algoritmo.

Va sottolineato che l'algoritmo spende del tempo anche per creare le immagini risultanti, che vengono scritte all'interno di una cartella separata; inoltre, i processi che compongono l'elaborazione vengono eseguiti in maniera seriale, nonostante alcuni di essi possano essere facilmente resi paralleli. Tuttavia, le prestazioni sono complessivamente buone, pur ribadendo che non sono ottimali e non ancora del tutto adatte ad un possibile impiego real-time dell'algoritmo.

Capitolo 5

Possibili spunti e sviluppi futuri

Il progetto realizzato, oltre a soddisfare gli obbiettivi prefissati, offre notevoli spunti per lavori futuri per i quali esso può rappresentare il punto di partenza. Il codice sviluppato può essere ulteriormente ampliato e raffinato per permettere nuove funzionalità e raggiungere scopi più complessi legati all'ambito della Gesture Recognition.

In questo capitolo si vogliono riportare alcune interessanti idee che potrebbero essere sviluppate in futuro, nonché una serie di miglioramenti implementabili all'interno del codice che possano rendere l'algoritmo di riconoscimento della posizione della mano più preciso ed efficiente.

5.1 Possibili estensioni e raffinamenti dell'algoritmo sviluppato

La soluzione proposta in questo progetto adempie a dovere al suo scopo e ha permesso di soddisfare gli obbiettivi che ci si era prefissati. Tuttavia, essa può essere ulteriormente ampliata dal punto di vista del codice con nuove funzionalità e caratteristiche tese a migliorarne l'efficienza e la precisione, agevolando anche un suo potenziale utilizzo in real-time. Di seguito vengono riportate alcune modifiche attuabili al codice atte a soddisfare tali requisiti.

- E' stato appurato che il maggior tempo richiesto dall'algoritmo riguarda la creazione della maschera sul colore, che presuppone una scansione di tutti i pixel (nel caso peggiore) che compongono l'immagine in ingresso. Questo processo necessita di una accelerazione se si vuole ridurre il costo computazionale complessivo, magari utilizzando una sola porzione dell'immagine come maschera, piuttosto che considerare zone che non interessano la mano.

- Come accennato alla fine del paragrafo 4.5, le singole operazioni che compongono l'elaborazione sono svolte in modo seriale; infatti, non è stata implementata alcuna parallelizzazione dei processi, che, invece, darebbe un contributo significativo alle prestazioni. Ad esempio, impiegando dei thread paralleli si potrebbero parallelizzare i processi di creazione delle maschere e quelli di calibrazione degli strumenti di acquisizione, oppure rendere parallele le stesse istruzioni che compongono i singoli filtraggi delle immagini.

- L'algoritmo che riguarda l'espansione della maschera di profondità svolge correttamente il suo compito, ma, come detto, porta ad includere eccessive porzioni di pelle che

non appartengono alla mano, bensì al braccio. Un ulteriore raffinamento potrebbe consistere nell'ideare una soluzione, basata su principi matematici, che permetta di bloccare l'espansione quando sono superati determinati vincoli, mantenendo l'efficienza, allo stesso tempo, relativamente alta.

- Attualmente, lo sfruttamento delle informazioni su colore e profondità avviene in maniera unidirezionale: a partire dai punti dell'immagine del sensore TOF, si ricavano quelli corrispondenti dell'immagine a colori. Ogni processo che coinvolge questa corrispondenza procede solo in questa direzione, mentre si vorrebbe che il lavoro di filtraggio si servisse di entrambe tali informazioni in maniera bidirezionale. E' bene osservare che tale aspetto è, tuttavia, non sempre attuabile, in quanto non tutti i punti presenti nell'immagine a colori sono individuabili in quella del sensore (a causa delle diverse risoluzioni dei dispositivi), perciò sarebbe necessario stabilire un metodo efficiente per individuare quei punti del sensore che si avvicinino a quelli scanditi nell'immagine della videocamera, magari servendosi di una mesh di punti. Tuttavia, si dovrebbe considerare il problema delle occlusioni che, inevitabilmente, si presenterebbero e richiederebbero attenzione.

5.2 Morfologia e modello 3D

Parallelamente a questa attività di progetto, è stato compiuto uno studio preliminare, da parte del collega dott. Fabio Dominio, riguardo un modello tridimensionale che approssimi al meglio la morfologia della mano. L'algoritmo descritto in questa relazione si serve di due sole informazioni (colore e profondità), senza però poter sfruttare le caratteristiche morfologiche degli oggetti ripresi; questo può rappresentare un grosso ostacolo in situazioni ambientali più complesse, superabile avendo a disposizione un qualcosa che permetta di distinguere le forme degli oggetti.

Il progetto fin qui illustrato potrà servire, in futuro, come ausilio ad un più complesso algoritmo che, oltre ad individuare precisamente la posizione della mano, permetterà anche di definirne un modello 3D che ben approssimi quello reale. Lo studio che permette tale risultato rappresenta una maggiore sfida, ma potrebbe portare a risultati ancora migliori di quelli fino a questo momento ottenuti.

5.3 Individuazione di entrambe le mani

Il cuore del metodo depthColorFilter è rappresentato dall'individuazione del punto a minima profondità che appartiene anche alla mano. Questo permette non solo di avere un riferimento per la posizione, ma anche di isolare la mano dal resto della scena creandone una maschera di profondità utile a tracciarne i contorni.

Ciononostante, l'algoritmo potrebbe non limitarsi a considerare un unico punto a minima profondità, ma molteplici punti che permettano di distinguere anche una mano dall'altra. Questo permetterebbe di tollerare più di una mano nella scena e ciascuna di esse verrebbe correttamente rilevata. Supponendo che la scena racchiuda più mani in posizioni diverse, una possibile idea potrebbe essere quella di partire sempre dal punto a minima profondità che appartiene alla pelle (e questo, ovviamente, apparterrà alla mano più vicina alla sorgente di acquisizione), tracciare una maschera di profondità con la procedura di espansione illustrata

in questa tesi e spostarsi più in profondità una volta che tale maschera è completa, in modo da passare alla creazione della maschera di una mano più distante. Impostando delle opportune soglie di valori, sarebbe dunque possibile analizzare tutte le mani che, lungo l'asse z del centro ottico, sono presenti nella scena.

Un ulteriore problema potrebbe essere rappresentato da casi in cui le due mani appaiono allineate tra loro, per cui la profondità non aiuterebbe a distinguerle con facilità. In tal caso, il calcolo di un singolo punto a profondità minima potrebbe essere sostituito da un vettore di punti con valori di profondità non troppo diversi tra loro, che permetta una espansione parallela delle maschere di profondità su entrambe le mani.

5.4 Riconoscimento di gesti, motion tracking e Microsoft Kinect©

Il riconoscimento della posizione della mano, con conseguente creazione di una maschera di profondità che la include, fornisce un ottimo punto di inizio per la vera e propria Gesture Recognition: a partire da una serie di immagini risultanti, si può integrare nell'algoritmo una funzione che riconosca, servendosi anche della morfologia, la posizione delle dita e del palmo, in modo da classificare e riconoscere gesti diversi. Da questo potrebbe nascere un filone di intelligenza artificiale che permetterebbe all'algoritmo di comandare determinate operazioni al computer in base ai gesti della mano ricevuti come input.

Un altro traguardo che lo staff del laboratorio vorrebbe raggiungere è quello di riuscire a tracciare il movimento della mano a partire da uno stream di dati in ingresso proveniente dalla sorgente di acquisizione scelta. Oltre ad essere un'applicazione real-time, questa attività rappresenterebbe un ulteriore passo in avanti e comporterebbe uno sviluppo importante dell'algoritmo che non solo dovrà tracciare il movimento, ma, a limite, anche provare a predirlo.

Un altro spunto interessante potrebbe essere quello di sostituire l'attuale sistema di acquisizione (sensore e camere) con il Kinect© della Microsoft: a differenza del sistema trinoculare, questo dispositivo permetterebbe di avere già subito le informazioni congiunte su colore e profondità (pur con una precisione di gran lunga inferiore) e semplificare in alcuni casi le corrispondenze tra punti. L'accessorio permetterebbe di lavorare con delle librerie apposite in C++ e fornire notevoli sviluppi legati allo scheletro delle figure e al tracciamento dei movimenti, nonché risulta una tecnologia più accessibile, in termini di costo, ad una futura utenza che si servirà dell'applicazione sviluppata.

Capitolo 6

Conclusioni

Questa relazione ha voluto mettere in evidenza gli importanti risultati raggiunti nell'ambito della Gesture Recognition. L'obbiettivo iniziale di individuare precisamente la posizione della mano all'interno di una scena è stato ampiamente soddisfatto e costituisce un ottimo punto di partenza per scopi e sviluppi futuri in questo ambito.

Ogni singolo passo del progetto svolto è stato adeguatamente testato e raffinato con l'aiuto di tutti i membri del gruppo di lavoro, che hanno contribuito all'analisi anche di casi limite e piuttosto rari. L'impedimento maggiore al lavoro svolto è risieduto principalmente nei tool software impiegati, spesso non adeguati o troppo limitati nelle loro funzionalità per permettere un'analisi più approfondità. I frame acquisiti ed elaborati sono in numero adeguato, ma dei test più accurati ne richiederebbero una quantità ancora maggiore e diversificata, al fine di analizzare una più ampia casistica di situazioni ambientali.

Complessivamente, i risultati ottenuti sono decisamente buoni ed il software rilasciato opera con una buona efficienza e permette di ricostruire adeguatamente i contorni della mano nella maggior parte delle posizioni all'interno della scena inquadrata. L'intero gruppo di lavoro dell'LTTM, me compreso, si ritiene soddisfatto del lavoro svolto e auspica che esso possa ritenersi un buon punto di inizio per qualcosa di ancora più grande e innovativo.

Appendice A

Procedura di acquisizione tramite trigger hardware

I frame utilizzati per testare l'algoritmo necessitano di una sincronizzazione accurata tra i dispositivi coinvolti nell'acquisizione; perciò, è necessario l'utilizzo della scheda di sincronizzazione in comune tra camere e sensore TOF e l'abilitazione dell'opzione di *trigger hardware* nel tool di acquisizione usato. A tal proposito, con la collaborazione del collega dott. Enrico Cappelletto e dell'ing. Carlo Dal Mutto, sono state integrate una serie di modifiche atte a fare in modo che i dispositivi acquisiscano siano perfettamente sincronizzati tra loro.

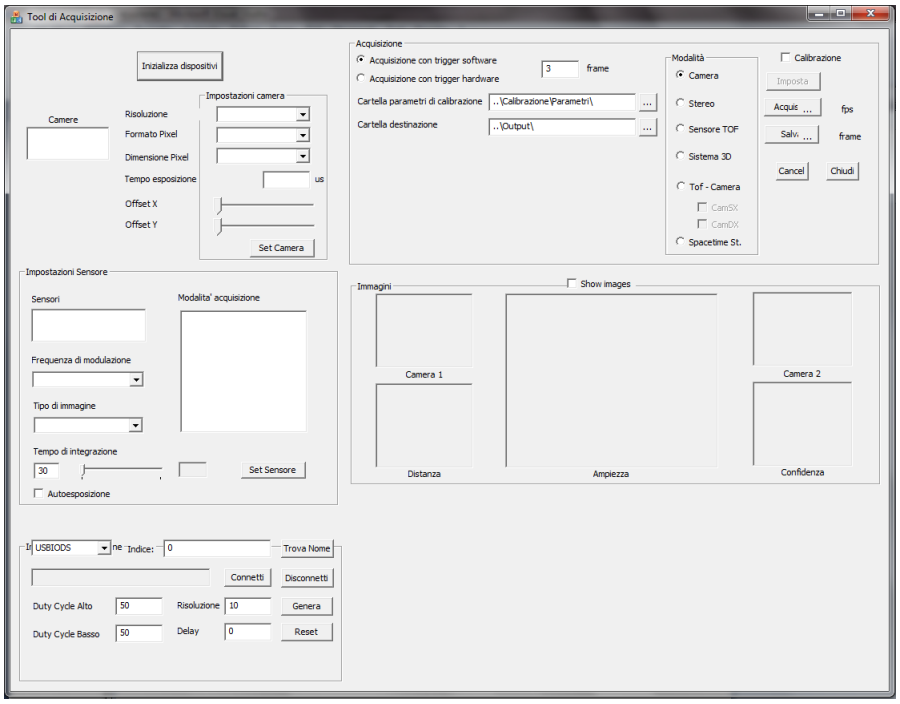


Figura A.1: Immagine del tool software di acquisizione presente in laboratorio.

Dopo aver inizializzato i dispositivi ed aver impostato la modalità $AM\_HW\_TRIGGER$ per il sensore di profondità, è necessario connettere la scheda di sincronizzazione e impostarne i parametri per generare il segnale ad onda quadra che segnali a camere e sensore il momento in cui acquisire simultaneamente i singoli frame.

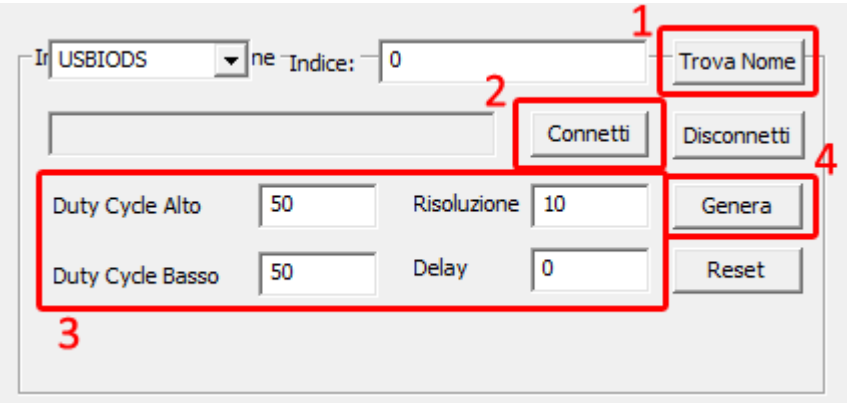


Figura A.2: Modulo del tool software relativo alla sceda di sincronizzazione.

Con riferimento alla figura A.2, la procedura di impostazione della scheda Delcom di sincronizzazione procede secondo i passi seguenti:

1. Mantenendo l'impostazione del menu a tendina su *USBIODS*, cliccare sul pulsante **Trova Nome** al fine di individuare la scheda, che, ovviamente, deve essere connessa al computer adibito alle acquisizioni tramite connessione USB.

2. Una volta individuata la scheda, cliccare sul pulsante **Connetti** per impostare la connessione alla scheda.

3. Impostare i parametri desiderati relativi al segnale generato dalla scheda:

   (a) *Duty Cycle alto* (DC_A) e *basso* (DC_B), che rappresentano, rispettivamente, le percentuali di periodo massimo dell'onda quadra (di 1 s) occupate dal fronte d'onda alto e da quello basso; questi parametri influenzano la frequenza degli impulsi del segnale della scheda secondo la seguente relazione:

   $$k = \frac{100}{(DC_A + DC_B)/100}$$

   dove k rappresenta il numero di frame al secondo che si vogliono ottenere per l'acquisizione. Ad esempio, se sono richiesti 5 fps, DC_A e DC_B vanno entrambi impostati a 10 (cioè il 10% di 1 s) oppure a due valori la cui somma dia 20. Il valore 100 al numeratore indica il 100% del periodo totale di 1 s.

   (b) *Delay*, il cui valore inserito, in ms, rappresenta il tempo che la scheda deve aspettare prima di generare il primo fronte d'onda (testato fino a 2 s).

   (c) *Risoluzione*, parametro che, per questi scopi, non è necessario cambiare.

   Il modo con cui la scheda è impostata determinerà le modalità di acquisizione delle camere e del sensore TOF.

4. Impostati i parametri, cliccare sul pulsante **Genera**. Il codice associato a questo bottone è stato modificato per far sì che resetti automaticamente la scheda e salvi i parametri inseriti dall'utente in apposite variabili statiche. Esso va distinto dal pulsante **Reset** che si limita a resettare le impostazioni di default della scheda.

Una volta terminata l'impostazione della scheda di sincronizzazione, si spunta la voce *Acquisizione con trigger hardware* e si imposta il sistema 3D. Una volta iniziata l'acquisizione, i parametri della scheda salvati in precedenza vengono inviati al componente, che genera un'onda quadra per far partire i dispositivi in maniera sincronizzata.

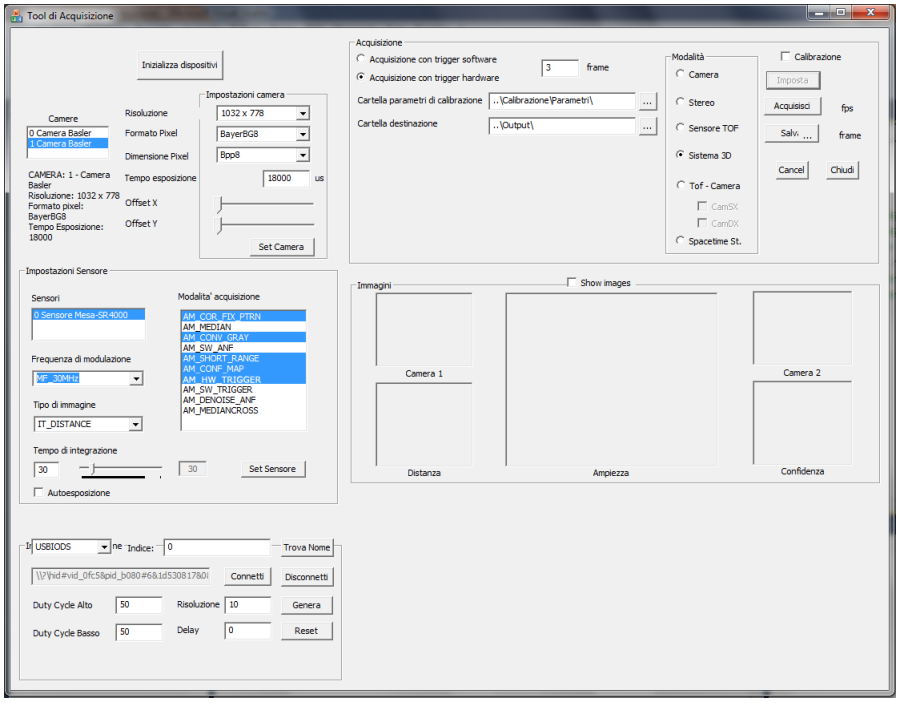


Figura A.3: Esempio di configurazione del tool software per acquisizioni tramite trigger hardware.

Bibliografia

[1] **Andrea Fusiello**: *Visione Computazionale - Appunti delle lezioni*, 2008

[2] **P. Kakumanu, S. Makrogiannis, N. Bourbakis**: *A survey of skin-color modeling and detection methods*, 2006

[3] **Paul Viola, Michael Jones**: *Robust Real-time Object Detection*, 2001

[4] **William T. Freeman, Michael Roth**: *Orientation Hitograms for Hand Gesture Recognition*, 1994

[5] **Richard Szeliski**: *Computer Vision: Algorithms and Applications*, 2010

[6] **Gary Bradski, Adrian Kaehler**: *Learning OpenCv*, 2008

[7] **LTTM:** *Rappresentazione delle immagini - Campionamento, quantizzazione e interpolazione*, Corso di Elaborazione delle immagini e dati 3D, 2008

Ringraziamenti

Un doveroso ringraziamento va all'intero team di sviluppo dell'LTTM, in particolare al relatore G. M. Cortelazzo, al correlatore P. Zanuttigh e all'ing. Carlo Dal Mutto che hanno contribuito in maniera decisa allo sviluppo di questo progetto con le loro idee innovative e la loro influenza positiva. Ringrazio i miei colleghi universitari per il supporto fornitomi, la mia famiglia e i miei più cari amici per il loro sostegno e il loro affetto che mi hanno permesso di impegnarmi con tutto me stesso per il raggiungimento di questo fondamentale traguardo.