UNIVERSITÀ DEGLI STUDI DI PADOVA

—

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

—

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA DELL'AUTOMAZIONE

SVILUPPO E VALIDAZIONE DI UNA PIATTAFORMA SOFTWARE INTEGRATA PER SISTEMI DI VISIONE INDUSTRIALE

RELATORE: PROF. GIULIO ROSATI
DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

LAUREANDO: MARCO BACCEGA

ANNO ACCADEMICO 2018-2019

“ Dio ci scampi! Siamo nelle mani degli ingegneri... ”

Dr. Ian Malcolm, dal film Jurassic Park

Indice

Sommario

Il lavoro svolto in questa tesi si svolge nell' ambito della visione indutriale. Negli ultimi anni si è assistito sempre più ad un utilizzo di questi sistemi per garantire il funzionamento di isole robotizzate e controllo qualità. Questa tesi fornirà gli strumenti per l'utilizzo della libreria professionale di Halcon all' interno di Matlab. Verranno quindi fornite le classi e metodi C# e Mex il cui funzionamento verrà poi sperimentato e messo in relazione con i metodi classici. Una volta forniti gli strumenti verrà approfondito l'utilizzo di Halcon rispetto tipici problemi di visione industriale, come ad esempio il problema della guida robot.

Introduzione

Un sistema di visione artificiale è costituito dalla integrazione di componenti ottiche, elettroniche e meccaniche che permettono di acquisire, registrare ed elaborare immagini. Il risultato dell'elaborazione è il riconoscimento di determinate caratteristiche dell'immagine per varie finalità di controllo, misura, classificazione, selezione, ecc.

I sistemi di visione artificiale, in ambito industriale, sono sistemi automatici per controllo qualità e misure dimensionali, che tipicamente vengono installati su una linea di produzione. L'uso di questa tecnologia può offrire numerosi vantaggi, come costanza, ripetibilità, garanzia di conformità e qualità sulla produzione. Consente inoltre la oggettivazione dei controlli qualitativi con la possibilità di predeterminare il rapporto qualità/scarti ottimale, la riduzione dei costi di produzione e un aumento del livello tecnologico del prodotto tramite l'automazione dei processi di fabbricazione.

Per poter essere utilizzati in tali contesti sono necessari strumenti hardware e software in grado di interfacciarsi con gli apparati industriali presenti sul mercato, inoltre le performance dell'intero processo in termini di qualità e velocità devono rispettare determinate specifiche.

Nella presente Tesi si è sviluppata l'integrazione delle librerie professionali di *Halcon* prodotto da MvTec con *Matlab* della MathWorks. Il primo è un software usato in ambito industriale per risolvere task legati alla visione e interfacciarsi con gli standard di acquisizione industriali. La semplicità d'uso del suo ambiante di sviluppo (HDevelop) unita al vasto numero di prodotti nativamente supportati permettendo di ridurre i tempi di sviluppo. Questo unito all'efficienza degli algoritmi (particolarmente performanti), forniscono uno strumento ideale per ottenere

ottimi risultati dal punto di vista della qualità e della velocità di elaborazione. *Matlab* invece è un software usato da ingegneri e scienziati in tutto il mondo. Fornisce un linguaggio di alto livello e un ambiente interattivo utilizzato nell'ambito della matematica e del calcolo numerico, dell'elaborazione di segnali e di immagini, dei sistemi di controllo e nella robotica.

L'integrazione dei due programmi porta notevoli vantaggi in ottica di gestire l'intero processo da *Matlab*, il tutto lasciando tutta la parte di visione al runtime di *Halcon* e sfruttandone i metodi. Questa scelta è suggerita dal fatto che *Matlab* può interfacciarsi a diversi linguaggi e in modo complementare *Halcon* nativamente possiede dei tool in grado di esportare le elaborazioni eseguite sfruttando i vari vantaggi di HDevelop.

Di seguito viene descritta la struttura in cui si suddivide la presente relazione.

Nel **Primo capitolo** vengono descritti gli obbiettivi del lavoro, introducendo quindi i sistemi di visione artificiale e le varie tecniche utilizzate in ambito industriale. Si vuole quindi brevemente introdurre tutti i componenti per la formazione dell'immagine (sensori, ottiche, sistemi di illuminazione, ecc), vengono illustrate le tecniche più comuni di pre-elaborazione e i metodi di matching di *Halcon* per risolvere i diversi problemi affrontati nella tesi.

Nel **Secondo capitolo** viene introdotto brevemente il funzionamento di *Halcon* e i vari modi per integrarlo con *Matlab*. In particolare vengono descritte le classi e i metodi implementati per raggiungere questo obbiettivo, nello specifico sono state sviluppate una classe C#, una libreria MEX, l'invio tramite Socket e il salvataggio da applicazione stand-alone. Viene infine descritto il tipo di applicazioni nelle quali conviene usare una determinata tecnica.

Nel **Quarto Capitolo** vengono descritte le prove effettuate nell'utilizzo di Halcon per risolvere alcuni dei problemi tipici in ambito industriale. Inizialmente vengono analizzati i tempi per l'invio a *Matlab* dei risultati con i diversi metodi implementati. Vengono poi illustrate e commentate le varie prove sperimentali come il riconoscimento di oggetti per un problema di guida robot, diverse elaborazioni su una batteria alettata, acquisizione tramite un nastro in movimento, infine riconoscimento oggetti 3D in particolare tramite l'utilizzo della telecamera di profondità del sensore *Kinect*.

Capitolo 1

Sistemi Visione

In questo capitolo verranno descritti gli obbiettivi del lavoro introducendo dei concetti che stanno alla base della visione industriale.
Inizialmente vengono brevemente illustrate alcune delle configurazioni più usate nel mondo della visione industriale, di conseguenza si discute dei sistemi 2D definendo le componenti hardware principali, in modo da ottenere una buona immagine. Poi vengono descritti i sistemi 3D, in particolare quelli del tipo TOF(time-of-flight) di cui fa parte il sensore Kinect.
Infine verranno descritte le più comuni tecniche di pre-elaborazione e di matching utilizzate da Halcon per risolvere diverse tipologie di problemi legate alla visione.

1.1 Obiettivi del Lavoro

In letteratura ed in commercio è possibile trovare diverse soluzioni e strumenti per risolvere problemi legati ai sistemi di visione, in genere sono sufficienti questi strumenti, ma se si vogliono ottenere determinate prestazioni in termini di precisione e tempo di sviluppo affidarsi a librerie professionali spesso è necessario. Questa tesi infatti vuole approfondire l'utilizzo di una di queste (*Halcon*) e verificarne attraverso test sperimentali le potenzialità. Sono quindi stati sviluppati gli strumenti in modo da interfacciare queste librerie con l'ambiente Matlab in modo da poter sfruttare appieno le potenzialità dei due programmi. Una volta forniti questi strumenti verranno testati in modo da verificare il possibile utilizzo in con-

testi in cui il tempo ciclo sia un aspetto cruciale. Durante le prove sperimentali verranno quindi affrontati diversi problemi tipici nel campo dell'automazione, ad esempio è stato realizzata la parte inerente alla visone per una applicazione di *pick-and-place* tramite l'ausilio di un manipolatore.

1.2 Hardware per sistemi 2D

Un sistema di visione artificiale integra una o più telecamere dotate di sistema di acquisizione ed elaborazione immagini integrato o esterno, un software interno e/o esterno alla telecamera ed un sistema di illuminazione. Un sistema di visione è in grado di misurare, riconoscere, identificare, selezionare, leggere codici e caratteri, guidare robot (guida robot). A tal fine trova larga applicazione nel controllo qualità dei prodotti, nella tracciabilità e nella loro movimentazione.

Il componente principale è la telecamera, questa è formata da due principali componenti: il sensore e l'ottica. La scelta di questi due elementi è fondamentale e verranno descritte le principali categorie con i rispettivi pregi e difetti. La funzione delle telecamere industriali è quella di catturare l'immagine proiettata sul sensore, attraverso una lente, per poterla trasferire attraverso un'interfaccia di comunicazione ad un PC e poterla quindi visualizzare, memorizzare o per effettuare su di essa delle misure o delle analisi. Le telecamere hanno avuto una rapida evoluzione negli ultimi anni, le più moderne telecamere sono provviste di sensori CCD (Charge Coupled Device) e CMOS (Complementary Metal Oxide Semiconductor).

Le telecamere utilizzate nella visione industriale e scientifica richiedono una serie di caratteristiche non comuni nelle classiche webcam e nemmeno nelle più sofisticate macchine fotografiche presenti sul mercato: è ad esempio necessario avere un completo controllo del tempo di esposizione, dei guadagni e della risposta colorimetrica della camera stessa. È necessario potere controllare esattamente l'istante in cui la camera dovrà acquisire un'immagine tramite un eventuale segnale elettrico, pilotare con la medesima precisione un illuminatore in modo che produca il flash luminoso sincronizzato con la fase di acquisizione dell'immagi-

ne da parte della camera. Deve essere inoltre possibile trasferire nel modo più efficiente l'immagine verso il computer di elaborazione.

1.2.1 Sensore

Il sensore è un insieme di elementi fotosensibili in grado di accumulare luce e di convertirla in un segnale elettrico. La disposizione di questi elementi fotosensibili in una matrice o lungo una singola linea definisce una prima distinzione tra telecamere matriciali e lineari. La sensibilità degli elementi fotosensibili al completo spettro visibile o a definite bande come ad esempio al rosso, verde e blu definiscono camere monocromatiche o a colori. I sensori delle telecamere moderne sono per lo più CMOS, ogni singolo pixel è composto di un materiale sensibile alla luce che converte i fotoni in carica elettrica. Un'elettronica di contorno si occupa di trasferire il segnale da ogni singolo pixel per formare l'immagine e trasferirla al computer tramite un'interfaccia. In ambiente industriale la risoluzione delle telecamere può arrivare fino a 80 Megapixel e il numero di immagini acquisite al secondo può spingersi fino ad alcune centinaia utilizzando l'intero sensore e fino ad alcune migliaia selezionando una regione (ROI) del sensore. Il collegamento Telecamera-Computer avviene attraverso standard di comunicazione e attualmente gli standard più diffusi sono basati su porte gigabit Ethernet, Firewire e USB3, telecamere particolarmente performanti richiedono tuttavia delle schede dedicate all'interno del PC dette *framegrabbers* per poter essere utilizzate. Le telecamere con sensore CMOS sono diventate molto popolari negli ultimi anni. La qualità dell'immagine è migliorata significativamente e questo tipo di tecnologia è essenziale in alcuni tipi di applicazioni come nei sistemi ad altissima velocità. I sensori CMOS utilizzano un substrato di materiale sensibile alla luce, ma a differenza dei sensori CCD, trasferiscono le cariche dal sensore all'elettronica interna della camera mediante un metodo di accesso casuale, invece di usare registri a scorrimento comunemente usati nei sensori CCD. Questo permette di ottenere telecamere in grado di acquisire un numero di immagini al secondo molto maggiore rispetto alle telecamere CCD. Tra le telecamere come precedentemente detto se ne distinguono due macrogruppi:

- **Telecamere Matriciali**: Il termine telecamera matriciale o ad area si riferisce al fatto che il sensore della telecamera copre un'area o che è formata da una matrice di N x M pixel. Le dimensioni di un sensore sono definite in pollici, tuttavia la dimensione effettiva del sensore non ha nulla a che fare con la dimensione della sua definizione ma si basa sul rapporto del primo CCD con tubi Vidicon. I formati più comuni attualmente sono 1/3 ", 1/2", e 2/3". Queste possono essere monocromatiche oppure a colori; nel primo caso le telecamere utilizzano un sensore basato su silicio e sensibile a tutto lo spettro della radiazione visibile da circa 400nm fino a 1000nm, mentre le seconde applicano dei filtri in grado di far passare solo un determinato sottoinsieme d'onda dello spettro visibile. In particolare si utlizzano i filtri *RGB* in grado quindi di far passare rispettivamente "red, green , blue" formando quindi un immagine sfruttando diversi canali e sovrapponendoli è possibile ottenere l'immagine a colori.

- **Telecamere Lineari**: La più semplice definizione di telecamera lineare è una telecamera composta da un sensore avente una sola linea, per poter acquisire una scena bidimensionale come con una classica telecamera matriciale la telecamera si deve muovere rispetto all'oggetto o l'oggetto si deve muovere rispetto alla telecamera. Con questa tecnologia si possono acquisire immagini ad altissima risoluzione. Se la risoluzione perpendicolare al movimento è infatti limitata dalla lunghezza del sensore, la risoluzione verticale non è limitata e dipende solamente dal numero di linee che si vogliono acquisire. A livello software si definisce infatti la dimensione verticale dell'immagine che consiste in un numero di linee definito dall'utente Le telecamere lineari trovano impiego in tutte le applicazioni ove il materiale da analizzare è un oggetto con rapporto lunghezza/larghezza particolarmente alto oppure un insieme di oggetti che si presentano di fronte alla telecamera con alta frequenza. Nella maggior parte delle applicazioni con telecamere lineari la velocità degli oggetti che passano di fronte alla telecamere varia, per tanto è necessario sincronizzare la velocità di acquisizione della telecamera con la velocità di movimento dell'oggetto. Questo si realizza inviando un segnale trigger esterno generato ad intervalli di spazio regolare in modo che le linee

si sincronizzino con il movimento. Questo si esegue generalmente mediante un encoder.

1.2.2 Ottica

Nella maggior parte delle applicazioni di visione si impiegano ottiche a lunghezza focale fissa mentre le ottiche con zoom (motorizzato o manuale) sono poco utilizzate in quanto le parti mobili rendono difficile, se non addirittura impossibile, ottenere misure ripetibili. Le ottiche a fuoco fisso sono inoltre ideali per le applicazioni di visione in quanto i pezzi vengono solitamente presentati alla telecamera in una posizione sostanzialmente invariabile e ad una distanza fissa dalla telecamera. La soluzione ottica ideale per una qualsiasi applicazione dipenderà da diversi elementi, quali:

- Area inquadrata (FOV).
- Distanza di lavoro (distanza dell'oggetto dalla telecamera).
- Profondità di campo (DOF).
- Distorsione dell'ottica

La maggior parte delle ottiche è provvista di una ghiera per la regolazione della messa a fuoco. Questa ghiera in realtà non modifica la distanza focale, bensì regola il 'back focus' (la distanza effettiva fra il sensore e il piano focale della lente), permettendo di mettere a fuoco oggetti a varie distanze come mostrato in figura 1.1. La lunghezza focale f di un'ottica è la misura (solitamente in mm) della distanza fra il centro ottico dell'obiettivo e il piano dell'immagine. La luce che parte da un oggetto all'infinito interseca l'asse ottico dell'obiettivo nel punto focale f. Nelle telecamere per sistemi di visione, come in qualsiasi telecamera, il sensore coincide con il piano dell'immagine (punto focale) dell'ottica.
La profondità di campo (o profondità di messa a fuoco) definisce il campo entro cui un oggetto può essere posizionato rispetto alla telecamera restando sempre nitido e a fuoco. Il valore è una funzione di molti parametri, principalmente le dimensioni dell'iride del diaframma. Tanto più chiuso è il diaframma, tanto maggiore sarà la profondità di campo come mostrato in fig.1.2. Teoricamente, una

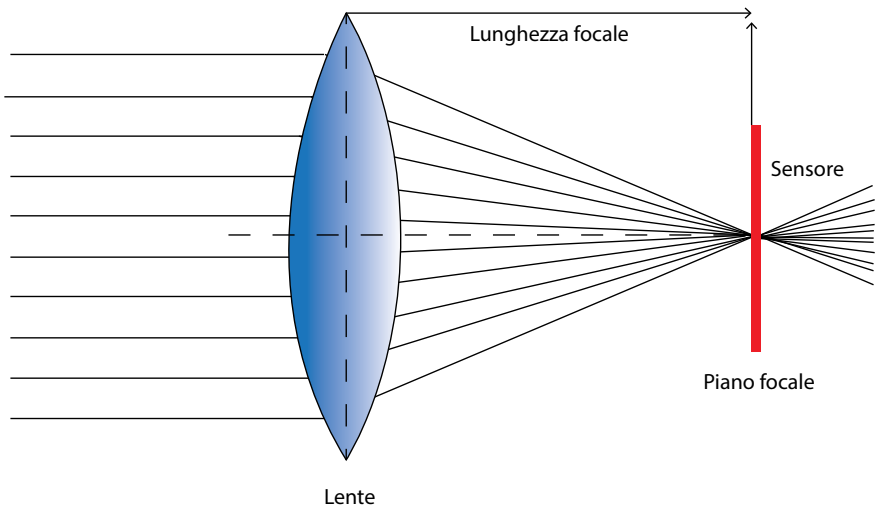

Figura 1.1: Schema geometria dell'ottica tra lente e sensore, in particolare è mostrata la lunghezza focale. Quest'ultima è modificabile tramite la ghiera presente nella maggior parte delle ottiche industriali.

telecamera con un'apertura infinitamente piccola ha una profondità di campo infinita, ma richiede una quantità di luce infinita per poter ottenere un immagine. Nella pratica, si deve sempre trovare il giusto compromesso fra luce e apertura del diaframma per ottenere una profondità di campo accettabile. Un'altra tecnica per far aumentare la luce che entra nel sensore mantenendo costante la profondità di campo è quella di aumentare il tempo di esposizione in modo quindi da aumentare la luce che entra nel sensore mantenendo il diaframma con la stessa dimensione.
Per quanto riguarda invece il problema della distorsione questa indica un'alterazione nella rappresentazione geometrica di un oggetto sul piano dell'immagine. Ad esempio, un rettangolo può assumere la forma di un "cuscino" o di un "barile". La distorsione dell'immagine può causare seri problemi nella visione industriale. Se infatti è necessario effettuare delle misure sull'oggetto, è fondamentale che l'immagine sia una riproduzione fedele dell'oggetto stesso. A volte è possibile realizzare una piccola correzione della distorsione attraverso algoritmi di compensazione che sono però costosi dal punto di vista computazionale. Pertanto è molto importante valutare le proprietà di distorsione di un'ottica prima di adottarla nel proprio sistema di visione. La distorsione dell'ottica è causata dalla rifrazione

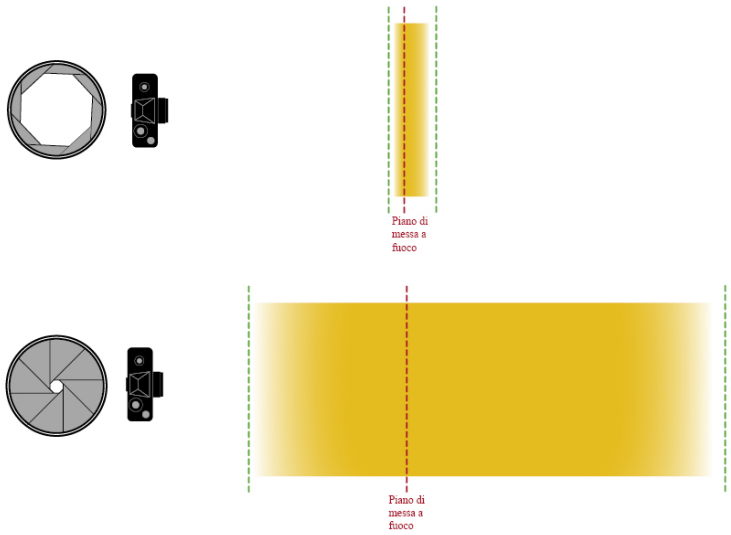

Figura 1.2: Rappresentazione della variazione del piano di messa a fuoco a seconda dell'apertura del diaframma. In particolare è possibile vedere come al restringimento del diaframma corrisponda un aumento del piano di messa a fuoco.

della luce nel vetro della lente. Poiché le lenti non sono composte da un vetro con spessore uniforme e le superfici non sono piatte, la rifrazione varia nelle diverse aree della lente, raggiungendo solitamente i livelli massimi lungo i bordi. L'effetto è particolarmente evidente negli obiettivi a grandangolo (con lunghezza focale corta), che possono presentare distorsioni periferiche notevoli. Nelle ottiche di alta qualità la distorsione viene limitata attraverso l'uso di lenti multi-stadio, anche se non esiste un'alternativa ai grandangoli per applicazioni di misura. Un metodo per diminuire questo fenomeno è quello di utilizzare un'ottica con una focale che induce un FOV stretto ma che rappresenta un oggetto lontano dalla camera, in questo modo l'effetto di distorsione diminuisce.

1.2.3 Sistemi di Illuminazione

Si può affermare che l'illuminazione sia la parte più critica di un sistema di visione e per questo motivo nelle applicazioni di machine vision la scelta del metodo di illuminazione non deve essere sottovalutata. L'illuminazione è una tecnica sofisticata e non esiste una scienza esatta per capire come e perché un certo tipo di illuminazione è adatto a una determinata applicazione. Telecamere e fotocamere

sono molto meno versatili rispetto all'occhio umano e le condizioni di illuminazione devono spesso essere ottimizzate affinché la telecamera possa rilevare oggetti che l'occhio umano vede in qualsiasi condizione. Questo vale soprattutto quando si ha a che fare con forme complesse o componenti riflettenti. La telecamera, come l'occhio umano, è in grado di vedere la luce riflessa dagli oggetti e di conseguenza l'illuminazione di un sistema di visione determina il modo in cui l'oggetto appare alla telecamera. La luce viene riflessa in maniera diversa da un oggetto metallico, da un'etichetta bianca piatta o da una scheda a circuito stampato e pertanto diverse tecniche di illuminazione sono in grado di evidenziare differenti particolari dell'oggetto analizzato. Un'illuminazione inadatta può spesso fare la differenza fra un sistema che funziona e uno che non funziona. Ottimizzando l'illuminazione si elimina spesso la necessità di filtrare l'immagine e si ottiene una soluzione complessivamente migliore. Nella visione artificiale sono utilizzate molte tecniche di illuminazione standard per enfatizzare determinati particolari di un oggetto, non bisogna però confondere le "tecniche" con il "tipo" di illuminazione.

Tipi di Illuminazione

I principali tipi di illuminazione comunemente utilizzati per i sistemi di visione sono: fibra ottica, fluorescente e LED. Ogni tipo ha i suoi vantaggi e svantaggi, cosicché ogni applicazione è adatta a un tipo di illuminazione specifico.

- **Illuminazione a fibra ottica:** Le fibre ottiche sono filamenti di materiali vetrosi o polimerici, realizzati in modo da poter condurre al loro interno la luce (propagazione guidata) sino ad un "adattatore" luminoso posizionato nelle vicinanze dell'oggetto.

- **LED:** In elettronica il LED o "diodo ad emissione luminosa" è un dispositivo optoelettronico che sfrutta le proprietà ottiche di alcuni materiali semiconduttori di produrre fotoni attraverso un fenomeno di emissione spontanea. Illuminatori a LED rappresentano attualmente la soluzione più adatta praticamente per la maggior parte delle applicazioni di visione industriale poiché normalmente questi dispositivi producono luce ad alta intensità a un costo relativamente basso e vantano una durata operativa molto lunga.

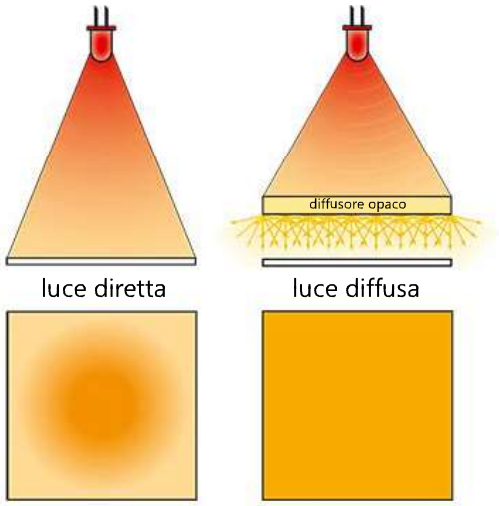

Figura 1.3: Differenza del sistema di illuminazione tra luce diretta a sinistra e luce diffusa a destra.

Un altro vantaggio è rappresentato dal fatto che gli illuminatori a LED richiedono solo l'alimentazione in corrente continua.

- **Illuminazione fluorescente:** I tubi fluorescenti sono un sistema di illuminazione comune per uso domestico caratterizzato da buona efficienza energetica che trova impiego limitato nella visione industriale a causa della disponibilità limitata di forme e dimensioni. Inoltre spesso lo "sfarfallio" prodotto a causa dell'alimentazione in alternata può produrre un disturbo notevole.

Tecniche di Illuminazione

Prima di illustrare le diverse tecniche di illuminazione, è importante spiegare i termini "diretto" e "diffuso". I termini si riferiscono alla qualità di luce prodotta. La luce diretta ha un percorso ininterrotto dalla sorgente (ad esempio i LED) all'obiettivo. La luce diffusa passa attraverso un diffusore opaco che ammorbidisce e disperde la luce, rendendola meno intensa ma più uniforme come mostrato in figura 1.3.

Ci sono diverse tecniche di illuminazione le più comuni sono:

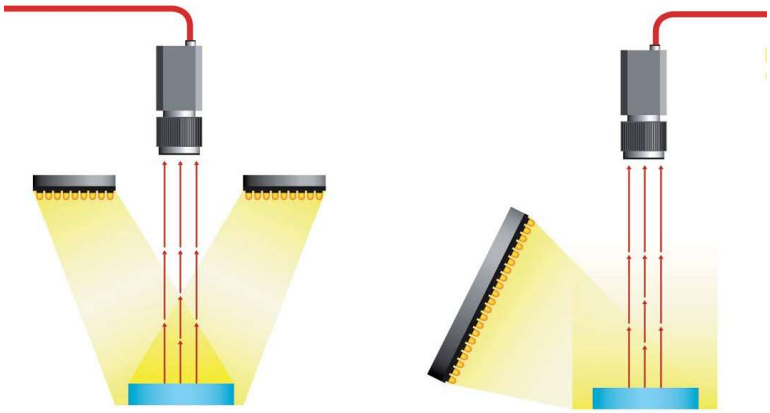

Figura 1.4: A sinistra è possibile osservare l'illuminazione di tipo bright-field in cui la luce arriva in maniera ortogonale al piano. A destra invece è presente il sistema dark-field in cui la luce arriva in maniera parallela.

- **Illuminazione bright-field:** È la tecnica più comune per l'illuminazione di oggetti omogenei non riflettenti. Il termine bright-field si riferisce alla posizione di montaggio dell'illuminatore. Se la telecamera viene posizionata in modo assiale rispetto ad uno specchio piano(figura 1.4.a), si definisce "bright-field" l'area in cui la luce riflessa si trova all'interno del campo visivo (FOV) della telecamera.

- **Illuminazione dark-field:** Questa tecnica è utilizzata per evidenziare alcuni particolari che sono visibili solo illuminando l' oggetto con una la luce laterale rispetto alla posizione della telecamera come mostrato in figura 1.4.b. Efficace per evidenziare difetti, graffi o difetti superficiali, è particolarmente utile per l'ispezione di oggetti riflettenti. Solitamente realizzata con un sistema di illuminazione ad anello con diametro molto superiore all'oggetto da inquadrare con una bassa angolazione, questa tecnica è totalmente dipendente dalla posizione di montaggio dell'illuminatore. Le due immagini in figura 1.5 illustrano il principio dell'illuminazione dark-field quando vengono rilevati graffi o difetti di superficie su un oggetto piatto come una moneta. Come mostra la prima immagine, su una superficie piatta si ha un angolo di incidenza basso e la maggior parte della luce viene riflessa al di fuori del campo di vista della telecamera. La seconda immagine

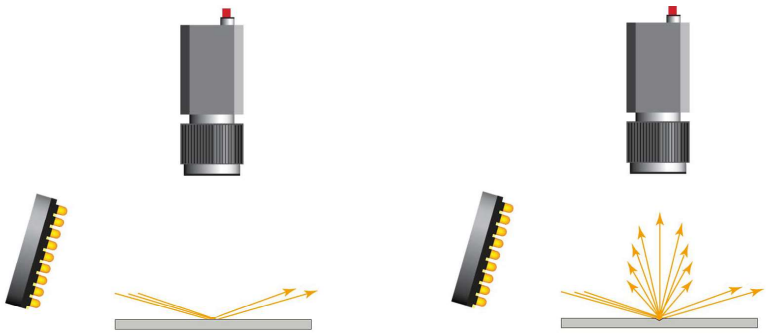

Figura 1.5: Comportamento della luce con illuminazione dark-field. A sinistra un oggetto piatto senza deformità riflette poca luce, a destra invece un oggetto con presente una deformità riflette in maniera consistente la luce verso la telecamera.

mostra come la luce venga normalmente dispersa e solo in presenza di un difetto sulla superficie, venga riflessa verso la telecamera evidenziando in tal modo il graffio o la deformazione superficiale.

- **Illuminazione back-light** Questa tecnica prevede che l'oggetto sia posizionato tra la sorgente di illuminazione e la telecamera. In questo modo è possibile riconoscere la sagoma dell'oggetto. Questa tecnica è utilizzata per ottenere misurazioni estremamente precise ed è utilizzata in applicazioni con materiali traslucidi o trasparenti al fine di individuare difetti o caratteristiche. Nella pratica, questo metodo viene utilizzato normalmente quando è necessario rilevare "fori passanti", quando l'oggetto è complessivamente opaco con variazioni di chiaroscuro nelle aree di interesse oppure quando è necessario delinearne i bordi.

1.2.4 Standard di Acquisizione

Un'interfaccia standard codifica come una telecamera è collegata ad un pc, fornendo un modello definito che permette l'utilizzo della machine vision in maniera più efficace. Un sistema di visione è composto da vari componenti fra cui telecamere, schede di acquisizione e librerie di elaborazione; spesso i componenti arrivano da produttori differenti e quindi uno standard può aiutare nell'interfacciare questi componenti fra loro in maniera efficiente. Quando le telecamere utilizzate

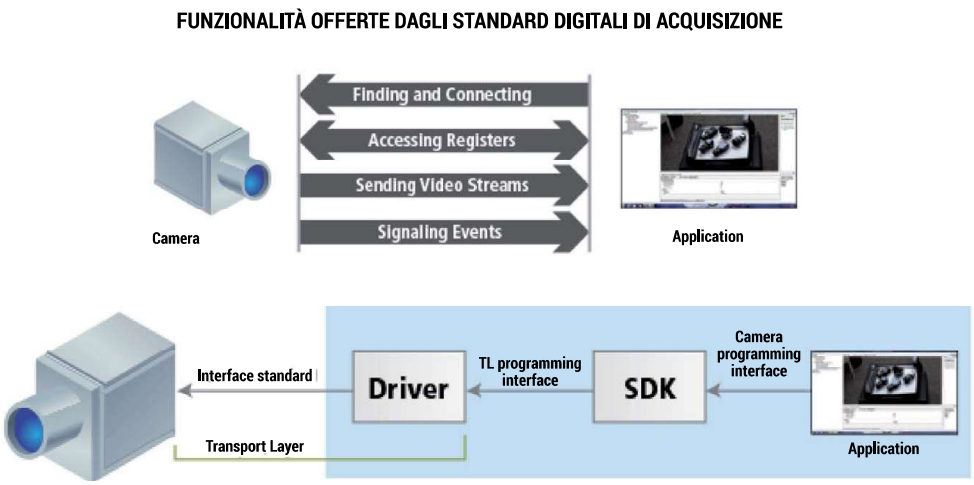

Figura 1.6: Schema di connessione degli standard di comunicazione PC-camera

erano di tipo analogico, ogni produttore creava il proprio modo di triggerare le telecamere attraverso connessioni proprietarie. Gli standard digitali consentono il controllo della telecamera ed il trasferimento dell'immagine attraverso lo stesso canale di comunicazione. Il trasferimento digitale delle immagini inoltre consente una grande flessibilità e può semplificare il progetto con una riduzione complessiva del sistema. Le applicazioni di visione richiedono quattro compiti basilari: Trovare e collegare una telecamera, configurare la telecamera, acquisire immagini dalla telecamera, gestire le sincronizzazioni fra sistema e telecamera. Due layers del software servono a gestire questi compiti come mostrato in figura 1.6. Il primo layer (Transport Layer : TL) fornisce il numero della telecamera, fornisce accesso ai registri della telecamera, si prende carico del trasferimento dei dati ed invia eventi. Il TL è governato dall'interfaccia standard hardware. A seconda dell'interfaccia il TL richiede una scheda di acquisizione oppure un adattatore per il bus (Firewire, GEVision, USB3Vision). Il secondo layer è la libreria di acquisizione delle immagini che è parte del SDK del produttore. Il software di sviluppo SDK può essere autonomo o fornito con un frame grabber oppure parte di una libreria di elaborazione immagini. Questo utilizza il TL per accedere alle funzionalità proprie della telecamera e per l'acquisizione delle immagini. Mentre lo standard per l'accesso alla telecamera e per il mapping dei registri più usato è il GenICam il quale è nativamente supportato da *Halcon*.

1.3 Sistemi di visione 3D

I sistemi per l'acquisizione in formato digitale di informazioni tridimensionali di oggetti e superfici sono numerosi ed hanno assunto negli ultimi anni un'importanza sempre più rilevante. Le applicazioni sono le più svariate: controllo dimensionale e di qualità, reverse engineering, automazione, prototipazione rapida, ricostruzione tridimensionale e virtualizzazione. I campi sono i più diffusi: industria, architettura, settore legale, tutela di beni artistici, medicina.

Le tecniche ottiche possono essere distinte in due grandi gruppi: tecniche passive e tecniche attive. Con i metodi passivi non è necessario un controllo sulla sorgente di luce: il più grosso svantaggio e rappresentato dal notevole aggravio computazionale per l'elaborazione dei dati e l'ottenimento dell'informazione di profondità. Nei metodi attivi, invece, l'uso di luce strutturata (con un preciso pattern) semplifica di molto questo problema. Nell'ultimo decennio si è assistito alla diffusione di sensori che, sfruttando differenti tecnologie, consentono di acquisire informazioni tridimensionali. Tali sistemi di acquisizione operano secondo principi di funzionamento diversi; essi sono la triangolazione, la misura del tempo di volo o altre proprietà fisico-geometriche.

1.3.1 Time-of-flight (TOF)

I sensori basati sul tempo di volo (TOF) utilizzano una tecnologia esistente da diversi anni, tuttavia si sono diffusi nel mercato consumer soltanto negli ultimi tempi dopo una sensibile riduzione dei costi di produzione. I sensori calcolano la distanza fra la sorgente e la superficie che si desidera misurare calcolando il tempo che la sorgente luminosa puntiforme impiega per arrivare sulla superficie del sensore. Gli impulsi luminosi sono segnali infrarossi inviati tramite una sorgente di luce modulata e il ricevitore è una matrice di sensori CCD/CMOS. La misurazione viene eseguita in maniera indipendente per ogni pixel della telecamera, permettendo di acquisire interamente la scena inquadrata.

Vantaggi:

- È un sistema compatto dal basso costo che non richiede particolari operazioni di installazione, al contrario di altri sistemi con maggiore accuratezza

come gli scanner laser.

- Fornisce direttamente una mappa di profondità, al contrario di telecamere stereo per le quali devono essere eseguiti algoritmi particolari per il calcolo della disparità e la triangolazione.

- Effettua la misurazioni di tutta la scena inquadrata in tempo reale; esegue tutti i calcoli utilizzando il microprocessore interno in maniera molto rapida ed efficiente, dando la possibilità di restituire solo i risultati tramite interfacce diffuse (USB o Fast Ethernet). Non richiede pertanto particolari requisiti per il computer a cui è collegato.

- E una tecnologia più robusta ai cambiamenti di luce rispetto ad altri sensori come le telecamere stereo.

Svantaggi:

- La scarsa risoluzione permette di ricavare un'informazione limitata sulla geometria della scena e sulle superfici presenti.

- Si possono verificare sbilanciamenti nelle rilevazioni causa errate disposizioni del sensore o angoli di inquadratura. Il "multipath error" è un esempio: nel caso di una scena con geometrie concave (due parenti incidenti), il segnale luminoso colpisce una parete, ma il segnale riflesso colpisce la seconda parete prima di essere diretto verso il sensore. In questo caso, la distanza della parete rilevata risulta maggiore rispetto a quella reale.

- La rilevazione della distanza lungo i bordi di superfici risulta imprecisa. I punti appartenenti a superfici che presentano un angolo di curvatura troppo ampio con la direzione della camera sono affetti da scarsa affidabilità nella stima della loro distanza. In questi tratti il segnale luminoso viene riflesso in più direzioni e il ricevitore rileva solo una parte del segnale di ritorno; questa difficoltà fisica unita alla risoluzione limitata, costringe il sensore TOF a generare un valore di profondità unico per più particolari dell'immagine.

Microsoft Kinect One

Questo accessorio sviluppato da Microsoft è pensato originariamente per l'intrattenimento in grado di creare un interazione tra giocatore e la console Xbox. Per la sua relativa economicità, questo sensore risulta essere un buon compromesso alle più costose telecamere 3D presenti sul mercato. Per tale ragione questo sensore è stato largamente impiegato in numerosi studi scientifici, ad esempio nella ricerca degli oggetti, nelle ricostruzioni 3D e nell'analisi posturale del corpo umano. La ricostruzione 3D di Kinect si basa sul principio della triangolazione attiva. Il sensore depth è basato sul chip Primesensor prodotto da Primesense che opera mediante luce infrarossa invisibile all'occhio umano. Un pattern di radiazione infrarossa viene proiettato sull'oggetto e catturato dalla telecamera IR. Il principio di acquisizione è riportato in Fig. 1.7, dove con C si identica la telecamera e con A il proiettore. Sia P_C un punto nell'immagine catturata da C e P_A il punto coniugato proiettato da A, la profondità z è valutata mediante il principio della triangolazione attiva che può essere espresso in maniera semplificata come segue:

$$z = \frac{b|f|}{d} \tag{1.1}$$

dove f rappresenta la lunghezza focale dei due dispositivi ottici, b è la distanza tra i due centri ottici e $d = u_L - u_R$ rappresenta la cosiddetta disparità ed è la differenza tra le coordinate orizzontali. La depth camera Kinect fornisce l'immagine I_K acquisita da C, la mappa di disparità $\hat{D}_k$ e la conseguente mappa depth $\hat{Z}_k$ ottenuta dalla mappa di disparità. All'aumentare dell'accuratezza, la difficoltà maggiore della triangolazione attiva matriciale consiste nel determinare le corrispondenze tra immagine e proiezione. L'utilizzo di una luce "codificata" permette di risolvere efficacemente il problema delle corrispondenze tra camera e proiettore facendo associare ad ogni pixel una posizione specifica di un pattern proiettato da A, rappresentato in figura 1.8.

In questo modo è possibile implementare in forma matriciale il principio di triangolazione attiva per acquisire la scena tridimensionale. Si consideri un pattern proiettato composto di N_R, N_C righe e colonne; quest'ultimo viene acquisito dalla telecamera C nell'immagine I_K. Ogni pixel dovrà essere associato ad una parola-codice, ovvero a una specifica configurazione locale del pattern proiettato.

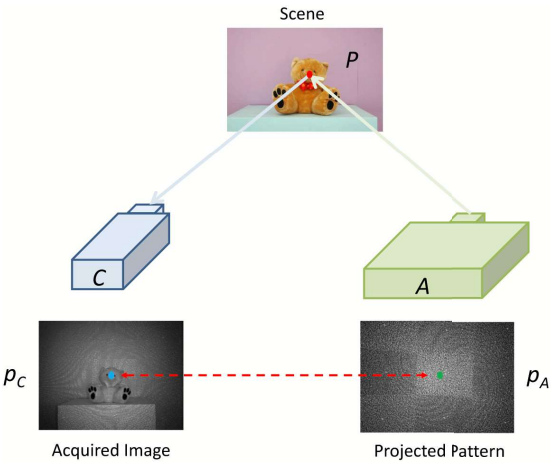

Figura 1.7: Triangolazione tra il punto p_a proiettato corrisponde un punto p_c acquisito.

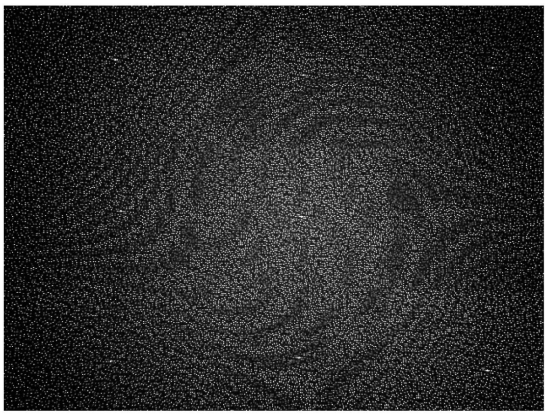

Figura 1.8: Pattern di luce codificata e proiettata dal Kinect.

Figura 1.9: Microsoft Kinect One con il sistema di riferimento coincidente al sensore di profondità.

Un algoritmo di stima della corrispondenza analizza le parole-codice ricevute e individua i pixel coniugati ai vari pixel del pattern. Il pattern adotta delle parole-codice decodificabili in maniera efficiente, in particolare fa riferimento al valore di illuminazione dei pixel attorno al punto considerato P_A.

Nel lavoro di Tesi è stata impiegata la nuova versione Kinect One (Fig. 1.9) Questa versione, come la precedente, presenta una telecamera RGB (di risoluzione 1920 $\times$ 1080) e microfoni integrati, mentre per la ricostruzione 3D utilizza un sensore Time of Flight con risoluzione 512 $\times$ 424 pixel accompagnato da un emettitore IR.

1.4 Tecniche risolutive

L'organizzazione di un sistema di visione artificiale dipende fortemente dall'applicazione in cui viene utilizzato. Alcuni sistemi sono semplici applicazioni standalone (a sé stanti) che risolvono specifici problemi di misurazione o individuazione, mentre altri costituiscono un sotto-sistema in un disegno più grande, che per esempio contiene anche altri sottosistemi per il controllo di attuatori meccanici ecc. Ci sono tuttavia delle funzioni tipiche presenti nella maggior parte dei sistemi. L'approccio tipico ad un problema di visione artificiale è composto principalmente da due fasi:

- **Pre-Elaborazione:** Prima che una immagine venga trattata da un metodo di visione artificiale è solitamente necessario elaborare i dati in modo da verificare che questi soddisfino specifiche regole necessarie al metodo. Per esempio togliere eventuale rumore che introduce informazioni false, oppure modificare il contrasto per assicurarsi che le informazioni rilevanti vengano individuate.

- **Feature extraction:** Una volta effettuata la pre-elaborazione si vogliono ottenere dei risultati, questo viene effettuato attraverso l'uso di particolari metodi in modo da ottenere informazioni significative ad esempio delle aree, oppure attraverso o studio delle componenti principali di determinate regioni connesse. Uno strumento molto utile è anche l'operazione di *threshold*, in quanto fissato un valore, appunto di soglia, si genera un immagine binaria. In questo modo è possibile ad esempio eliminare lo sfondo da un immagine rendendo più semplici le future operazioni.

- **Matching:** Un elaborazione più complessa di una semplice feature extraction è l'utilizzo di algoritmi di matching. In molte situazioni è necessario andare ad identificare un oggetto, fornendone quindi le coordinate rispetto ad un determinato sistema di riferimento. Vi sono diversi algoritmi ad alto livello che permettono di risolvere questo problema, verranno quindi descritti i principali metodi utilizzati da *Halcon*.

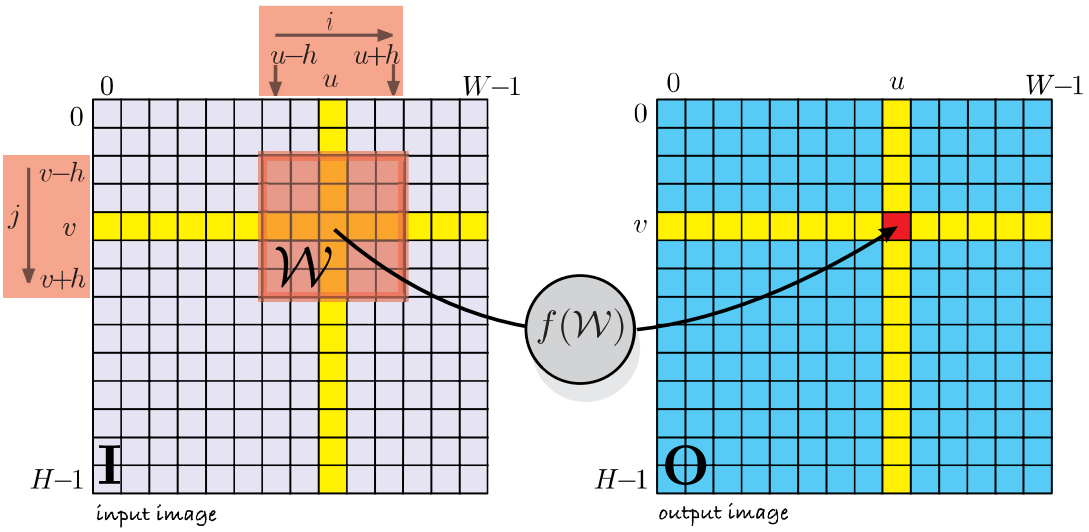

Figura 1.10: Schema delle operazioni di convoluzione di una finestra in un immagine.

Verranno di seguito introdotte le più comuni tecniche sia per il pre-processing sia per il matching che verranno utlizzate in seguito durante le prove sperimentali.

1.4.1 Pre-Elaborazione

Per poter eseguire operazioni complesse su un immagine di solito non basta una buona acquisizione. Ma spesso è necessario eseguire delle operazioni aggiuntive. Le più comuni l'applicazione di un filtro mediano (smoothing), migliorare il contrasto. Queste operazioni che verranno spiegate brevemente spesso sono necessarie a garantire il funzionamento delle elaborazioni più complesse ad esempio per gli algoritmi di matching oppure di controllo qualità.

- **Filtro Mediano:** nella teoria dei segnali, un filtro mediano è un filtro digitale che viene utilizzato per eliminare il rumore da immagini o da segnali digitali.

  Solitamente questo filtro viene utilizzato come pre-processing prima di effettuare altre operazioni, come ad esempio trovare i contorni. È anche spesso usato per rimuovere il rumore sale e pepe.

Figura 1.11: La figura a sinistra mostra l'immagine originale. Quella centrale mostra l'immagine modificata con un kernel costante, mentre quella a destra mostra un immagine filtrata con un Gaussian kernel con $\sigma = 5$.

Il principio di funzionamento di questo filtro è di lavorare su singolo campione, andandolo a sostituire con il valore mediano dei suoi vicini, rappresentati da una cosiddetta finestra, con al centro il valore da sostituire. L'approccio che si vuole applicare è quello di una convoluzione come mostrato in fig. 1.10 , nella quale è mostrato come la finestra viene applicata per ottenere il valore nell'immagine di uscita. Un tipico esempio di filtro mediano è la finestra di gauss bidimensionale:

$$G(u,v) = \frac{1}{2\pi\sigma^2} e^{\frac{u^2+v^2}{2\sigma^2}} \tag{1.2}$$

Questa finestra è il cosidetto kernel da applicare all'immagine secondo la convoluzione:

$$O(u,v) = \sum_{(i,j \in W)} i[u+i, v+j] G(u,v), \quad \forall (u,v) \in I \tag{1.3}$$

In questo modo ottengo l'immagine mediata andando ad esempio a ridurre il rumore ad alte frequenze. L'operazione di convoluzione su un immagine come descritto nell'eq.1.3 è alla base delle elaborazioni di immagini che verranno descritte nei prossimi metodi.

- **Aumentare il Contrasto:** A volte per distinguere alcuni dettagli come bordi e angoli è necessario aumentare il contrasto dell'immagine. Per ottenere tale risultato vi sono diverse operazioni. Il metodo che si userà sarà

quello utilizzato da Halcon: in primo luogo viene applicato un filtro mediano ottenendo un valore di media m, di conseguenza viene poi effettuato il calcolo:

$$O(u,v) = \text{round}((I[u,v] - m)\alpha) + I[u,v] \tag{1.4}$$

Dove α è un fattore di scala. In questo modo è possibile rendere i bordi e i gli angoli più nitidi.

1.4.2 Feature extraction

L'obbiettivo è quello di segmentare o separare i pixel considerati degli oggetti di interesse dagli altri della scena. Vengono quindi introdotte le più comuni tra cui sono la sogliatura (thresholding),e la segmentazione (segmentation). Queste tecniche sono molto importanti in quanto permettono il riconoscimento di un determinato oggetto fornendone le coordinate[1].

- **Sogliatura:** nell'elaborazione digitale delle immagini questa operazione è un semplice metodo per segmentare un'immagine. Da un'immagine a livelli di grigio, la sogliatura restituisce un'immagine binaria. Durante un processo di sogliatura, singoli pixel dell'immagine sono catalogati come "pixel oggetto" se il loro valore è maggiore di una certa soglia e come "pixel di sfondo" se il valore è sotto la soglia. Solitamente l'immagine binaria in uscita ha valore pari a "1" dove è presente l'oggetto e pari a "0" per lo sfondo, ottenendo quindi un'immagine in bianco e nero. Per ottenere questo tipo di immagine è necessario eseguire la seguente operazione:

$$I_{binaria}(x) = \begin{cases} 0 & \text{se} \quad I_{originale}(x) < t \\ 1 & \text{se} \quad I_{originale}(x) \geq t \end{cases} \tag{1.5}$$

  Dove t è la soglia desiderata. Il parametro chiave in una sogliatura è la scelta del valore di soglia, ad esempio un determinato valore di grigio. Esistono diverse metodologie per la scelta automatica di questo valore, oppure è possibile sceglierlo a mano.

- **Segmentazione:** La segmentazione di un'immagine nell'elaborazione digitale delle immagini è il processo di partizione di un'immagine in regioni

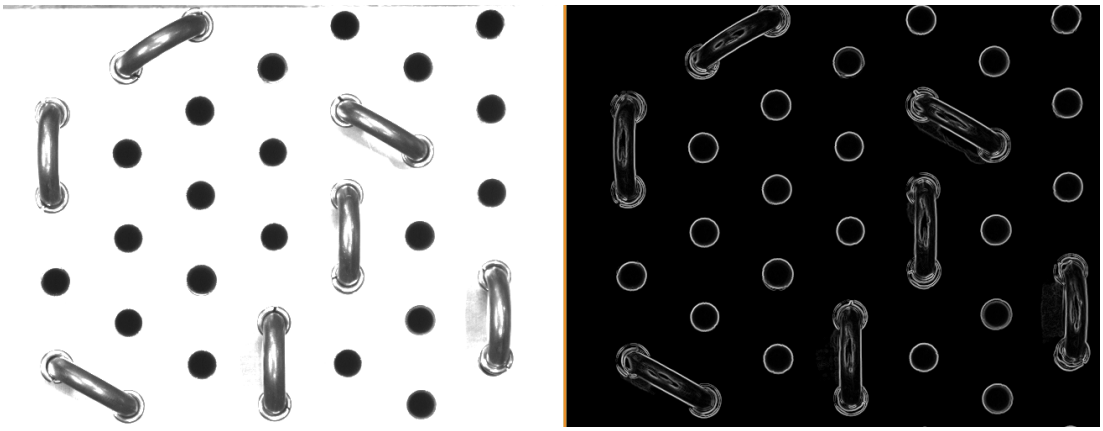

Figura 1.12: A sinistra è mostrata l'immagine originale, a destra è mostrata quella ottenuta tramite segmentazione.

significative. Lo scopo della segmentazione è semplificare e/o cambiare la rappresentazione delle immagini in qualcosa che è più significativo e facile da analizzare.

La segmentazione è di solito utilizzata per localizzare oggetti e bordi (linee, curve, ecc.). In questo processo si classificano i pixel dell'immagine che hanno caratteristiche comuni, pertanto ciascun pixel in una regione è simile agli altri della stessa regione per una qualche proprietà o caratteristica (colore, intensità o texture). Regioni adiacenti sono significativamente differenti rispetto ad almeno una di queste caratteristiche. L'estrazione di bordi è un operazione molto importante e per svolgerla si può utilizzare una convoluzione come per il filtro mediano ma applicando un diverso kernel, nello specifico quello più diffuso è il cosidetto filtro di sobel:

$$G_u = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \qquad G_v = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{1.6}$$

Applicando all'immagine questi due filtri si ottengono le derivate discrete dell'immagine rispetto a u, v con il primo è possibile distinguere un bordo verticale mentre con il secondo uno orizzontale. L'unione di questi due possono formare un immagine rappresentativa di solo i contorni degli oggetti all'interno. Come mostrato in figura 1.12.

- **Momenti:** I momenti sono delle variabili in grado di contenere moltissima informazione e sono semplici da calcolare, questi sono in grado di descrivere grandezza e posizione di una regione (ad esempio dopo una sogliatura). Il momento di un immagine I è lo scalare:

$$m_{pq} = \sum_{(u,v \in I)} u^p v^q I[u,v] \tag{1.7}$$

Dove $(p+q)$ è l'ordine del momento. Il momento di ordine zero è rappresentato con $p = q = 0$

$$m_{pq} = \sum_{(u,v \in I)} I[u,v] \tag{1.8}$$

questo per una immagine binaria dove lo sfondo è rappresentato dai pixel uguali a zero, rappresenta il numero di pixel (bianchi) della regione.
I momenti possono avere un'interpretazione fisica di una distribuzione di massa. Dove ogni pixel considerato oggetto gli viene attribuita un unità di massa. La massa totale della regione m_{00} e il centro di massa del centroide della regione è:

$$u_c = \frac{m_{10}}{m_{00}}, \quad v_c = \frac{m_{01}}{m_{00}} \tag{1.9}$$

dove m_{10}, m_{01} sono i momenti del primo ordine. Per quanto riguarda invece il momento centrale di ordine p, q è :

$$\mu_{pq} = \sum_{(u,v \in I)} (u - u_c)^p (v - v_c)^q I[u,v] \tag{1.10}$$

questi sono invarianti riguardo la posizione della regione. Di conseguenza se relazionati a m_{pq} con:

$$\mu_{10} = 0, \quad \mu_{01} = 0 \tag{1.11}$$

$$\mu_{20} = m_{20} - \frac{m_{10}^2}{m_{00}}, \quad \mu_{02} = m_{02} - \frac{m_{01}^2}{m_{00}}, \quad \mu_{11} = m_{11} - \frac{m_{10} m_{01}}{m_{00}} \tag{1.12}$$

La matrice di inerzia della regione diventa allora:

$$J = \begin{bmatrix} \mu_{20} & \mu_{11} \\ \mu_{11} & \mu_{02} \end{bmatrix} \tag{1.13}$$

Andando a calcolare l'ellisse equivalente con la stessa matrice di inerzia della regione, denotando gli autovalori λ_1, λ_2 e gli autovettori v_1, v_2 questi

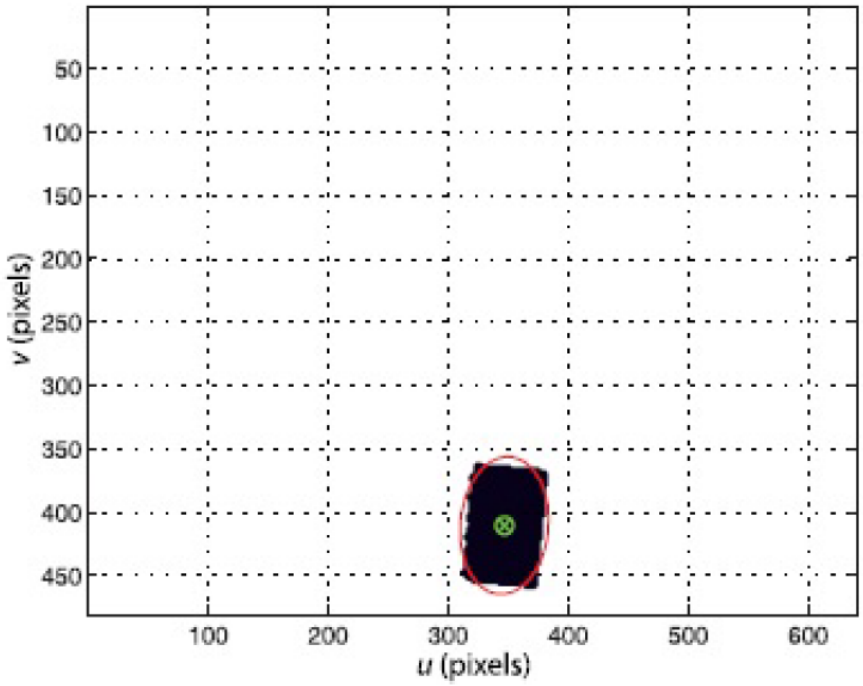

Figura 1.13: Ellisse equivalente della regione ottenuta.

si relazionano ai raggi dell'ellisse in modo che il massimo e minimo diventa:

$$a = 2\sqrt{\frac{\lambda_2}{m_{00}}}, \qquad b = 2\sqrt{\frac{\lambda_1}{m_{00}}} \tag{1.14}$$

Gli autovettori denotano quindi la direzione degli assi dell'ellisse di conseguenza per ottenere l'angolo tra i due :

$$\theta = \tan^{-1}\frac{v_{2,y}}{v_{2,x}} \tag{1.15}$$

Dove $v_{2,x}, v_{2,y}$ sono rispettivamente le componenti , y di v_2.

In questo modo è possibile attraverso i momenti descrivere posizione e orientazione degli oggetti ottenuti tramite alcune elaborazioni ad esempio tramite treshold.

1.4.3 Algoritmi di Matching

Gli algoritmi di matching permettono di trovare in maniera robusta la posizione e orientazione di determinate forme, questo ha molteplici usi come il riconoscimento di un determinato oggetto per andare in presa con un robot. L'idea principale del

matching è quella di utilizzare un template creato da un set di immagini(anche contenente una sola immagine) e andare a "cercare" questo modello in un diverso set. Per poter creare questo modello Halcon utilizza diversi metodi molto semplici da utilizzare anche per chi non ha grande esperienza nella computer vision. Ci sono due tipologie di approccio per risolvere questo problema: il primo consiste nell'utilizzare i valori di grigio dell'immagine campione da confrontare con quella nuova, questo metodo è il Correlation-based matching, il quale utilizza la normalized cross correlation (NCC) per trovare oggetti o patterns. Il secondo invece si basa invece di andare a creare un modello del contorno dell'oggetto che si vuole trovare (Shape-based-matching). Per entrambi i metodi per ricavare il modello iniziale in *Halcon* è necessario definire una ROI (Region of interest) in modo da ritagliare solamente la parte significativa dell'immagine. In questo modo a seconda del metodo usato viene creato il modello che si userà nelle corrispettive funzioni di *find*.

Correlation-based matching

Uno dei metodi più classici, per risolvere il problema del matching, è quello di utilizzare un immagine come template per poi ricercarla nelle immagini successive. Halcon per risolvere questo problema fornisce i metodi necessari, per prima cosa dopo aver definito una ROI, ricavata l'immagine di template si deve usare il metodo *create_ncc_model*. Questo permette di creare un template usando diversi parametri ottenendo un trade-of tra qualità e velocità di ricerca. Questo metodo deve essere utilizzato assieme al suo complementare *find_ncc_model*, quest'ultimo, dato in ingresso un immagine e il template, restituisce in uscita le coordinate in pixel della posizione dell'oggetto. Anche qui vi sono diversi parametri da impostare, il principale tra questi è il "Min Score". Questo parametro definisce il valore per il cui un oggetto è ritenuto trovato in una determinata posizione e orientazione. Per calcolare questo score si possono usare diversi kernel, ma hanno in comune la struttura presentata precedentemente (eq.1.3). Nello specifico la formulazione usata, definita l'immagine template come $t(u, v)$ e l'immagine su cui

cercare $i(u, v)$, per ottenere il valore di score per un singolo pixel è la seguente:

$$\mathrm{ncc}(u, v) = \frac{1}{n} \sum_{(r,c) \in W} \frac{t(r, c) - m_t}{\sqrt{s_t^2}} \cdot \frac{i(u + r, v + c) - m_i(u, v)}{\sqrt{s_i^2(u, v)}} \tag{1.16}$$

Dove n denota il numero di punti del template, W denota la finestra(ROI), m_t è il valor medio di grigio del template:

$$m_t = \frac{1}{n} \sum_{(r,c) \in W} t(r, c) \tag{1.17}$$

s_t^2 è la varianza dei valori di grigio del template:

$$s_t^2 = \frac{1}{n} \sum_{(r,c) \in W} (t(r, c) - m_t)^2 \tag{1.18}$$

$m_i(u, v)$ è il valor medio di grigio dell'immagine alla posizione (u, v) in una finestra della dimensione del template:

$$m_i(u, v) = \frac{1}{n} \sum_{(r,c) \in W} i(u + r, v + c) \tag{1.19}$$

$s_i^2(u, v)$ è la varianza dei valori di grigio dell'immagine i alla posizione (u, v) rispetto a tutti i punti del template:

$$s_i^2(u, v) = \frac{1}{n} \sum_{(r,c) \in W} (i(u + r, v + c) - m_i(u, v)))^2 \tag{1.20}$$

In questo modo il valore calcolato di $\mathrm{ncc}(u, v)$ è tra 0 e 1. In particolare se *ncc* è nell'intorno di 1 vuol dire i valori di grigio in quelle coordinate sono molto simili a quelli del template, di conseguenza può essere considerato un oggetto trovato. Altri parametri utili da impostare possono essere il range di angoli con cui far cercare il template, in maniera da diminuire il più possibile numero di iterazioni del metodo. Un altro parametro degno di nota è il numero di livelli della piramide, che permette di andare a velocizzare di molto la procedura andando a diminuire la qualità dell'immagine.

Shape-based matching

Lo *shape-based matching* è un sistema analogo allo correlation-based matching. Ma si basa sul principio, secondo il quale, il template ottenuto è un immagine

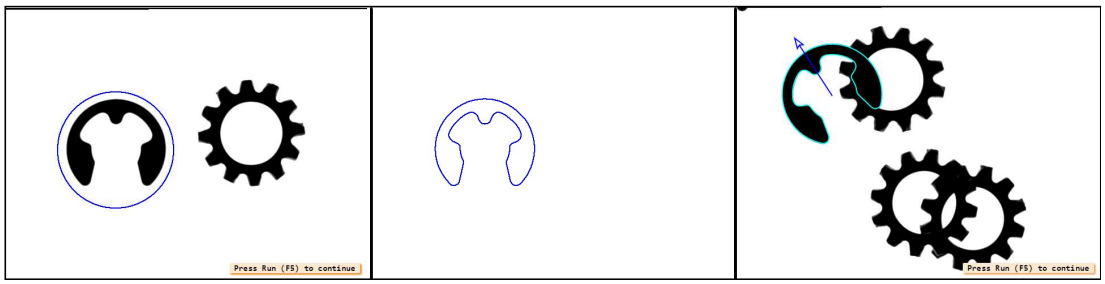

Figura 1.14: A sinistra è possibile vedere la ROI manualmente selezionata tramite HDevelop. In centro è possibile vedere il modello estratto dalla ROI tramite *create_shape_model*, mentre a destra è possibile osservare il risultato del metodo *find_shape_model* col modello creato.

binaria con la quale si identifica solamente il bordo dell'oggetto da cercare. La base del metodo è l'utilizzo di un filtro, questo in modo da trovare solamente i bordi degli oggetti all'interno, come per il sobel kernel, per poi elaborare questi. Nella figura 1.14 è possibile notare i passi da svolgere in Halcon per riuscire a fare matching. In particolare possiamo osservare come si può impostare la ROI (fig.1.14.a), per ricavarsi il contorno da usare come modello(fig.1.14.b) con il *create_shape_model*, infine si vede il risultato del metodo *find_shape_model* con il quale si trovano le coordinate dell'oggetto. L'operatore usato da Halcon per creare il template è *create_shape_model* e come per il metodo precedente vi sono molteplici parametri da impostare. Ad esempio è possibile scegliere il numero di livelli della piramide, come può variare l'angolo dell'oggetto da trovare (mettendo un angolo ristretto si eseguono meno iterazioni durante il *find*). Un altro parametro fondamentale è la metrica. Quest'ultima serve a determinare sotto quali condizioni di luce deve venire riconosciuto l'oggetto. In particolare se la metrica è impostata su *"use polarity"* le due immagini devono avere lo stesso contrasto, mentre se viene usato ad esempio con la metrica "ignore local polarity" l'oggetto viene riconosciuto anche se il contrasto cambia localmente, un esempio è fornito in figura 1.15. Questo approccio permette di avere soluzioni più robuste alla variazione della luce rispetto al template iniziale. Il tutto rendendo meno dispendiosa la parte di illuminazione, cosa non possibile con la soluzione precedente. Inoltre l'immagine essendo binarizzata, l'intera operazione risulta meno onerosa dal punto di vista

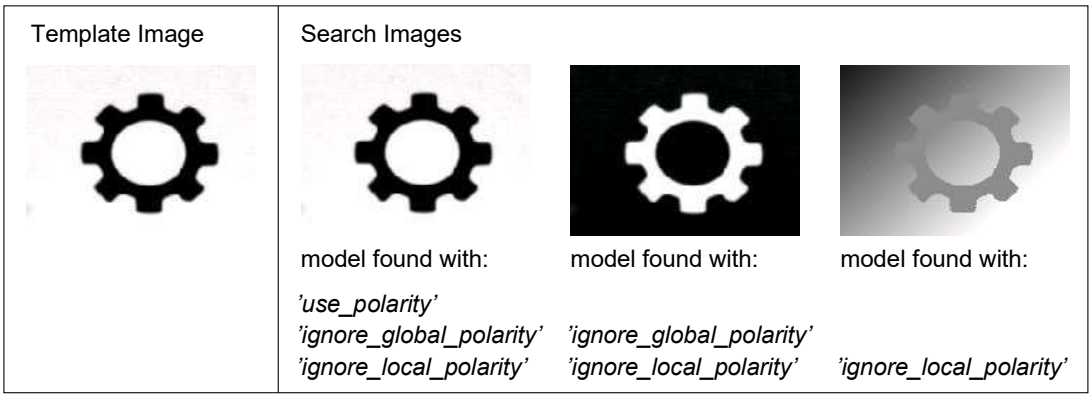

Figura 1.15: Influenza della metrica utilizzata durante lo shape-based-matching

computazionale. Infatti in questo lavoro sono stati confrontati i due metodi dal punto di vista del tempo necessario al calcolo. Il risultato mostrato nel capitolo 3 favorirà lo shape-based matching sia in termini di media che di deviazione standard. La soluzione trovata sarà la coppia di pixel (u, v) dove la funzione score sarà maggiore di una certa soglia impostata.

Capitolo 2

Integrazione di Halcon con MATLAB

In questo capitolo verranno descritti come è possibile utilizzare in *Matlab* i risultati ottenuti tramite Halcon. Nella prima sezione viene descritto l'utilizzo di Halcon e quali vantaggi riesce a portare lo sviluppare applicazioni di visione in questa piattaforma. Poi si discute di come sono stati implementati e discussi diversi metodi per poter sfruttare Halcon all'interno di *Matlab* in maniera efficiente.

2.1 Halcon

MvTec Halcon[2] è una delle librerie utilizzate nell'ambito della visione industriale, porta notevoli vantaggi sia in termini di prestazioni, sia in termini di tempo necessario per sviluppare un'applicazione di visione all'interno del suo ambiente di sviluppo *HDevelop*. Questa può essere facilmente esportata attraverso il suo tool e utilizzata dai più comuni linguaggi come C, C++, .NET (C#, VB.NET). Inoltre *Halcon* ha la possibilità di potersi interfacciare nativamente con le più comuni telecamere e framegrabber supportando diversi standard come GenICam, GigE Vision, e USB3 Vision. In questo lavoro di tesi si sono sviluppati diversi modi per esportare i risultati ottenuti dal runtime di Halcon e poterli utilizzare all'interno di Matlab. Questo può portare numerosi vantaggi dato che *Matlab* può facilmente configurarsi con molti altri software e hardware come verrà discusso nel capitolo

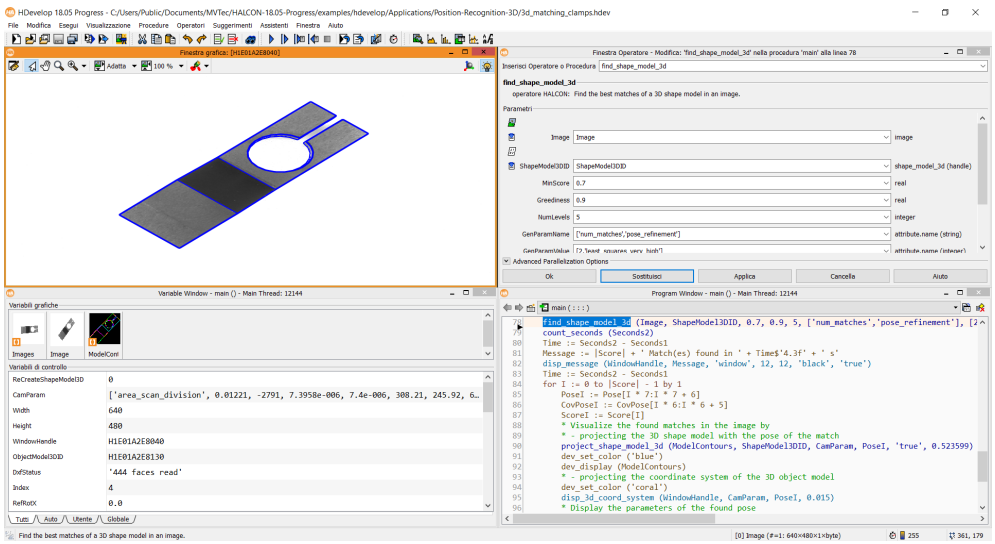

Figura 2.1: Schermata di HDevelop, in cui è possibile osservare la finestra grafici in alto a sinistra, la finestra variabili in basso a sinistra, la finestra operatore in alto a destra e infine la Program Window in basso a destra.

3, nel quale l'utilizzo combinato di Halcon/Matlab/Ace permette di risolvere un problema di *pick and place* attraverso l'uso di un robot. L'ambiente *HDevelop* è l'IDE di *Halcon*, questo particolare IDE è in grado di fornire moltissimi strumenti utili nella fase di progettazione per un qualsiasi applicativo di visione. Come si può vedere in figura 2.1, *HDevelop* fornisce una semplice interfaccia in grado di vedere ad ogni passo visivamente i risultati dello pseudocodice proprietario. Ogni operazione infatti che viene effettuata può essere applicata direttamente sull'immagine passo-passo, in modo da rendere lo sviluppo molto intuitivo e veloce. È presente inoltre la *Variable Window* dove è possibile osservare tutte le variabili passo-passo durante il debug. La *finestra operatore* infine agevola moltissimo la compilazione dei campi ogniqualvolta si vuole richiamare un metodo, andando quindi a compilare in maniera automatica oppure consigliando i valori da inserire. Tutte queste caratteristiche, assieme al vasto numero di esempi, sono in grado di agevolare moltissimo la fase di pre-elaborazione ed operazioni successive.

Utilizzare il linguaggio proprietario di Halcon è molto semplice, in particolare in fase di acquisizione e nelle operazioni spesso con un solo comando si riesce ad eseguire anche operazioni complesse. Come si può vedere in figura 2.2, per inizia-

lizzare la camera basta utilizzare il comando *open_framegrabber* e per andare ad acquisire un immagine si utilizza il comando *grab_image_async*. Una volta acquisita l'immagine è possibile iniziare la fase di elaborazione dell'immagine in cui è possibile utilizzare i metodi standard di halcon come nel caso dell'applicazione di un semplice filtro mediano con il comando *smooth_image* oppure si può eseguire una segmentazione tramite il comando *threshold*. HDevelop inoltre fornisce una serie di assistenti molto utili nelle operazioni più comuni di visione, ad esempio in figura 2.3 è possibile vedere l'assistente per inizializzare la telecamera e completare il campo del metodo *open_framegrabber* in maniera automatica e senza dover scrivere righe di codice. Inoltre permette di inserire il codice corrispondente in maniera immediata senza dover scrivere ogni singolo comando. Vi sono numerosi assistenti i più comuni possono essere ad esempio l'assistente di calibrazione che permette di acquisire le foto per la calibrazione direttamente dalla finestre vedendo in anteprima la qualità, oppure vi sono diversi assistenti per le misure effettuate su immagine oppure per le operazioni su OCR.
Vi sono due tipi di variabili le *Iconic variable* e le *HTuple*. Le *iconic varible* fanno parte della classe *HObject*, queste possono essere degli oggetti in grado di contenere anche un grande numero di immagini e/o regioni. Le *HImage* e *HRegion* possono essere usate semplicemente come *HObject*, in particolare le *HRegion* sono immagini binarie che di norma sono i risultati di alcuni metodi, per esempio l'operatore *threshold*. Il formato delle *HImage* è un immagine in formato RGB (se a colori) quindi differente rispetto al classico bitmap come mostrato in figura 2.4.

```
1 * Inizializzare la telecamera
2 open_framegrabber ('File', 1, 1, 0, 0, 0, 0, 'default', -1, 'default'
3 grab_image_start (AcqHandle, -1)
4 * Acquisizione immagine
5 grab_image_async (Image, AcqHandle, -1)
6 * Pre elaborazione
7 smooth_image (Image, ImageSmooth, 'gauss', 2)
8 *Feature matching
9 threshold (ImageSmooth, Region, 128, 255)
```

Figura 2.2: Esempio di codice in Halcon, in particolare si esegue l'inizializzazione della camera, acquisizione, filtraggio e sogliatura.

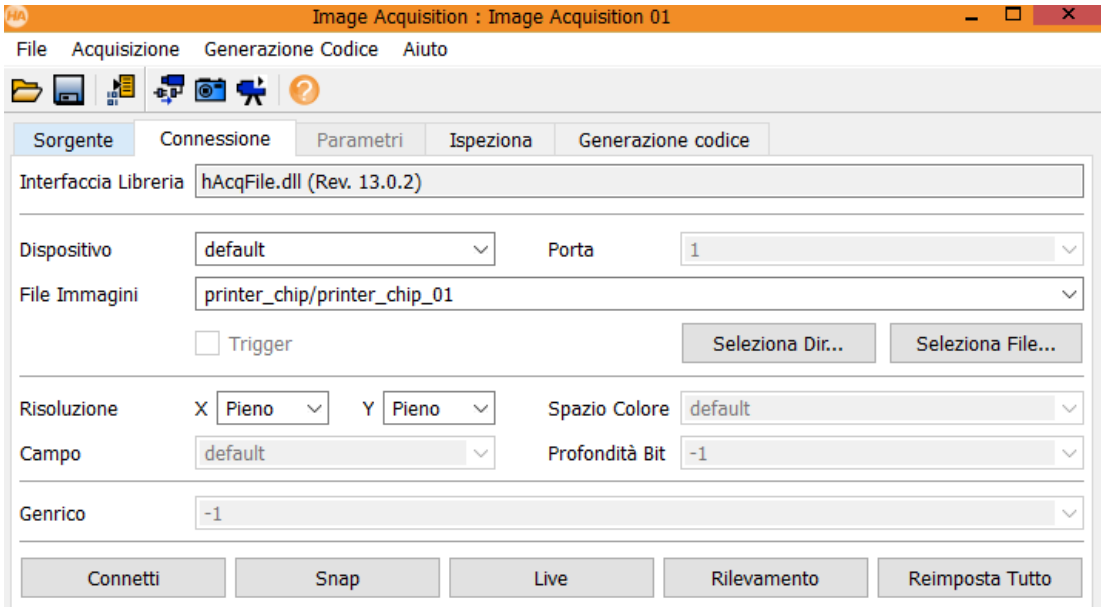

Figura 2.3: Image Assistant fornito da Halcon, questo permette di configurare la telecamera in maniera immediata.

Nelle *HImage* infatti vi sono tre canali in cui vengono salvati i rispettivi valori di

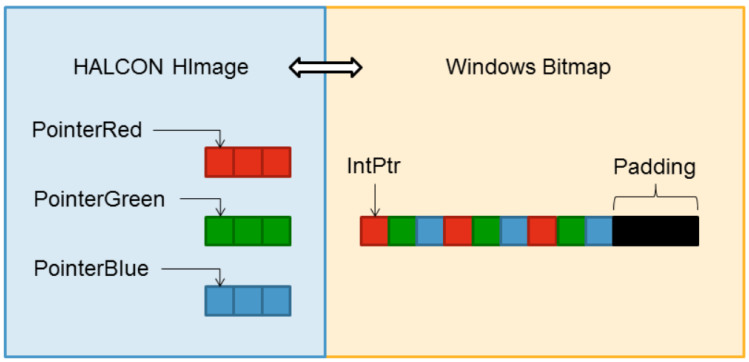

Figura 2.4: A sinistra è mostrato il formato immagine RGB di Halcon, mentre a destra è mostrato il formato standard Windows Bitmap.

ogni colore. Questo richiede infatti delle particolare procedure per poter utilizzare gli *HObject* all'esterno di *HDevelop*. Le *HTuple* sono degli array contenti valori alfanumerici per risultati oppure semplici variabili. Per poter utilizzare i metodi Halcon è necessario usare unicamente questi due tipi di variabili, non sono infatti disponibili diversi formati per utilizzare i metodi. Buona norma se si vuole esportare un programma realizzato tramite *Halcon* è quello di completare l'intera

procedura ed esportare solo i risultati in termini di immagini e tuple.
Per poter esportare il codice in un linguaggio classico di programmazione, ad esempio per creare un semplice eseguibile in grado di far partire la procedura, è necessario utilizzare il tool di esportazione fornito da Halcon sotto il menu "File/Esporta". Una volta selezionato apparirà una finestra come in figura 2.5. In

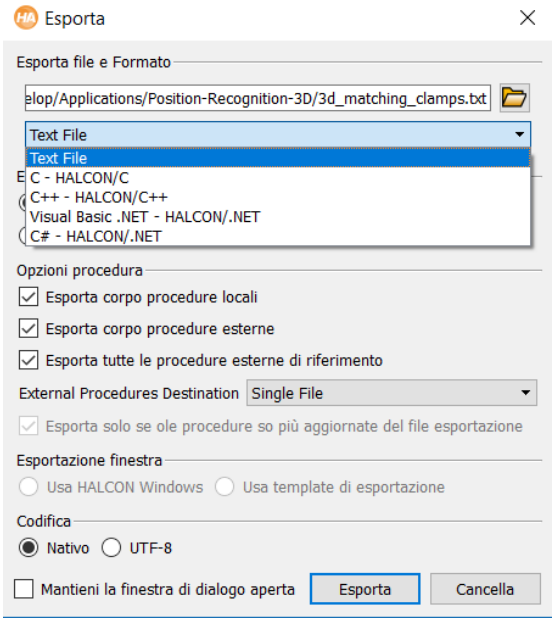

Figura 2.5: Tool di esportazione di Halcon, nel quale è possibile scegliere con che linguaggio esportare il codice.

questo modo è possibile esportare la procedura come semplice file di testo oppure come codice per il linguaggio scelto. Il file che si genera in questo modo non è altro che un programma in cui viene eseguita la procedura main, la quale non fa altro che richiamare la funzione *action()*. Quest'ultima è esattamente l'intera procedura sviluppata nello pseudocodice in *HDevelop* solamente utilizzando le classi del linguaggio scelto.

2.2 Importazione Halcon stand-alone

Una volta sviluppata la propria applicazione all'interno dell'ambiente di sviluppo HDevelop è possibile esportarla attraverso il tool di esportazione, questo può creare degli script da poter essere utilizzati in diversi linguaggi, nello specifico

in C++, C#, C, Visual Basic. Lo script generato per ogni tipo di linguaggio è in grado, una volta compilato, di creare un programma eseguibile(.exe) nel quale avviene la replica delle elaborazioni effettuate in HDevelop. Questa soluzione può portare diversi vantaggi nel momento in cui si vogliono ottenere i risultati in *Matlab*. Ad esempio, come sarà mostrato nella sezione dei test, nella prova del nastro si è preferito utilizzare *Halcon* stand-alone per l'acquisizione lasciando libere le risorse di *Matlab*. Questo è stato suggerito dal fatto che l'acquisizione deve avvenire in maniera continuativa e non si voleva andare a usare le risorse di *Matlab*. Per importare in *Matlab* i risultati sono state implementate due alternative: interscambio dati tramite salvataggio su HDD e l'invio tramite socket.

2.2.1 Interscambio tramite HDD

E' il metodo più semplice nel quale si utilizzano i metodi Halcon per il salvataggio dei risultati. Per le *HTuple* questo avviene attraverso il metodo *write_tuple* in modo da salvare, ad esempio in un file di testo (.txt), i valori ottenuti dalle elaborazioni come delle coordinate. Per salvare invece un immagine si utilizza il metodo *write_image* salvando sempre su HDD l'immagine voluta nel formato desiderato (ad esempio .bmp) non dovendo quindi affrontare il problema del formato immagine RGB di *Halcon* come mostrato in figura 2.4. Nel caso in cui si volessero salvare le elaborazioni, che si vedono nella window di *HDevelop*, bisogna utilizzare il comando dumb_image. Questo infatti crea una nuova immagine in modo da sovrapporre regioni e sfondo in maniera immediata. Per aumentare le prestazioni inoltre è possibile, attraverso appositi programmi, creare una allocazione di memoria virtuale da utilizzare come HDD. Un RAM disk infatti è molto più veloce nello leggere/scrivere nella memoria, rendendo il processo molto più veloce. Se invece si vuole salvare una *HTupla* di Halcon, si può utilizzare il metodo *write_tuple* senza specificare uno specifico formato. Per importarla dentro il workspace di *Matlab* si devono utilizzare delle funzioni per scompattarle. Oppure uno strumento molto semplice e efficace è usare il tool di Matlab "import data", andando addirittura a creare automaticamente delle funzioni in modo da importare le varie selezioni. Questo tipo di soluzione è molto semplice e non serve andare a modificare il codice per importare i dati in *Matlab* una volta compilato

il programma, di conseguenza è la soluzione più indicata per applicazioni che non necessitano di particolari performance oppure di complessi sistemi di visione.

2.2.2 Socket

Un altro metodo per sfruttare *Halcon* da Matlab è quello di usare i protocolli di comunicazione. Un socket, in informatica, indica un'astrazione software progettata per utilizzare delle API standard e condivise per la trasmissione e la ricezione di dati attraverso una rete oppure come meccanismo di IPC. È il punto in cui il codice applicativo di un processo accede al canale di comunicazione per mezzo di una porta, ottenendo una comunicazione tra processi che lavorano su due macchine che possono essere fisicamente separate o dalla stessa. Un socket è un particolare oggetto sul quale leggere e scrivere i dati da trasmettere o ricevere. Questa soluzione è particolarmente vantaggiosa se si utilizzano dispositivi separati oppure se si vogliono sfruttare diversi applicativi come Matlab, in questo caso mettendosi in ascolto/ inviando dati sulla stessa porta per ottenere facilmente le informazioni. Un problema di questo approccio è che per inviare i dati bisogna andare a modificare lo script ottenuto da Halcon, in particolare bisogna includere le librerie e aggiungere all'esecuzione del programma l'intera parte inerente alla comunicazione, dovendo quindi dover mettere le mani al codice cosa che può non risultare semplice. Sono stati sviluppati dei metodi per facilitare questo passaggio, nello specifico sono stati sviluppati due metodi per permettere di inviare a Matlab i valori ottenuti dalle HTuple e immagini. Questo è necessario dato che Halcon fornisce i risultati nelle proprie classi non compatibili con Matlab, il lavoro è stato fatto solamente nel linguaggio C++ però i concetti possono essere esportati negli altri linguaggi compatibili con *Halcon*. Per poter sfruttare questo sistema conviene isolare tutte le variabili desiderate in poche o in un unica HTuple. Questo perché è possibile andare a ispezionare le *HTuple* con il metodo *tupleselect()*, in questo modo è possibile eseguire un cast del singolo elemento e copiarlo in un array da poter spedire tramite Socket a matlab. Lo stesso principio viene poi applicato anche per le immagini, le quali vengono spedite, bit per bit, dalla memoria direttamente al socket dato che non è possibile in altro modo fare un cast della classe HImage/HObject. Per fare ciò è stato creato un metodo chiamato *sendImage()* il

quale permette di convertire l'immagine in byte per poi spedirla tramite socket. Questo metodo si basa sul comando di halcon *GetImagePointer1()* che permette di avere un puntatore sull'immagine, quest'ultimo però è un formato di Halcon quindi bisogna effettuare un doppio cast. Prima in *HLong* (sempre un formato di Halcon) e poi in *void**, così da poter essere usato da *memcpy* funzione C++ in grado di copiare blocchi di memoria. A questo punto una volta calcolati i bit dell'immagine si copiano in un array di byte che verrà poi spedito. A questo punto basta mettere Matlab in ascolto dei dati trasmessi e si ottengono i valori in grayscale delle immagini desiderate. È necessario inoltre eseguire un reshape in modo da ottenere delle matrici col le dimensioni delle immagini corrette.

2.3 Utilizzare Halcon come libreria

L'utilizzo di Halcon come libreria di appoggio per *Matlab* può portare numerosi vantaggi, infatti la possibilità di poter essere richiamati direttamente da Matlab permette di poter decidere il momento in cui avviene l'acquisizione e la conseguente elaborazione. In entrambe le soluzioni presentate è necessario adattare il codice esportato dal tool di Halcon in un formato adeguato alla piattaforma scelta, in particolare sono stati creati i metodi in modo da rendere semplice questo trasferimento. In entrambi i linguaggi sono stati introdotti i metodi per l'esportazione dei risultati di Halcon, nello specifico:

- **HalconTuple2MatlabArray**: questo metodo permette l'esportazione di una HTupla contente qualsiasi risultato numerico o di tipo stringa.
- **HalconObject2MatlabArray** : questo metodo permette di esportare un intero HObject il quale può contenere numerose immagini in un formato matriciale da utilizzare con Matlab.

2.3.1 Libreria C#

Per poter compilare e utilizzare Halcon come una libreria utilizzando C# bisogna impostare il riferimento disponibile nella cartella ($HALCONROOT$ \bin\dotnet35\) e importare il file halcondotnet.dll (maggiori informazioni guida "programmer's

```
1   using System;
2   using HalconDotNet;
3   using System.Runtime.InteropServices;
4
5   namespace ClassImportHalcon
6   {
7       public class ExportTupleImages
8       {
9           public struct results...
16
17          static public results FromHalcon()
18          {
19              results Output;
20              Output = action();
21              Output.matlab_Images = HalconObject2MatlabArray(Output.imageResults);
22              Output.matlab_Array = HalconTuple2MatlabArray(Output.tupleResults);
23              return Output;
24          }
25
26          // Main procedure
27          static public results action()...
93
94          // Funzioni di supporto
95          public static byte[][][] HalconObject2MatlabArray(HObject hImagep)...
150
151         static public byte[][] HalconImage2MatlabArray(HObject hImagep)...
195
196         static double[] HalconTuple2MatlabArray(HTuple hData)...
273     }
274
275 }
276
```

Figura 2.6: struttura della classe C# per l' esportazione di Halcon in Matlab

guide"). Inoltre bisogna consentire nelle impostazioni del programma C# in Visual studio la possibilità di compilare codice *unsafe* nella sezione compilazione delle proprietà, in modo da poter usare la classe INTPTR di C#. Nella soluzione presentata sono stati implementati solamente la conversione di immagini in grayscale a 8 bit, mentre per quanto riguarda i risultati delle tuple solamente gli array numerici.

A questo punto bisogna modificare lo script esportato da Halcon, questo infatti genera un file che consente la compilazione di un app stand-alone comprensiva di main. Bisogna quindi realizzare una classe libreria in grado da poter essere richiamata direttamente da Matlab. È stata riportato lo schema in fig.2.6 della tipica soluzione utilizzata e testata per risolvere questo problema. Come si può notare per poter utilizzare questa classe è necessario importare i riferimenti ne-

cessari. Nello specifico *HalconDotNet* permette di utilizzare le classi di Halcon, mentre *System.Runtime.InteropServices* permetterà l'utilizzo del metodo *Marshal.Copy()* necessario per poter copiare le immagini da passare a Matlab. La funzione che verrà richiamata da Matlab sarà soltanto *FromHalcon()* (figura 2.7) questa funzione non fa altro che richiamare le elaborazioni esportate tramite il metodo *action()* generato automaticamente dal tool di esportazione di Halcon. Quello che si vuole infatti è partire dall'export di Halcon e incorporarlo in una libreria con questa struttura. Per fare ciò bisogna modificare l'intestazione del metodo *action()* aggiungendo l'attributo *static* e come output vogliamo che restituisca un oggetto della struttura di supporto *results*. Questa contiene al suo interno sia dei campi *HObject, HTuple* per poter estrarre i valori desiderati da *action()* sia dei normali array in formato byte necessari per poi essere restituiti a Matlab. Per prima cosa bisogna andare a modificare *action()* salvando esplicitamente in Output tutte le immagini o valori desiderati, per fare questo la cosa più semplice e conveniente è usare soltanto metodi Halcon, così da lavorare con HTuple e HObject soltanto, per poi rimandare ad un unica conversione finale dei risultati. Queste classi comportano notevoli vantaggi ad esempio all'interno della stessa variabile un HObject può contenere un numero qualsiasi di immagini/regioni appunto le *iconic varibles*, di fatto si può decidere di racchiudere all'interno di una sola variabile tutte le immagini da esportare non basta altro che assegnarle alla fine di *action()* dentro ad Output. Per quanto riguarda le HTuple si può fare un discorso analogo anche se spesso è necessario esportare più array, per risolvere a questo problema si possono aggiungere diversi output quindi creare più oggetti di tipo *results*, altrimenti aggiungere campi HTuple all'interno di della struttura *results*. Una volta ottenuto l'output bisogna convertire i risultati prima di restituirli a Matlab, per fare ciò sono state introdotti due metodi:

- **HalconObject2MatlabArray :** questo metodo permette di convertire un HObject in un array di matrici di byte Compatibile con Matlab. Infatti la procedura non fa altro che selezionare ogni figura al suo interno e richiama *HalconImage2MatlabArray()*, il quale si occupa della conversione della singola immagine. Quest'ultimo metodo di basa sull'operatore di Halcon *GetImagePointer1*, il quale restituisce un puntatore in formato INTPTR

```
% importazione assembly
import ClassImportHalcon.dll.*

% Richiamo metodo C#
Risultati_Halcon =ClassImportHalcon.ExportTupleImages.FromHalcon();

%display risultati
result = uint8(Risultati_Halcon.imout);
firstFigure = reshape(result(1,:,:),480,640);
imshow((uint8(Risultati_Halcon.imout(1))));
```

Figura 2.7: Struttura codice Matlab per l'importazione della classe C#.

dove è fisicamente collocata l'immagine nella memoria; viene quindi effettuata una copia dell'immagine tramite *Marshal.Copy()* andando a creare un doppio array di byte riempito per righe, le quali corrispondono effettivamente alle righe dell'immagine desiderata. In questo modo si ottiene la stessa immagine soltanto all'interno di un doppio array di byte compatibile con Matlab. Questa viene poi effettuata per tutte le immagini, mettendole all'interno di un triplo array di byte.

- **HalconTuple2MatlabArray :** Questo semplice metodo non fa altro che ispezionare ad uno a uno gli elementi all'interno della HTupla tramite il comando *tuple_select*. Dopodiché ogni elemento viene eseguito un cast in double in modo da poter essere esportato in un oggetto DOUBLE, questo per poi essere letto da matlab tramite un semplice cast nativo ad esempio *double()*.

A questo punto non resta che richiamare direttamente il metodo da Matlab, per fare ciò come mostrato in fig. 2.7, è necessario importare il .dll creato. Richiamando quindi classe e metodo, verranno ottenuti i risultati e caricati nel workspace. Per visualizzare correttamente un immagine bisogna fare un cast, quindi specificando che il formato dei valori all'interno delle matrici sia in UINT8. Non resta che eseguire un reshape. Si usa quindi la funzione reshape di *Matlab* a causa della particolare efficienza di *Matlab* nell'usare formati di tipo matriciali.

2.3.2 Mex

Un file Mex permette di chiamare le proprie subroutine C, C++ o FORTRAN da MATLAB® dalla comand line come se fossero funzioni incorporate. Questi programmi, chiamati file binari MEX, sono subroutine collegate dinamicamente che l'interprete MATLAB carica ed esegue. L'obbiettivo quindi è quello di creare un Mex in grado di richiamare una subroutine che esegue il runtime di Halcon. Per creare il Mex si è deciso di utilizzare il C++ come linguaggio per creare la libreria. Per poter compilare correttamente un programma che deve essere un Mex, e contemporaneamente usare le funzioni di Halcon come fosse una libreria bisogna includere diverse directory nella sezione. Per poter utilizzare Visual studio per creare il Mex, bisogna andare a realizzare una libreria dinamica (.dll) con una estensione di destinazione di tipo .mexw64. Nella sezione "proprietà/ C/C++" è possibile selezionare le directory di inclusione, nello specifico servono $(HALCONROOT)$\include\halconcpp; $(HALCONROOT)$\include; $(MATLAB)$\extern\include . Per quanto riguarda la configurazione del linker devono essere aggiunte le Directory librerie:
$(HALCONROOT)$\lib\$(HALCONARCH)$; $(MATLAB)$\lib\win64\microsoft\.
E aggiungere nella sezione input le dipendenze aggiuntive:
halcon.lib;halconcpp.lib;libmx.lib;libmex.lib;libmat.lib. Infine nella riga di comando come opzione aggiuntiva bisogna dare il comando "\export:mexFunction" in modo da creare il .mexw64 a fine compilazione.
Per poter creare allora la funzione mex bisogna includere i seguenti files di intestazione all'inizio del programma: "mex.h","HalconCpp.h","HDevThread.h". Una volta che viene richiamato da Matlab un mex file, viene eseguito il codice all'interno della mexFunction a modo di una qualsiasi funzione *main*. Questo particolare tipo di funzioni permettono, oltre che ad andare a dare in uscita a matlab i risultati voluti, la possibilità di passare parametri da poter utilizzare all'interno del programma C++ e di conseguenza è possibile modificare vari parametri delle elaborazioni fatte tramite Halcon direttamente da Matlab senza dover esportare nuovo codice. È stato sviluppato un header "HalconHelper.h" con tutti i metodi necessari a convertire immagini, regioni e Htuple sia in uscita che in entrata a *Matlab*. La struttura tipica per una semplice mexFunction viene descritta in fi-

```
//Mex function (main)

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]) {

    HTuple  *inputs_Matlab = new HTuple();                  //parametri importati da Matlab
    HObject *Image_input = new HObject();                   //immagine/regione dove salvare i risultati
    HObject *Image_Output = new HObject();

    MatlabArray2HalconTuple(prhs[0], inputs_Matlab);   // importo  i dati che verranno convertiti in una HTupla
    MatlabArray2HalconObject(prhs[1], Image_input);     // per eventuali input Images

    //normale main da esportazione
    try
    {
        static HTuple  Output1, Output2;        //variabili Output

        action(inputs_Matlab, Image_input, Image_Output, &Output1, &Output2);  //richiamo Halcon action

        //converto i risultati e gli esporto in Matlab
        HalconObject2MatlabArray(Image_Output, &plhs[0]);       //converto gli HObject e gli esporto in Matlab

        HalconTuple2MatlabArray(&Output1, &plhs[1]);            //converto le HTuple e le esporto in Matlab
        HalconTuple2MatlabArray(&Output2, &plhs[2]);

    }
    catch (HException &exception)           //gestione eccezioni Halcon
    {
        mexPrintf( "  Error #%u in %s: %s\n", exception.ErrorCode(),    //Usare mexPrintf per codice errore
            (const char *)exception.ProcName(),
            (const char *)exception.ErrorMessage());

    }

    FinalizeHALCONLibrary();            //serve per rilasciare la memoria usando Halcon come libreria
}
```

Figura 2.8: Esempio di MexFunction per l'esportazione dei risultati Halcon in Matlab.

gura 2.8. Nella quale si può notare come diventi semplice impacchettare l'action ottenuto da Halcon inizialmente si convertono i valori importati da Matlab, se presenti, passati all'action dove deve essere modificata l'intestazione, in modo da potergli passare le variabili statiche e vari riferimenti di input e output. Una volta che finisce l'elaborazione non resta che convertire i risultati ottenuti e passargli in uscita della mexfunction.

Nella soluzione adottata, il modo più semplice per passare un numero qualsiasi di parametri di input è quello di passare alla mexFunction una variabile cella contenente vettori e stringhe. Non è possibile passare con i metodi sviluppati matrici oppure struct. Un'altra operazione che bisogna andare ad eseguire nell'action() è quella di andare selezionare e salvare in una nuova variabile ogni elemento ottenuto dalla cella di input attraverso l'operatore *TupleSelect()*, in questo modo si possono sfruttare i metodi halcon con dei nuovi parametri richiamati da *Matlab*.

I principali metodi di input implementati sono i seguenti:

- **MatlabArray2HalconTuple:** questo metodo permette di convertire una qualsiasi variabile di tipo vettore, stringa o cella in un *HTupla*, il metodo si basa sul *mxGetData* della classe mex.

- **MatlabArray2HalconImage:** questo metodo converte una matrice 2D contenente un'immagine in un *HObject*. Il metodo implementato funziona sia per immagini grayscale sia a colori. Questo metodo si basa sul comando di Halcon GenImage1 per immagini grayscale, mentre si basa su GenImage3 per immagini a colori. Sono inoltre gestiti tutti i possibili data type supportati da Halcon per il formato dei bit immagine.

Mentre per quanto riguarda invece le esportazioni dei risultati da parte di *Halcon* i principali metodi implementati sono i seguenti:

- **HalconTuple2MatlabArray:** Questo metodo permette l'esportazione dei risultati ottenuti dal metodo *action()* in una matrice oppure in una cella a seconda del tipo di dato in ingresso. Questo metodo è basato sul metodo *mxCreateNumericArray* oppure *mxCreateCellArray* della classe mex a seconda del tipo di dati.

- **HalconObject2MatlabArray:** Questo metodo è in grado di esportare direttamente qualsiasi tipo di iconic variables anche per immagini a colori, infatti converte array sia contente *HImage, sia HRegion*. Questo metodo si basa sul metodo *memcopy* il quale va a copiare un blocco di memoria in un altro, questo una volta ottenuto e convertito il puntatore dell'immagine di Halcon tramite *getImagePointer1*, dopo aver calcolato la dimensione dell'immagine in termini di byte ed eseguito un doppio cast del puntatore (prima *HLong*, poi in *void**), copia direttamente il blocco dell'immagine nell'uscita creata.

Come si può vedere in figura 2.8, al termine della mexFunction è stata richiamata la funzione *FinalizeHALCONLibrary()*. Questo metodo permette di liberare correttamente la memoria al termine dell'esecuzione della funzione richiamata

da *Matlab*. Se questa non viene usata al termine del programma saranno ancora disponibili le variabili di *Halcon* per elaborazioni successive. Questo può essere utile ad esempio se si vuole aprire solamente una volta il framegrabber, oppure per evitare comandi ripetitivi. Se la memoria però non è correttamente liberata a fine esecuzione, oppure una volta richiamato l'operatore *clear*, necessario per liberare la memoria del Mex per nuove operazioni, avviene un conflitto di librerie facendo andare Matlab in crash. A questo punto è possibile richiamare la funzione sviluppata in *Matlab* e per visualizzare correttamente un immagine in Matlab è necessario eseguire un reshape e trasporla. Mentre i valori saranno correttamente importati in matrici oppure celle in maniera automatica.

2.4 Casi di utilizzo

Una volta forniti gli strumenti per poter esportare rapidamente i risultati ottenuti da Halcon, bisogna andare a specificare quali siano i più casi in cui un metodo risulta più vantaggioso di un altro.

- **Halcon stand-alone su HDD:** Se si vuole la semplicità in fase di progettazione, ed essere facilmente compatibili con gli standard più usati questo è il metodo più indicato, in quanto non necessita da parte dell'operatore mettere le mani sul codice. In particolare nel contesto in cui si vuole mantenere la comand line di *Matlab* libera e nel frattempo avere una applicazione che gira su un eseguibile in maniera continuativa (caso di acquisizione su nastro) questo metodo è il più indicato. In termini di prestazioni, se utilizzato con l'ausilio di un Ram-Disk, sono abbastanza buone e una analisi approfondita dei tempi di esecuzione si trova nel prossimo capito di test. Questo metodo dato il salvataggio di informazioni tramite formati standard può essere usato anche da applicativi diversi da Matlab facilmente.

- **Socket:** Questo metodo è quello in cui bisogna maggiormente andare a modificare il codice esportato da *Halcon* nonostante i metodi sviluppati. Questo tipo di trasferimento è consigliato usarlo solamente nel caso in cui è necessaria la comunicazione tramite socket, ad esempio se le esecuzioni dei

programmi di visione sono affidate ad un server, oppure se si vuole usare un altro programma che non sia Matlab, nel caso in cui non sia possibile il trasferimento tramite HDD. Questo metodo risulterà anche il più lento dal punto di vista del trasferimento.

- **Classe C#:** Questo tipo di soluzione permette di avere il controllo dei tempi di esecuzione del codice dato l'utilizzo tramite libreria. Questo metodo è consigliato nel caso in cui si ha a che fare con l'ambiente di sviluppo .NET, in modo da poter collegare diversi dispositivi condividendone il linguaggio (ad esempio con dispositivi PC embedded). Nella soluzione proposta è possibile esportare solamente immagini in grayscale, mentre come tempi è tra i migliori.

- **Mex:** Nel momento in cui si voglia utilizzare *Halcon* come libreria, l'utilizzo tramite mex è il sistema risultato migliore. Questo permette di ottenere le migliori performance in termini di velocità per il trasferimento, inoltre tramite gli strumenti forniti, risulta essere molto semplice andare a modificare il codice per realizzare la mexFunction. Infatti è possibile trasferire valori sia in input sia output, cosa non direttamente realizzata con altri modi. Sono inoltre supportati tutti i diversi tipi di dati sia in termini di *HTuple* che *HObject*.

Capitolo 3

Prove Sperimentali

In questo capitolo verranno descritte le prove sperimentali affrontate per testare i metodi e le classi sviluppate. Si eseguono infine altri test per andare ad approfondire l'utilizzo di *Halcon* testandone i metodi principali in ambito di matching, questo in ottica di risolvere alcuni task tipici in ambienti industriali.

3.1 Analisi tempi per i metodi di trasferimento

Il primo test che andiamo ad analizzare sono i tempi di trasferimento dei risultati dei metodi illustrati nel capitolo 2.

Per affrontare questi test è stata usata la telecamera guppy f201\b della allied vision. Attraverso il suo software vimba viewer è possibile impostare i parametri di acquisizione, in questo test è stato fissato il tempo di esposizione a 30 ms, gli fps calcolati dal vimba viewer riusltano essere 29.16 *fps* in questo modo. È stata inoltre fissata una ROI di 640× 480 e metodo di acquisendo in grayscale. Per ese-

	Media [ms]	Deviazione standard [ms]
Trasferimento HDD	282.4	12.4
Socket	508.7	10.0
C#	138.6	21.6
Mex	92.4	0.81

Tabella 3.1: Media e deviazione standard per il trasferimento di 100 immagini.

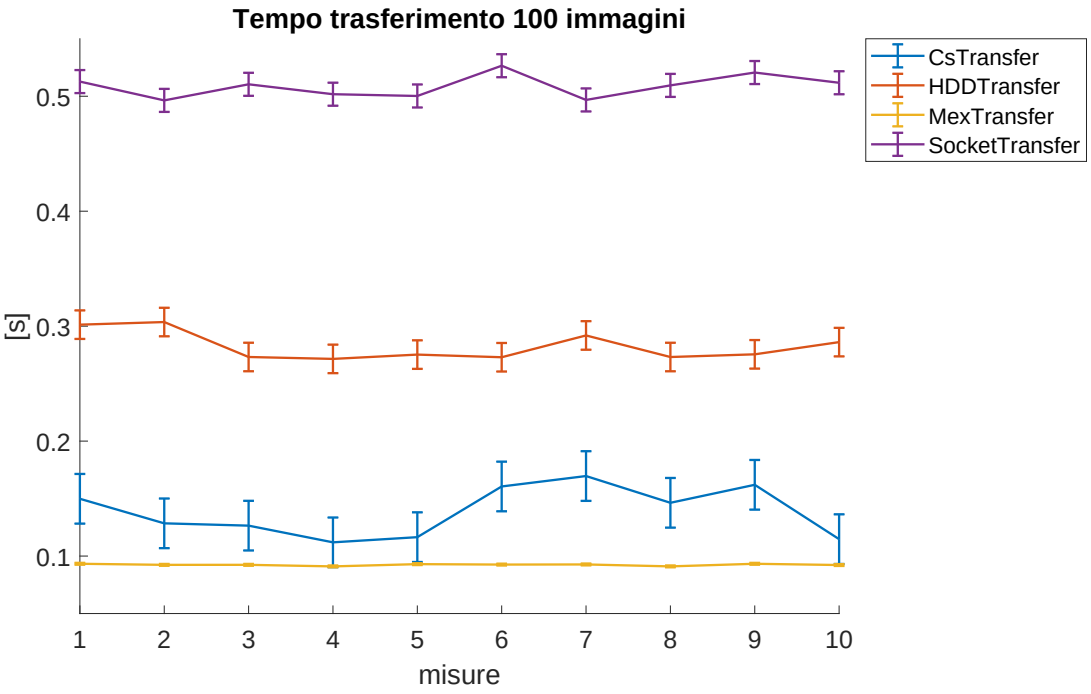

Figura 3.1: Confronto metodi di trasferimento di 100 immagini ottenute da Halcon ed esportate in Matlab.

guire il test è stato scelto di acquisire cento immagini tramite Halcon e passarle con i metodi sviluppati a Matlab. Questa operazione è stata ripetuta per 10 volte in modo da andare a confrontare i tempi attraverso media e deviazione standard, come mostrato nella tabella 3.1. I risultati possono essere osservati in figura 3.1. Ogni misura rappresenta il tempo necessario al solo trasferimento dei risultati una volta calcolati dalla procedura, quindi è stato eliminato il tempo di acquisizione. Dal grafico riassuntivo è possibile notare come li metodo migliore risulti essere il trasferimento tramite Mex. I tempi sono stati calcolati tramite l'operatore "tic-toc" di Matlab a cui è stato tolto il tempo di esecuzione del metodo action eseguito dal richiamo del metodo da parte di Matlab. In questo modo, vengono quindi analizzati solamente i tempi di trasferimento, questi risultano avere una media di 0.092 s e una deviazione standard di appena 0.81 ms. Rendendo di fatto questo metodo il più veloce e allo stesso tempo più preciso di quelli confrontati. Come si può osservare dal grafico il metodo che ottiene le prestazioni peggiori è il trasferimento tramite Socket. Per calcolare i tempi sono stati considerati

solamente i tempi di trasmissione e ricezione quindi non vi è stata considerata tutta la parte di acquisizione, ma solo l'importazione in *Matlab* incluso il tempo di reshape delle immagini dato che vengono inviati 100 array monodimensionali, contenti ciascuno un'immagine, per ricomporre la larghezza e l'altezza immagine. Il tempo medio quindi è di 0.508 s, all'incirca 5 volte maggiore rispetto il metodo migliore tramite Mex che con una media di 0.09 s e con la deviazione standard minore rimane il più affidabile e veloce. Per quanto riguarda il salvataggio tramite Ram Disk si ottengono dei buoni risultati con una media di 0.282 s, circa tre volte maggiore rispetto al metodo Mex. Questo deve comunque essere preso come riferimento una volta in cui si vuole affrontare il problema e non si vogliono ottenere particolari performance, come già introdotto nel precedente capitolo, è di gran lunga il più semplice e spesso preferibile non essendo costretti a modificare il codice. Per quanto riguarda il trasferimento tramite C# si ottengono nel complesso delle buone prestazioni con una media di 0.138 s solamente la versione Mex risulta più veloce, per quanto riguarda invece la precisione è quella che risulta più volatile con una deviazione standard di 21.6 ms. Valgono quindi le considerazioni effettuate nel secondo capitolo, nel caso si stessero utilizzando sistemi .Net è una buona alternativa al semplice trasferimento su HDD, in quanto è possibile combinare più elementi che condividono lo stesso linguaggio.

3.2 Elaborazione batteria alettata

Un esempio in cui è stato usato il metodo di trasferimento tramite Mex è l'elaborazione della "batteria alettata" mostrata in figura 3.2. Questa elaborazione è pensata nel contesto di assemblaggio dello scambiatore di calore, nello specifico un robot è adibito al piazzamento delle "curvette" dentro ad una coppia di fori pre-determinata. L'obbiettivo della visione in questo caso diventa quindi di andare ad identificare la posizione dei fori e il riconoscimento delle curvette precedentemente allocate. Il problema principale è la precisione necessaria per il piazzamento tramite robot, in particolare nel riconoscimento dei fori il bicchiere in cui deve essere piazzata la curvetta è molto irregolare, di conseguenza è necessario trovare il modo di distinguere il foro dal bicchiere. Inoltre è importante poi

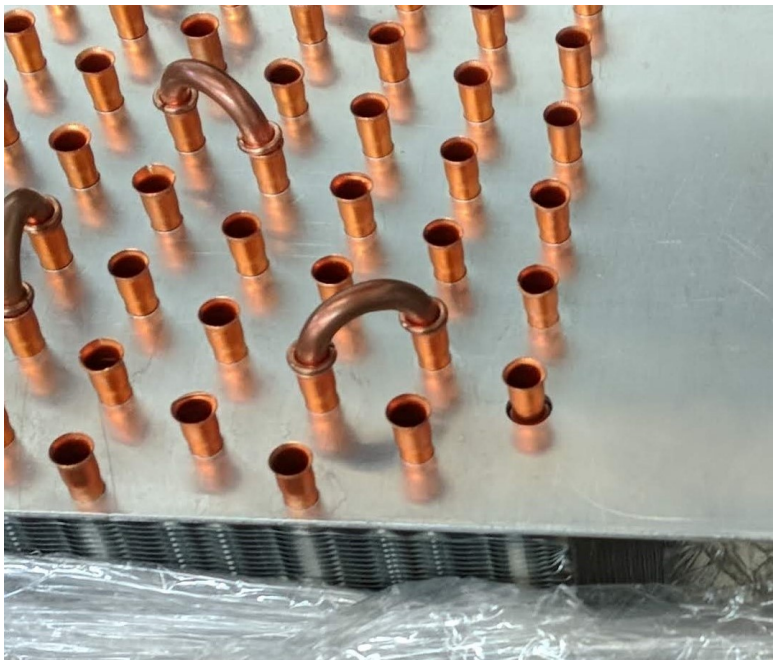

Figura 3.2: Particolare della batteria alettata raffigurante fori e curvette.

passare le coordinate a Matlab in modo da poter confrontare i fori con il CAD di riferimento, in modo da utilizzare il baricentro della curvetta come coordinata da cui trovare la coppia di fori occupati in questo modo.

3.2.1 Soluzione adottata

Come setup di lavoro è stata utilizzata una telecamera gupy f201/b con un ottica di 25mm, questa telecamera è stata piazzata a 1.1 metri di altezza rispetto al piano dei fori in modo da diminuire il più possibile la distorsione. Anche l'ottica è stata scelta in modo da cercare di avere la distorsione minore possibile andando a inquadrare da lontano una regione stretta ma comprensiva di tutta la scena, attraverso l'uso appunto di un 25 mm. L'intero problema era stato precedentemente risolto tramite l'ausilio di 2 foto una acquisita tramite illuminazione "dark-field" per andare a esaltare la differenza tra foro e bicchiere, l'altra invece tramite "bright-field" in modo da rendere le curvette riconoscibili. Un primo passo per migliorare il lavoro precedente è quello di usare una singola foto per completare l'elaborazione, andando quindi a evitare il costo di sistemi di illuminazione supplementare, non dovendo eseguire 2 acquisizioni per la stesso task.

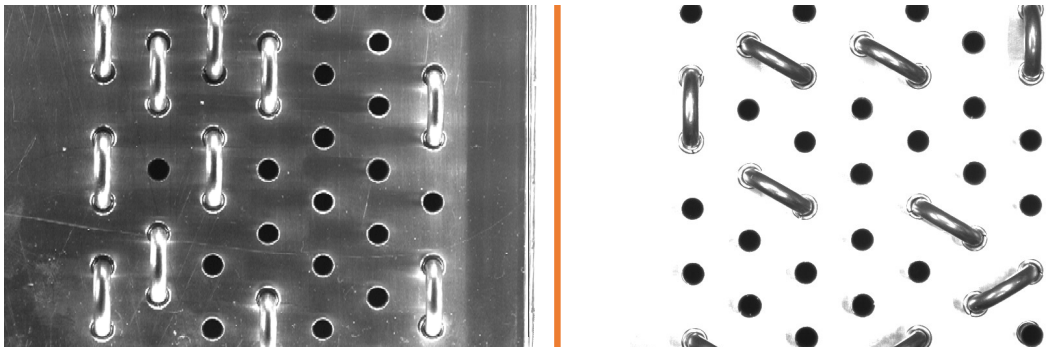

Figura 3.3: A sinistra è possibile vedere un immagine ottenuta tramite dark-field. A destra e mostrata un immagine ottenuta tramite bright-field dello stesso oggetto.

Per prima cosa si è studiato il metodo di illuminazione da usare per risolvere il problema, in figura 3.3 è mostrato sia l'approccio "dark-field" sia "bright-field". Il primo mettendo dei LED in maniera parallela al piano della batteria, il secondo con l'illuminazione diffusa dall'alto. Per quanto riguarda il primo tipo di illuminazione è stato scartato per 2 motivi: il primo riguarda i riflessi sulle curvette, i quali rendono molto complesso il loro riconoscimento, il secondo è dovuto alle alette che in alcune batteria è disposta lungo tutti e quattro i bordi, andando a ridurre la qualità dell'illuminazione data la creazione di ombre. Come si può vedere in figura 3.3 dalla seconda immagine diventa difficile andare a distinguere il bicchiere dal foro per risolvere questo problema è stato applicato un filtro derivativo in modo da andare ad analizzare il gradiente di intensità dei pixel e non il valore di grigio. Il risultato ottenuto è visibile in figura 3.4, nella quale risultano ben evidenti i contorni dei bicchieri non distinguibili dall'immagine di partenza. In questo modo si è andati a sfruttare l'operatore di treshold non sull'immagine di partenza ma da quella derivata così da considerare foro i pixel in cui il valore di derivata era basso mentre quando il gradiente saliva si andavano a definire come bicchieri. Una volta ottenute le regioni in pixel dei fori andando ad analizzare area e il parametro circolarità, si è eseguito un fit del cerchio che meglio rappresentava il foro in modo da aumentarne la precisione.

Per quanto riguarda il metodo per trovare le curvette si è deciso di utilizzare lo shape-based-matching proprio a causa del filtraggio precedentemente eseguito.

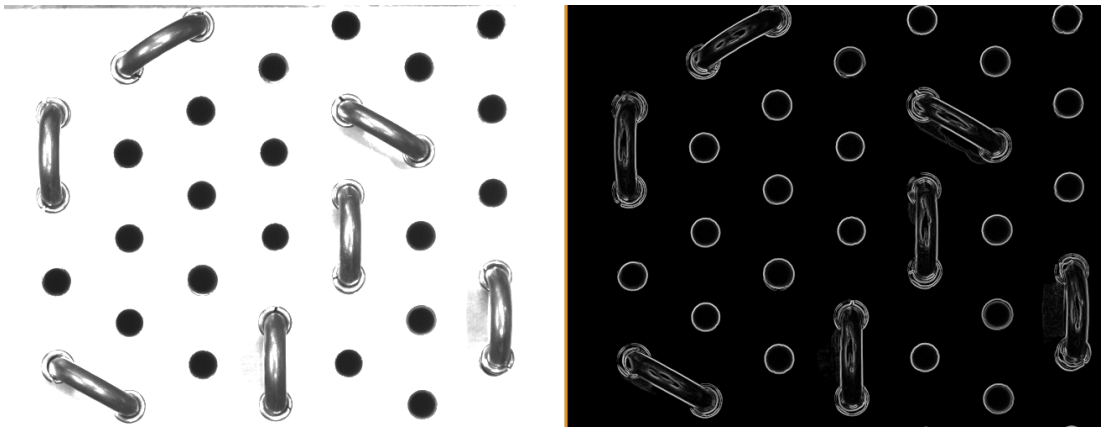

Figura 3.4: A sinistra è rappresentata l'immagine originale, a destra l'immagine ottenuta tramite l'operatore di segmentazione *derivate_gauss*.

Si è quindi andati a definire un modello isolando la singola curvetta, creato il modello attraverso il create_shape_model. Una volta settati tutti i parametri si è riuscito a risolvere il problema. È possibile osservare i risultati visivi in figura 3.5.

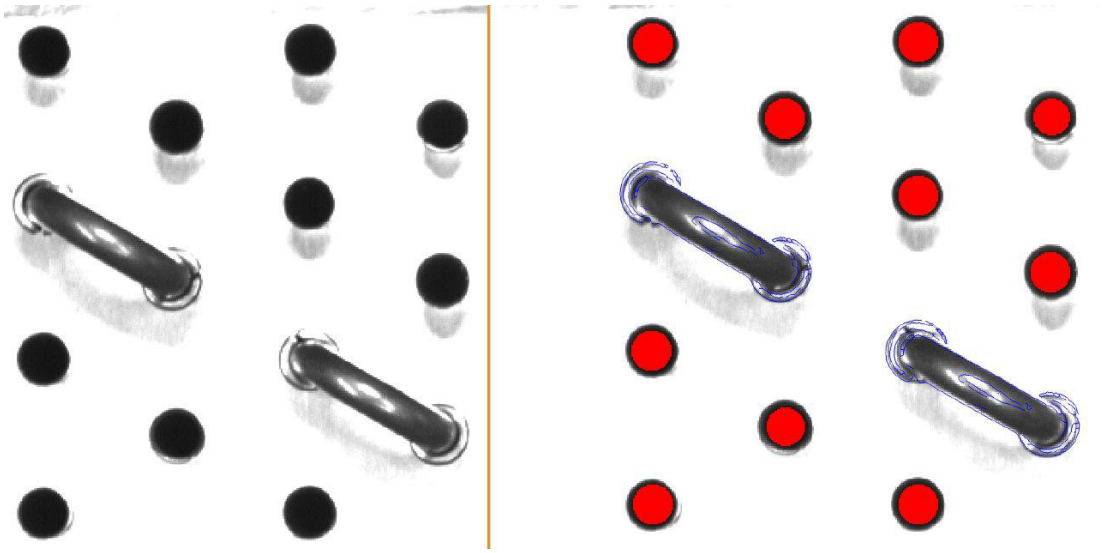

Figura 3.5: Risultato ottenuto tramite la procedura in cui sono evidenziati in rosso i fori mentre in blu i contorni delle curvette.

	Media [ms]	Deviazione standard [ms]
Totale	120	3.5
Acquisizione	83.3	3.5
Elaborazione	36.8	0.43

Tabella 3.2: Tempo elaborazione batteria alettata

3.2.2 Analisi tempi

Sono stati analizzati i tempi di esecuzione dell'intera procedura, in particolare sono stati eseguiti 100 test per il riconoscimento di entrambi gli oggetti. Per quanto riguarda i tempi totali di acquisizione ed elaborazione come si può vedere in tabella 3.2 sono in media di 120 ms con una deviazione standard di 3.5 ms. Per quanto riguarda la sola fase di acquisizione è possibile vedere dal grafico 3.6 come sia l'operazione più onerosa dal punto di vista temporale, infatti il valor medio per acquisizione è di 83.3 ms con una deviazione standard di 3.5 ms. Di conseguenza in media l'acquisizione in questa applicazione impiega l'69.37% del tempo, andare quindi a ridurre il numero di immagini acquisite da 2 a 1 porta un notevole vantaggio dal punto di vista temporale dato che sono limiti hardware non direttamente migliorabili. Per quanto riguarda il tempo di elaborazione invece il valor medio è di 36.8 ms con una deviazione standard di 0.43 ms, di conseguenza il tempo di elaborazione ottenuto impiega il restante 30.63% del tempo totale, risultato da considerarsi positivo. È stato quindi raggiunto l'obiettivo di eliminare un operazione costosa come un ulteriore acquisizione e sfruttare i metodi Halcon in modo da minimizzare i tempi per risolvere questo problema mantenendo un buon livello di precisione. Il risultato è stato quindi trasferito a Matlab in modo da poter completare ulteriori operazioni una volta ottenute le informazioni trovate, questo è un tipico utilizzo ad esempio di utilizzo di un mex infatti sono stati eseguiti 100 test di trasferimento delle coordinate ottenute e il tempo impiegato è in media di 3.2 ms con una deviazione standard di 0.01 ms.

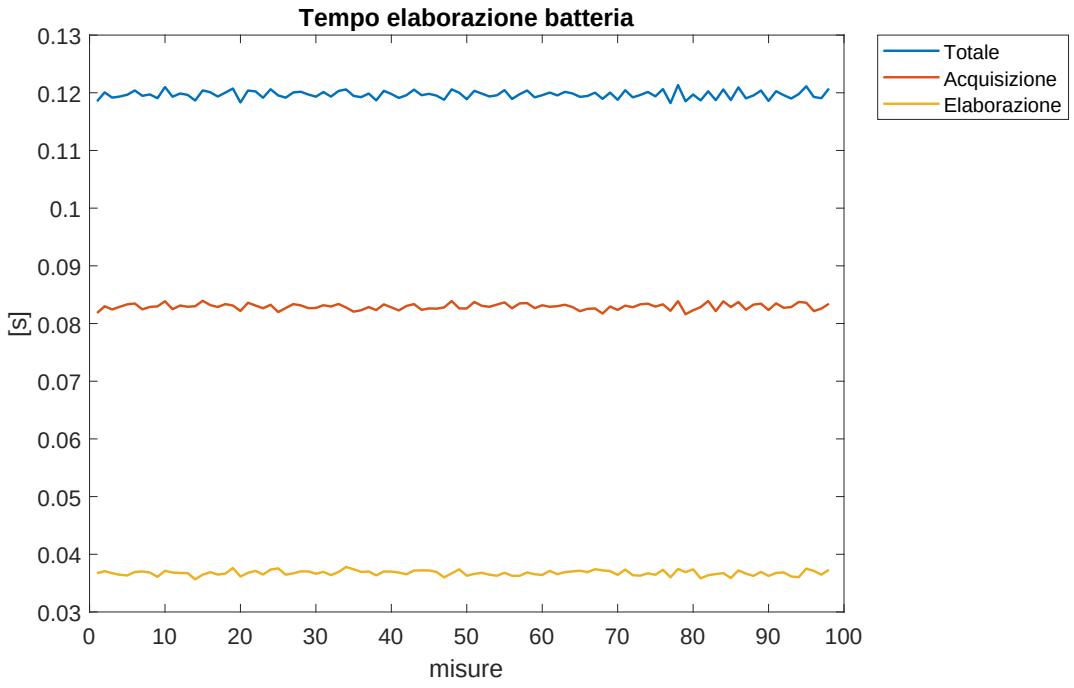

Figura 3.6: Confronto tempi di acquisizione immagine ed elaborazione della stessa per la batteria alettata.

3.3 Applicazione pick and place

Uno dei problemi affrontati è stato implementare la parte di visione in un'isola robotizzata. In particolare si vuole mostrare come Halcon possa essere utilizzato tramite gli strumenti sviluppati in una applicazione composta da più elementi. Per risolvere questo problema è stata implementata una GUI in Matlab tramite Guide, in modo da aver uno strumento in più in mano ad uno sviluppatore senza conoscere Halcon ma riuscirne ad usare gli algoritmi di Matching. Viene poi effettuato un confronto in termini di velocità tra i 2 tipi di matching più usati lo shape-based-matching e il tamplate-matching. In questo lavoro sarà approfondito soltanto il lato che riguarda la parte di visione i vantaggi portati dagli strumenti sviluppati.

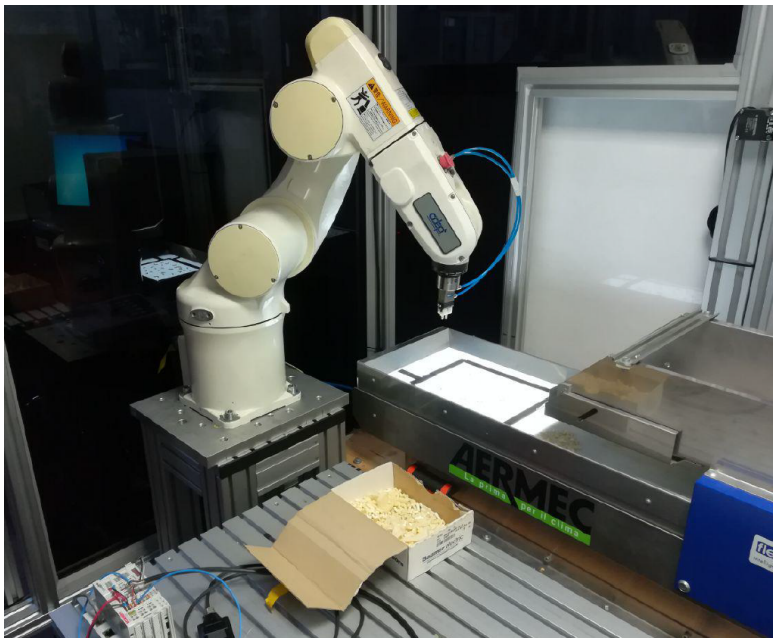

Figura 3.7: Isola robotizzata utilizzata durante i test.

3.3.1 Setup

Il sistema di visione è stato utilizzato all'interno di una cella di lavoro nel laboratorio di robotica dell'università di Padova. Dove precedentemente era stato sviluppato un layout contenente i componenti per una applicazione di pick and place, nello specifico un robot e un feeder. Un immagine dell'interno della cella di lavoro è presente in figura 3.7, nella quale è possibile notare il robot un l'*Adept Viper 650*[3] dell'azienda *Omron*. Questo particolare manipolatore è un antropomorfo[4] a 6 gradi di libertà ad alte prestazioni, adatto per applicazioni di precisione. Mentre il feeder usato è di tipo flessibile, *Anyfeeder* prodotto dalla *omron*, è il componente che in grado di garantire l'approvvigionamento continuo di pezzi sfusi di piccole dimensioni, come le viti, al robot. Dal piano di questo retroilluminato da un LED sono presenti i pezzi in cui il robot deve andare in presa. Uno dei problemi principali in questo tipo di applicazione, in cui i pezzi sono messi alla rinfusa sul piano del feeder, è fornire le coordinate in cui il robot troverà il pezzo da raccogliere. Per ottenere una buona precisione per pezzi di piccole dimensioni, il robot è stato precedentemente calibrato tramite l'ausilio di un tastatore. Nasce quindi il problema di eseguire un task utilizzando diver-

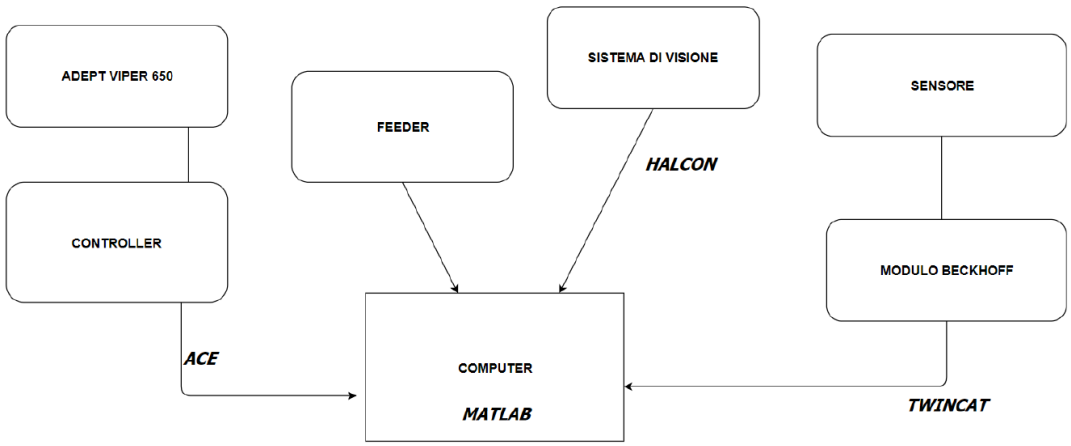

Figura 3.8: Mappa programmi utilizzati.

si programmi e hardware che necessitano di comunicare tra loro. Per risolvere questo problema è stato utilizzato Matlab per poter comunicare e condividere informazioni con gli altri software. La mappa che mostra le dipendenze tra questi è mostrata in figura 3.8. Nella quale è possibile notare come Matlab sia usato in modo da poter comunicare tramite Twincat di beckhoff con il tastatore in modo da andare a fornire una calibrazione accurata e utilizzare direttamente i risultati in Matlab. Dall'immagine si nota inoltre come Matlab sia in grado di comunicare con il software ACE per inviare e ricevere coordinate al robot, questo sfruttando una comunicazione tramite socket, inoltre è possibile andare a comandare il feeder direttamente con delle apposite funzioni precedentemente sviluppate. Infine per generare le coordinate con cui andare in presa col robot è stato aggiunto il sistema di visione tramite Halcon. Per i test sperimentali di presa pezzo si sono usati oggetti di piccoli dimensioni in plastica stampati in 3D di 4 tipologie diverse, mostrate in figura 3.9. I pezzi utilizzati hanno uno spessore di qualche millimetro di conseguenza è importante essere accurati nel movimento del robot per riuscire nella presa. In questo gioca un importante ruolo il sistema di visione in grado di ottenere dei buoni risultati attraverso la selezione di componenti quali telecamere e ottiche in grado di ottenere una buona immagine. La telecamera per la cella in questione è una pike F-505[5] dell'azienda Allied Vision, è una telecamera con una risoluzione di 5 megapixels a cui è stata aggiunta un ottica da 35 mm per riuscire ad inquadrare bene la scena ed evitare distorsione. Questa è stata

Figura 3.9: Pezzi utilizzati nei test di guida robot.

Figura 3.10: Telecamera Pike utilizzata nei test di guida robot.

messa perpendicolare al piano del feeder in modo da poter vedere i pezzi senza distorsione.

3.3.2 GUI

È stata sviluppata una GUI in grado di semplificare le operazioni necessarie per risolvere il problema appena discusso. In particolare la GUI rappresentata in figura 3.11 è stata usata per utilizzare diverse funzioni, tra cui la possibilità di salvare in una struttura di matlab la telecamera da selezionare, impostazioni e auto acquisizione di un modello, impostazioni e preview del metodo find. La GUI è stata pensata in modo da risolvere un completo problema di matching, è possibile infatti eseguire un auto-detect delle telecamere connesse, una volta selezionato il tasto

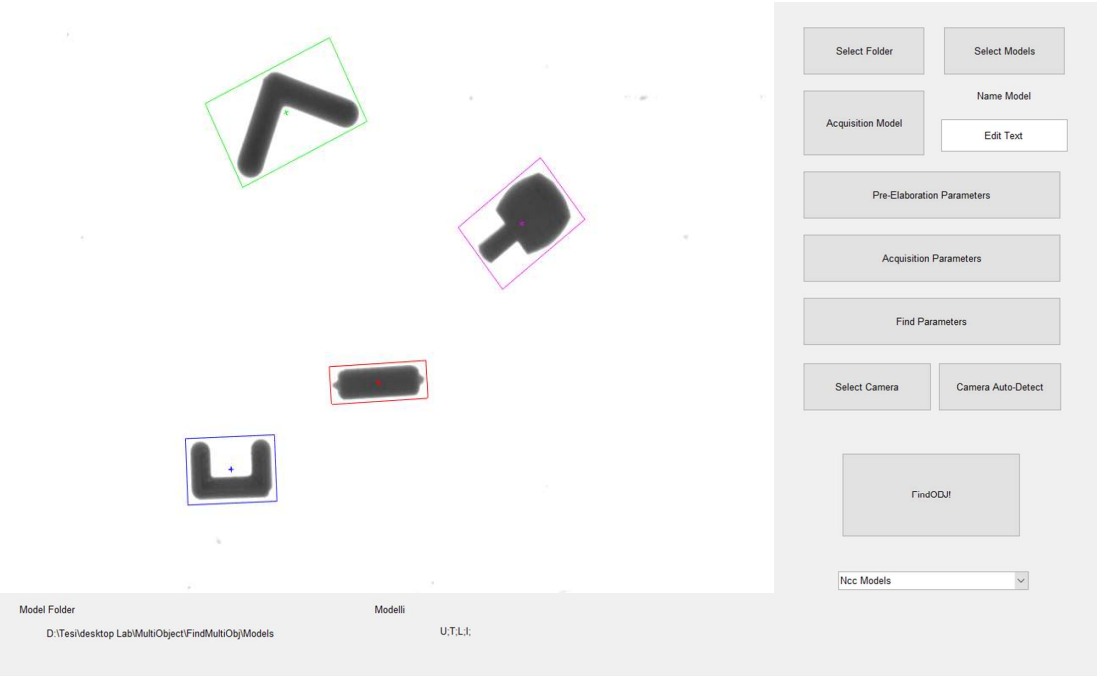

Figura 3.11: GUI realizzata per risolvere il problema di guida robot.

auto-detect compare un'altra schermata simile a quella mostrata in figura 3.12. Questo è possibile attraverso il comando *Info_frameGrabber* di Halcon, il quale permette di ottenere le informazioni quali interfacce e model ID delle telecamere connesse. Una volta ottenute queste informazioni in formato striga sono state poi elaborate in modo da estrarre soltanto le parti necessarie all'identificazione della telecamera. Grazie a questa GUI è quindi possibile andare a definire un oggetto telecamera all'interno di Matlab da poter andare a selezionare in maniera semplice. Una volta selezionata la telecamera è necessario impostare i parametri della pre-elaborazione, questo è possibile farlo direttamente da Matlab attraverso la finestra creata andando a definire i pre-elaboration parameters. Questa finestra è possibile vederla in figura 3.13. In questa è possibile effettuare le operazioni di filtraggio attraverso l'uso di un filtro mediano, aumentare il contrasto e fissare una treshold. In particolare per tutti questi parametri è possibile definirne i parametri e nel caso della treshold si può decidere se utilizzare una manuale oppure una automatica calcolata attraverso l'algoritmo di OTSU. Inoltre è possibile andare a vedere direttamente se questi parametri soddisfano le esigenza attraverso il comando preview. Una volta definiti tutti questi parametri è possibile scegliere,

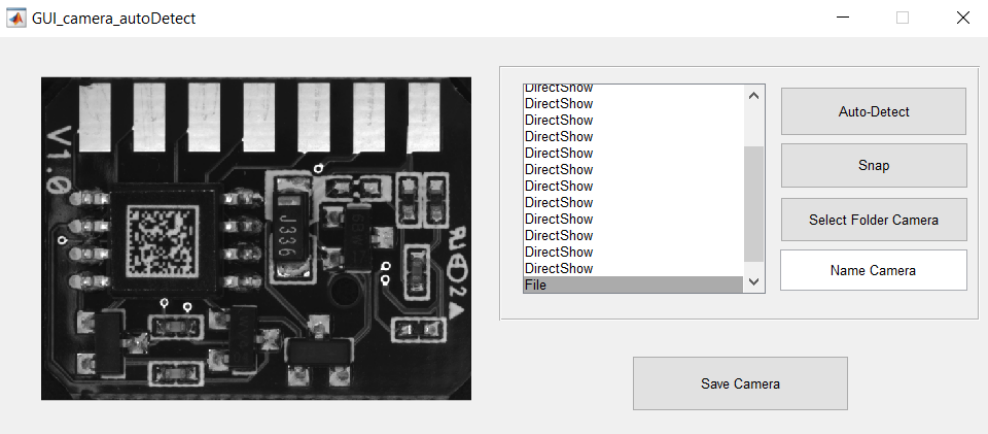

Figura 3.12: Schermata di auto-detect della GUI realizzata, in cui è possibile vedere direttamente la preview.

attraverso la schermata principale della GUI(figura 3.11), se utilizzare un matching di tipo shape-based oppure template-based. Una volta selezionato questo aspetto è possibile creare direttamente un modello dando un codice e salvandolo in una specifica cartella per le future applicazioni. Il metodo creato si basa rispettivamente sui comandi create-shape model oppure create-ncc-model, inoltre è implementata una funzione di auto-ROI in modo da creare un buon modello direttamente senza andare a definirla a mano. Per ottenere un buon risultato è fondamentale che la pre-elaborazioni vada a isolare l'oggetto desiderato infatti la funzione auto-ROI si basa sui valori ottenuti dalla treshold impostata. È possibile poi impostare tutti i principali parametri per la fase di acquisizione del modello e quella relativa alla procedura *find*. Una volta quindi selezionata la telecamera, i modelli da trovare e i parametri è possibile osservare i risultati direttamente dalla GUI in modo da avere un feedback visivo immediato. I modelli creati per risolvere questo problema sono stati salvati e poi utilizzati per andare a cercare gli oggetti che il robot dovrà andare ad afferrare.

3.3.3 Confronto Shape-based/template matching

Per risolvere il problema del riconoscimento oggetti e quale metodo utilizzare per far andare la procedura sono stati eseguiti dei test in modo da avere un confronto sia dal punto di vista temporale che di fattibilità. In entrambi i casi

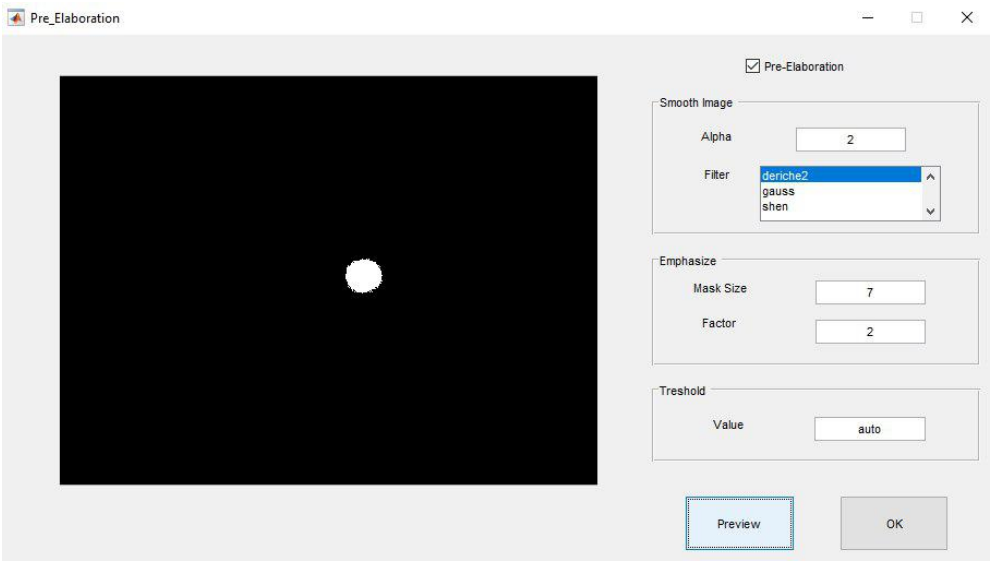

Figura 3.13: Schermata di pre-elaboration della GUI realizzata.

si sono ottenuti buoni risultati dal punto di vista del corretto riconoscimento degli oggetti esaminati, come si può vedere in figura 3.14. Si è quindi riusciti ad ottenere le coordinate da mandare al robot e l'intera procedura ha avuto successo, nella figura 3.15 è possibile vedere un corretto posizionamento da parte del robot durante una presa. Per poter poi andare a scegliere quale tipo di algoritmo usare, in un contesto in cui la velocità di elaborazione è uno dei parametri fondamentali, sono stati eseguiti dei test di elaborazione mantenendo le stesse impostazioni per entrambi. I risultati possono essere osservati in figura 3.16. In particolare

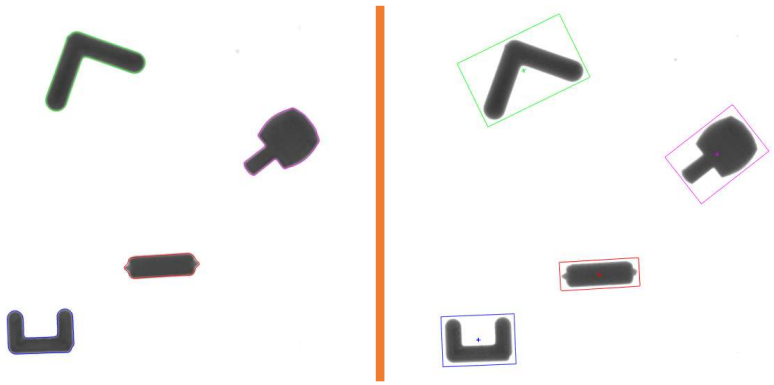

Figura 3.14: Risultati ottenuti a sinistra dallo shape-based matching mentre a destra dal template matching.

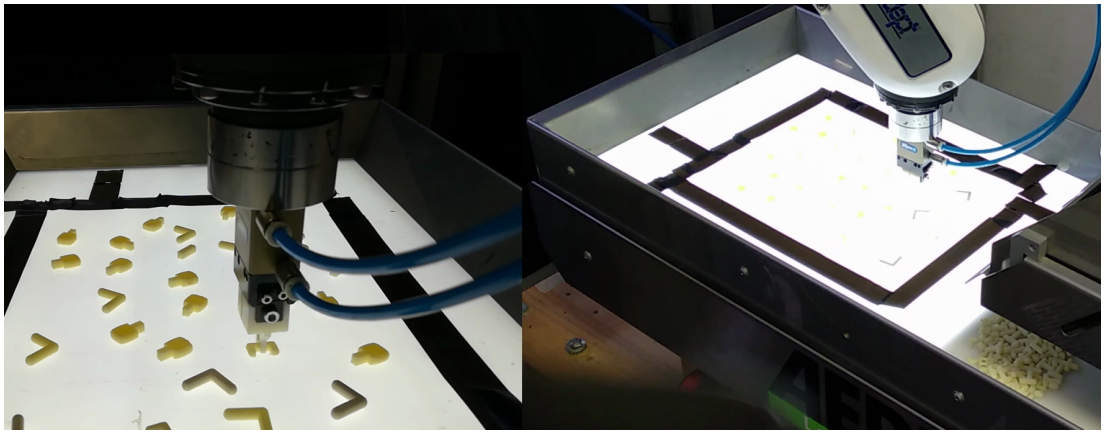

Figura 3.15: Presa Robot durante test sperimentali.

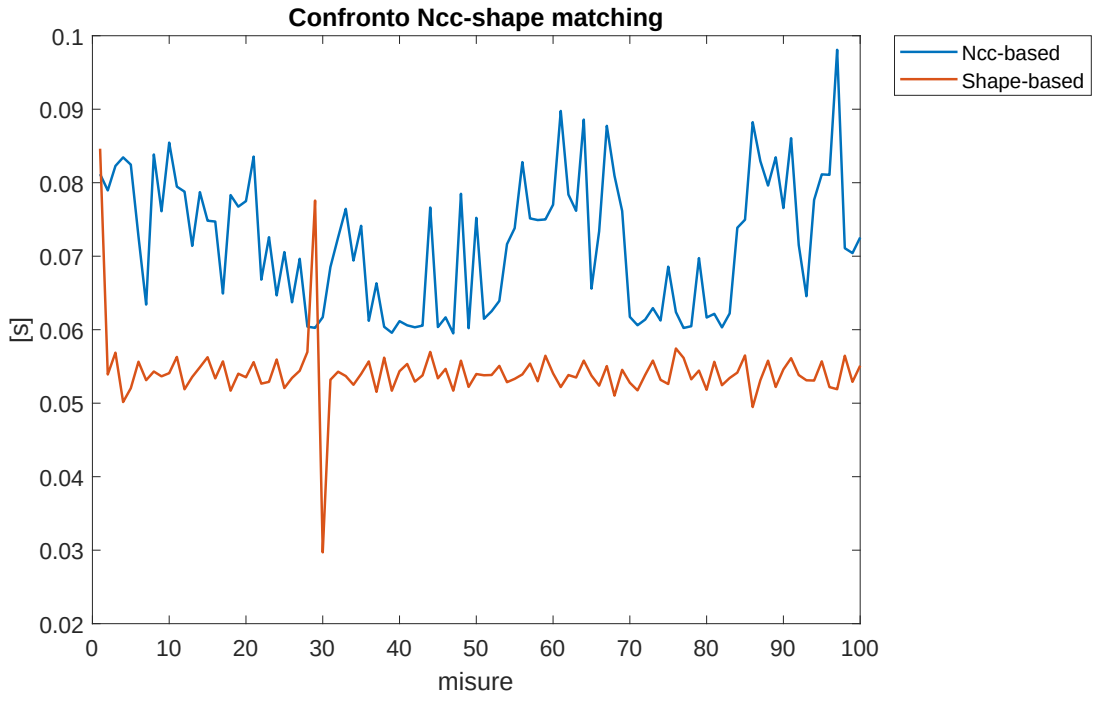

Figura 3.16: Confronto tempi di esecuzione Shape-based-matching con template matching.

	Media [ms]	Deviazione standard [ms]
Ncc	71.9	9.0
Shape_based	54.2	4.9

Tabella 3.3: tabella confronto shape-based-matching e template matching.

possiamo osservare come la curva dello shape-based rimane prevalentemente al di sotto di quella del grafico ncc. In particolare nella tabella riassuntiva 3.3 è possibile notare come il valor medio per la soluzione shape-based sia di 54.2 ms a fronte del tempo medio Ncc di 71.9 ms, di conseguenza il tempo impiegato dalla prima elaborazione corrisponde al 75.46% dell'altro . Anche per quanto riguarda la deviazione standard lo shape-based è da preferire essendo quasi la metà. Inoltre è da preferire lo shape-based matching a causa della sua robustezza rispetto ai cambi di luce. Con questa applicazione è stato possibile vedere tutti i benefici che i metodi per sfruttare halcon all'interno di Matlab possono portare. In particolare è stato possibile vedere come i metodi sviluppati siano stati utili sia nella parte di pre-processing andando a modificare direttamente da Matlab i valori di input, sia per i risultati esportati e utilizzati per pilotare il robot.

3.4 Acquisizione su nastro in movimento

Uno dei problemi tipici in ambito industriale è il riconoscimento di oggetti tramite un nastro in movimento. Questa operazione può risultare complessa in quanto la sincronizzazione tra acquisizione e velocità del nastro è di fondamentale importanza. Una volta ottenuto un segnale di trigger dall'encoder del nastro si può utilizzare direttamente per andare a dare un segnale di acquisizione alla telecamera. È stata allora sviluppata una procedura in Halcon in modo da poter gestire tutta la parte di acquisizione ed elaborazione senza andare ad utilizzare le risorse di Matlab. Per andare ad acquisire attraverso una telecamera matriciale una soluzione trovata è quella di andare a fissare la velocità del nastro e poi andare a selezionare un determinato numero di righe in fase di acquiszione in modo da non avere sovrapposizioni e cercare di non saltare troppe righe. Questo lavoro risulta molto più semplice e ideale per una telecamera di tipo lineare dato che acquisisce

la linea a seconda del trigger e non vi sono problemi di sovrapposizioni data la singola riga acquisita alla volta, in quelle matriciali non si può acquisire al di sotto di 2 righe. Dal punto di vista di Halcon non vi sono differenze se si utilizzano telecamere matriciali oppure una telecamera lineare. Infatti una volta definita l'altezza e larghezza dell'immagine non resta che lasciare scorrere il nastro e attaccare quella nuova successivamente in modo da creare un immagine di dimensioni maggiori. Infatti solitamente si utilizzano telecamere lineari le quali acquisiscono una singola riga ad ogni segnale di trigger, in modo allora di creare un immagine che sia in grado di contenere un intero oggetto al suo interno da riconoscere. Lo stesso discorso vale per le telecamere matriciali, di norma si utilizzano ROI molto strette(qualche pixel) in modo da fingere il funzionamento lineare. Per unire

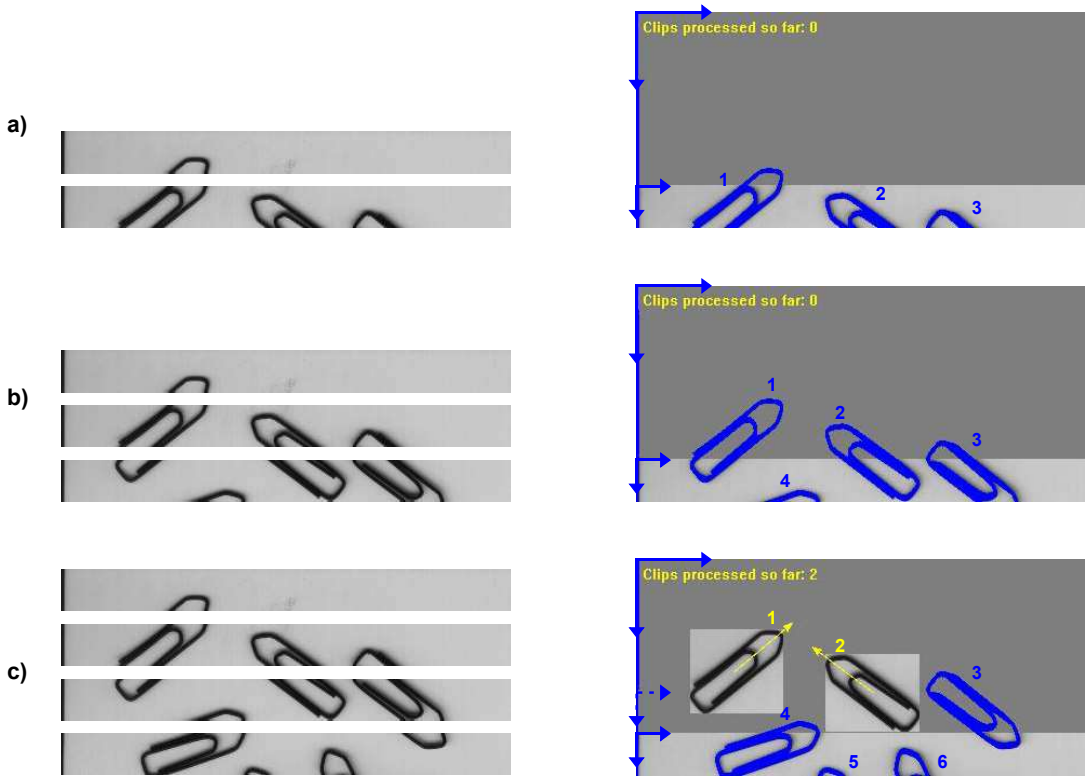

Figura 3.17: Procedura di acquisizione su nastro.

le diverse righe Halcon mette a disposizione il comando *tile_images* in modo da comporre un immagine sfruttando le linee precedenti. Si vuole sfruttare inoltre i comandi *merge_regions_line_scan* il quale una volta passata una regione viene confrontata con la precedente in modo da definire se corrisponde alla continuazione

della precedente. In caso positivo viene aggiornata in caso negativo viene scartata la precedente e utilizzata nella prossima iterazione. In questo tipo di applicazioni di solito viene utilizzato una luce di tipo back-light in modo da illuminare dal basso il nastro e facilmente imposatare una soglia per eseguire un operazione di treshold, la regione allora ottenuta sarà analizzata attraverso il metodo appena spiegato. Ad ogni iterazione le regioni possono essere elaborate quindi è possibile andare ad identificare un pezzo se rispetta determinate caratteristiche. Una volta riconosciuto un oggetto la regione corrispondente viene azzerata in modo da riuscire ad essere pronti a trovare un pezzo diverso. È possibile osservare questo procedimento in fiugra 3.17. Ad ogni nuova riga di codice quella vecchia viene soprascritta e di conseguenza vi è sempre presente un ciclo in cui avviene questo lavoro. A causa del tempo lungo di acquisizione e fatto in maniera continua, si è deciso di utilizzare halcon in formato stand-alone e per passare i valori di ogni oggetto trovato in termini di coordinate rispetto all'immagine di partenza vengono salvati direttamente attraverso *write_tuple*. Di conseguenza quando Matlab vuole accedere alla posizione del pezzo può semplicemente caricare il valore salvato in una cartella senza andare a gestire direttamente l'acquisizione. Un modo per andare a sincronizzarsi poi col nastro è quello di andarsi anche a salvare la tacca encoder rispetto la quale è stato trovato un pezzo in modo da poter sapere quella immagine dove si trova precisamente il pezzo in futuro.

3.5 3D matching con Kinect

Ultimamente in ambito industriale le soluzioni con sistemi di visione 3D sono sempre più utilizzate, questo è possibile a causa dell'abbattimento dei costi dei sensori. In questo lavoro è stato realizzato e testato l'utilizzo del microsoft kinect. Questo prodotto non è compatibile nativamente con Halcon, in particolare la fotocamera RGB funziona mentre quella di profondità non è supportata. Di conseguenza nasce l'esigenza di dover rendere disponibili le acquisizioni di questo strumento e il dover eseguire tramite Halcon l'elaborazione 3D dell'immagine di profondità. L'obbiettivo è, una volta definito un modello, andare a eseguire un algoritmo di matching in modo da ottenere la posa di un oggetto specifico nello

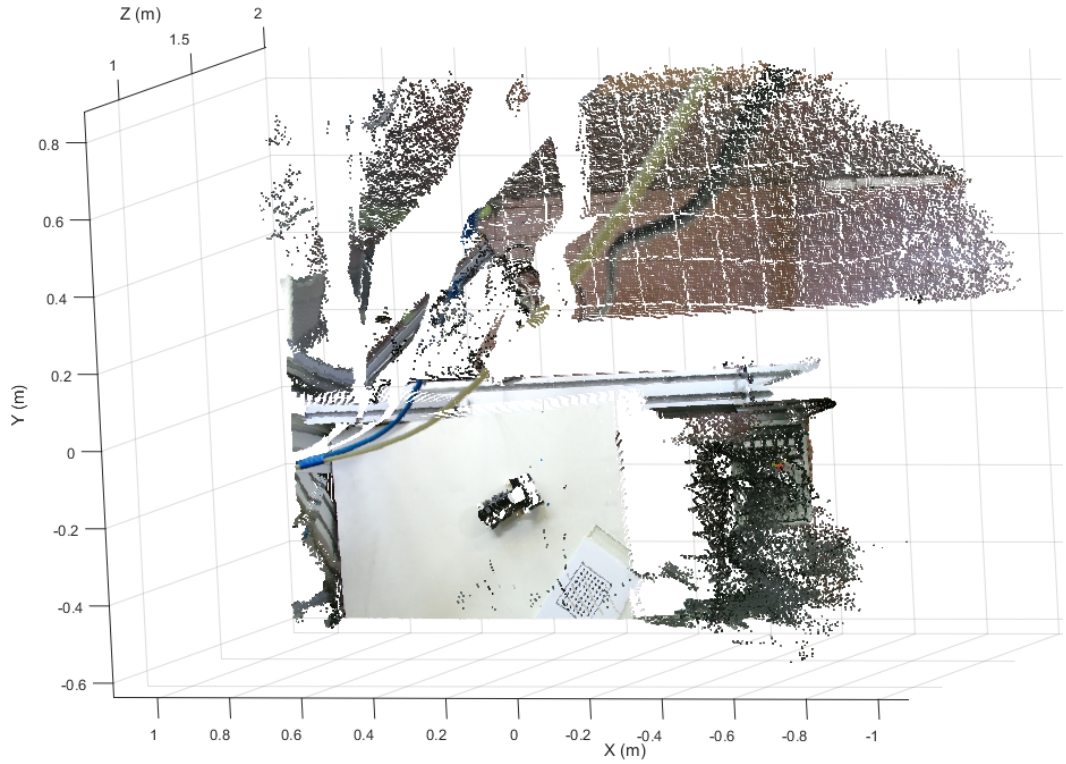

Figura 3.18: Nuvola di punti ottenuta tramite microsoft Kinect.

spazio 3D. Per prima cosa è necessario passare ad halcon l'immagine di profondità ottenuta dal Kinect, per risolvere questo problema si è utilizzato il computer vision toolbox[6] di Matlab in modo da gestire l'acquisizione, a cui è stata aggiunta l'estensione per acquisire tramite kinect. Utilizzando quindi i metodi di Matlab per l'acquisizione sfruttando entrambi i sensori presenti sul kinect è possibile ottenere la nuvola di punti in 3D ottenuta dal kinect illustrata in figura 3.18. Il sistema di riferimento è identificato dalla posizione del sensore di profondità. Questa immagine è il risultato della combinazione della mappa di profondità unita all'immagine RGB della fotocamera integrata nel kinect. Da questa è possibile notare come il campo visivo sia definito attraverso una mappa spaziale in 3D andando a definire la distanza dal sensore di ogni punto inquadrato. A questo punto bisogna passare ad halcon l'immagine di profondità. Il problema è che l'immagine depth creata da matlab non è compatibile, Halcon vuole un immagine con 3 canali rappresentanti una mappa in formato XYZ. Di conseguenza è necessario tramite Matlab convertire l'immagine depth in un immagine XYZ compattarla

Figura 3.19: Foto della scena acquisita tramite il sensore RGB del kinect a sinistra mentre a destra dal sensore di profondità.

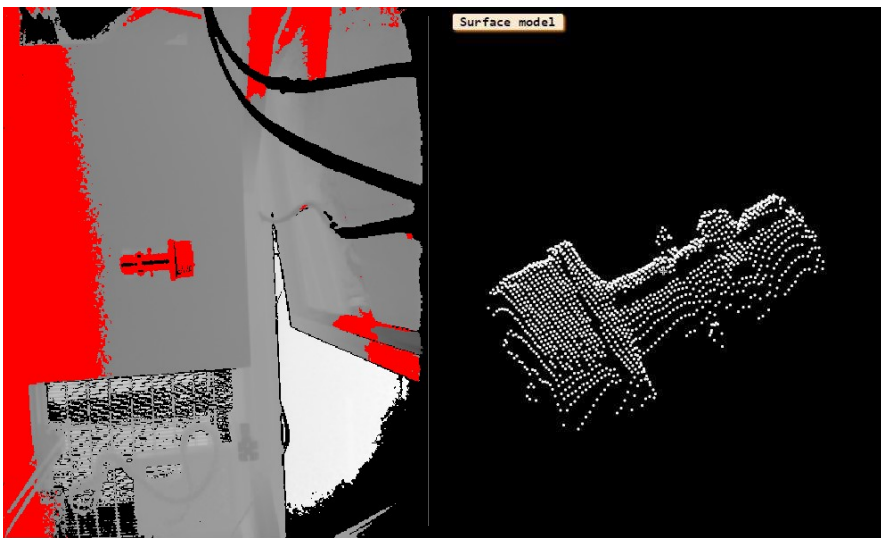

Figura 3.20: A sinistra si nota la treshold applicata ad un immagine di profondità, a destra è presente il modello acquisito tramite *create_surface_model*.

in una singola immagine e poi ci sono due alternative, salvarla attraverso un metodo halcon esportato *write_image* oppure utilizzare i tool forniti. Per convertire l'immagine da depth a XYZ si è seguita dalle librerie kinect si va quindi a definire i parametri del kinect:

- Lunghezza focale:

$$f_x = 367.28 \qquad f_y = 367.28 \tag{3.1}$$

- Principal point:

$$c_x = 255.16 \qquad c_y = 211.82 \tag{3.2}$$

Bisogna quindi andarsi a calcolare la Back-projection (2D to 3D) è necessario calcolarsi per ogni punto della mappa depth:

$$\begin{aligned} x &= \frac{(u - c_x)d}{f_x} \\ y &= \frac{(v - c_y)d}{f_y} \\ z &= d \end{aligned} \tag{3.3}$$

Si creano allora a partire da ogni pixel $(u, v)^T$ della mappa depth il vettore relativo alla mappa XYZ $(x, y, z)^T$. Attraverso il metodo esportato da Halcon *Compose3* si ottiene la mappa XYZ da poter utilizzare in Halcon, in figura 3.19 è possibile vedere il confronto tra la mappa esportata e l'immagine a colori acquisita. A questo punto l'intera proccesso si sposta su Hdevelop in modo da risucire a creare un modello dell'oggetto di prova in questo caso la telecamera lineare Dalsa completa di ottica. Per rendere tutto molto più semplice dal punto di vista computazionale si è applicata una treshold sui valori di grigio della mappa XYZ rispetto la componente Z. Questo si traduce non una soglia in luminosità come una normale treshold bensì in una soglia di distanza in cui è suddivisa la scena. Nel caso specifico essendo l'oggetto ad una distanza di circa 80cm dal kinect e con una altezza di circa 10 cm, si è impostata una treshold in modo da rimuovere il piano ma consierare solo i punti utili. Il risutlato di questa operazione è illustrato in figura 3.20. In questo modo diventa semplice attraverso HDevelop eseguire il taglio di una ROI in modo da isolare la telecamera in modo da poter ottenere il modello come nel caso di un template-matching. È possibile vedere il template acquisito

	Media [ms]	Deviazione standard [ms]
Rel_Sampling 0.05	60.10	1.28
Rel_Sampling 0.1	35.11	1.40

Tabella 3.4: Risultati numerici ottenuti dalle prove sperimentali.

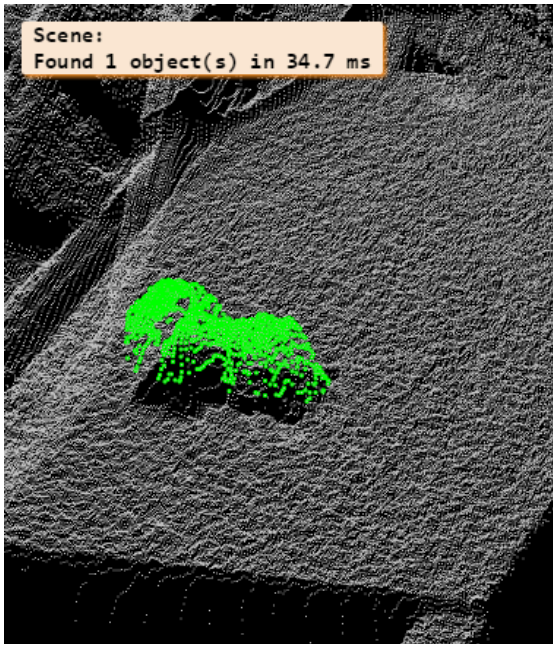

Figura 3.21: Risultato ottenuto tramite il *find_surface_model* del modello precedentemente cerato.

sempre in figura 3.20. A questo punto è possibile creare il modello attraverso il comando *create_surface_model*, l'immagine del modello creato è possibile vederla in 3.20. A questo punto bisogna andare a cercare l'oggetto una volta spostata la telecamera. Come per i casi di matching 2D non resta che utilizzare corrispettiva funzione *find_surface_model*. In questa maniera se vi sono match vengono restituiti tramite la posa in coordinate (x, y, z) e la terna di angoli di eulero rispetto ad un sistema di riferimento fissato (altrimenti viene usato come sistema di riferimento la posa del modello originale). Il risultato ottenuto è possibile vederlo in figura 3.21. Vi sono molti parametri cui impostare in modo da andare ad avere un trade of tra veloctità e robustezza come per i metodi 2D. In particolare i parametri che maggiormente influiscono la ricerca è il parametro di RelSamplingDistance. Questo parametro infuisce molto sulle performance dell'algoritmo in quanto è il

parametro che va a definire quanti punti considerare della scena in relazione al diametro dell'oggetto da trovare. Maggiore è questo parametro meno saranno i punti analizzati quindi si abbassa la precisione. Sono state eseguite delle misure appunto per andare ad osservare come al variare di questo parametro è possibile ottenere un miglioramento significativo. È possibile vedere questo confronto in fi-

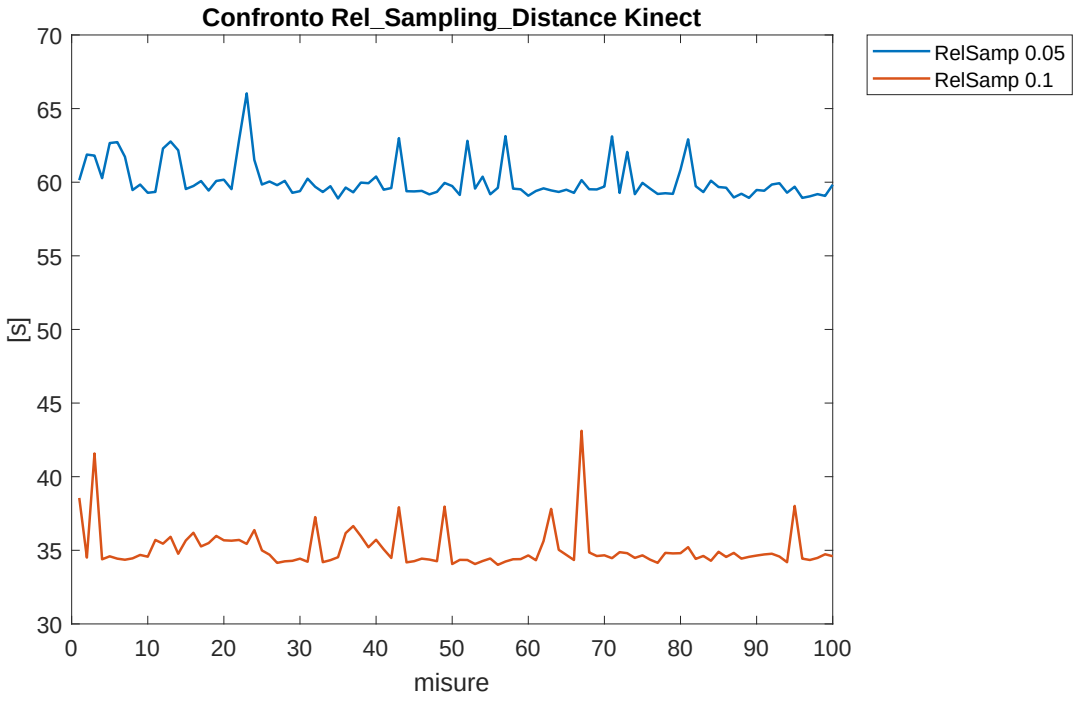

Figura 3.22: Confronto tempi di esecuzione metodo *find_surface_model* modificando il numero di punti acquisiti.

gura 3.22 e i parametri riassuntivi sono descritti in tabella 3.4. È possibile notare che i risultati sono buoni dato che è possibile trovare un oggetto 3D con un tempo medio di 35.11 ms, valore che a seconda delle applicazioni può essere migliore. Bisogna tenere presente che i problemi in ambito di visione 3D sono molto più complessi dal punto di vista computazione rispetto a quelli 2D. I tempi calcolati includono solamente i tempi del metodo *find* e non l'intera fase di acquisizione ed elaborazione.

Conclusioni

In questo lavoro si sono realizzati e sperimentati gli utilizzi della libreria professionale Halcon all'interno di Matlab. Questo è stato possibile tramite le classi e le funzioni realizzate rendendo molto semplice questo compito. È stato inoltre sperimentato il tempo di questi tool ottenendo dei buoni risultati rispetto ai metodi classici forniti, la soluzione migliore dal punto di vista delle prestazioni impiega circa un terzo del tempo del classico salvataggio su Ram-disk. Queste soluzioni sono state poi utilizzate per testare le librerie in diversi problemi classici legati alla visione. In particolare è stata possibile la realizzazione di una guida robot in maniera semplice sfruttando i metodi di Halcon. Sono state inoltre valutate le performance confrontando altri lavori, come per la batteria allettata, riuscendo ad eliminare un operazione costosa dal punto di vista temporale come l'acquisizione di un ulteriore immagine. Sono state sviluppate inoltre le procedure per l'acquisizione tramite nastro e attraverso il Kinect, questo in modo da semplificare i prossimi task di visione che verranno affrontati. In tutti questi compiti la libreria Halcon è risultata perfomante, in termini di qualità e velocità, in modo da favorirne l'uso per i futuri progetti attraverso i tool realizzati. I prossimi test da effettuare riguardano diversi metodi di acquisizione come la stereo visione e la lama laser, inoltre un ulteriore sviluppo potrebbe essere quello di semplificare il metodo di esportazione, andando a modificare il meno possibile il codice.

Bibliografia

[1] P. Corke, *Robotics, Vision and Control - Fundamental Algorithms in MATLAB*. Germany: Springer, 2011.

[2] https://www.mvtec.com/products/halcon/documentation/.

[3] A. Technology, *Adept Viper s650/s850 User's Guide*.

[4] B. Siciliano, *Robotics, Modeling Planning and Control*. Germany: Springer, 2011.

[5] A. Vision, *Pike Technical Manual*.

[6] https://it.mathworks.com/.