POLITECNICO DI MILANO

Corso di Laurea Magistrale in Ingegneria dell'Automazione

Dipartimento di Elettronica e Informazione

STUDIO E IMPLEMENTAZIONE DI UN ALGORITMO PER IL VISUAL CONTOUR FOLLOWING DI UN MANIPOLATORE ROBOTICO

Relatore: *Professor Luca Bascetta*

Tesi di Laurea di

Lorenzo Patanè

Matricola 754694

Anno Accademico 2011-2012

Indice

Indice delle Figure e delle Tabelle

Introduzione

I sistemi di visione artificiale sono una tecnologia chiave in molti ambiti della produzione industriale. Utilizzati in applicazioni sempre più sofisticate, come quelle di controllo della qualità, permettono di accedere a informazioni che non potrebbero essere diversamente elaborate automaticamente: forma, colore e contorni di oggetti, posizione e numero di difetti, ecc. In particolare nel campo dell'automazione, i sistemi di visione consentono di controllare il movimento di un robot durante la presa e l'assemblaggio di parti meccaniche o durante lavorazioni come saldatura, taglio, sbavatura, pulizia ecc. Per questo tipo di applicazioni sono state sviluppate innovative tecniche di controllo che consentono l'integrazione di questi sistemi ottici sfruttandone i vantaggi. Un esempio è il Visual Servoing, controllo che permette il posizionamento del robot usufruendo nell'anello di retroazione delle informazioni ottenute da una telecamera posta sull'organo terminale del manipolatore. In generale le tecniche di Visual Servoing sono classificate secondo le seguenti tipologie:

- Image Based (IBVS), basato sull'immagine
- Position Based (PBVS), basato sulla posizione
- Hybrid Approach, un approccio ibrido

In questa tesi è sviluppato un algoritmo di visione per l'edge detection di un oggetto di lavorazione. L'edge detection rappresenta uno degli step fondamentali di un sistema di controllo Visual-Servoing di tipo ibrido con configurazione Eye-In-Hand progettato per far compiere ad un robot industriale antropomorfo il contour following di un oggetto di lavorazione .

L'algoritmo di visione ha come obiettivi:

1. Acquisire ciclicamente da una telecamera in configurazione *Eye In Hand* (montata sull'end effector del robot) immagini in scala di grigi del pezzo posto sul piano di lavoro.

2. Elaborare le immagini al fine di identificare online la posizione e l'orientamento del contorno del pezzo, dati utili al sistema di controllo del robot per effettuare l'inseguimento del profilo durante la lavorazione.

Il lavoro s'innesta nel progetto di ricerca ECHORD-Fidelio attivo presso il Dipartimento di Elettronica e Informazione del Politecnico di Milano in collaborazione con il partner industriale COMAU S.p.A., leader nazionale per la produzione di manipolatori industriali.

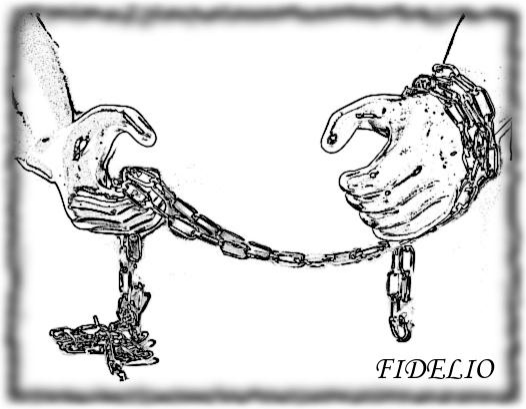


Figura 0.1 Logo del progetto Fidelio

L'attività di ricerca prevede sia lo studio teorico di nuovi paradigmi di programmazione basati su metodi probabilistici, che lo sviluppo di una metodologia di "programmazione autonoma" per manipolatori industriali. Quest'ultimo aspetto sarà finalizzato alla creazione di un sistema di sbavatura robotizzata per cerchioni automobilistici, in cui il robot dovrà essere in grado di apprendere autonomamente l'operazione di sbavatura a partire da una dimostrazione eseguita da un operatore umano esperto.

Organizzazione della tesi:

- Il primo capitolo si apre fornendo al lettore le informazioni necessarie per comprendere il problema affrontato, cosa s'intende per image features e in particolare per edge. Viene fornita una definizione di bordo e i metodi principalmente utilizzati per la sua identificazione (edge detection). Viene inoltre illustrato cosa si intende per contour following e quali sono le applicazioni industriali che ne fanno uso.
- Nel secondo capitolo si descrivono tutti i componenti hardware e software del sistema utilizzato in fase di progetto, fornendo le caratteristiche generali di ognuno ed entrando nel dettaglio di quelle unità di cui si è fatto largo uso.

- Nel terzo capitolo viene trattato il tema della Computer Vision presentando i maggiori campi di applicazione e sottolineando in particolar modo le applicazioni industriali che ne fanno uso. In seguito viene presentato il pacchetto software del sistema di visione artificiale, la libreria OpenCV, fornendo le informazioni relative alla sua reperibilità, alla sua organizzazione strutturale e alle sue convenzioni, e proponendo infine un semplice esempio di algoritmo per descriverne le funzione maggiormente utilizzate.
- L'implementazione dell'algoritmo di edge detection sviluppato durante il lavoro e di tesi è argomento nel capitolo quattro.
- Nel quinto capitolo sono riassunti i risultati sperimentali ottenuti.

Capitolo 1

Edge Detection e Contour Following

In questo capitolo saranno descritti tutti gli aspetti dell'*Image Analysis* che permettono di comprendere come il lavoro svolto ha soddisfatto gli obiettivi premessi.

Inizialmente sarà definito cosa si intende per *edge detection*, analizzando metodi di attuazione e problematiche annesse.

Nella seconda parte del capitolo verrà poi analizzata la procedura di *contour following,* descrivendo l'utilizzo che ne viene fatto nelle applicazioni industriali ed analizzando le soluzioni tecnologiche che possono fornire le linee guida per affrontare il problema.

1.1 Image features: Edge

Un'immagine può essere intesa come una struttura con un numero finito di elementi, chiamati pixel, ad ognuno dei quali è assegnato una preciso valore di intensità di grigio. Si riporta in Figura 1.1: Cerchione in lega di Alluminio un'immagine in cui sono state selezionate due aree rilevanti; analizzando queste regioni è possibile vedere come la distribuzione di intensità di grigio sia molto diversa dall'una all'altra.

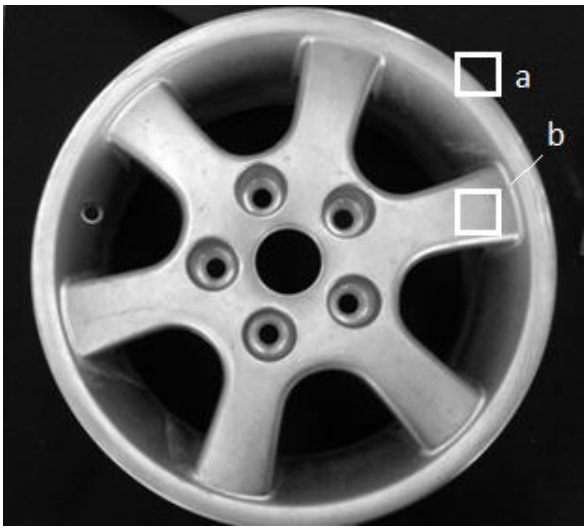


Figura 1.1: Cerchione in lega di Alluminio

Viene definita *featureless* un'immagine priva di caratteristiche rilevanti e che l'occhio umano percepisce come parte dello sfondo. La sua rappresentazione digitale sarà caratterizzata da valori di intensità uniformi.

Se la regione dell'immagine considerata presenta invece discontinuità in una direzione significa che l'immagine in tale zona presenta un *edge*. Queste interruzioni della continuità possono dovute a: discontinuità della profondità, discontinuità dell'orientamento delle superfici, modifica delle proprietà dei materiali o variazioni dell'illuminazione proveniente dall'ambiente circostante.

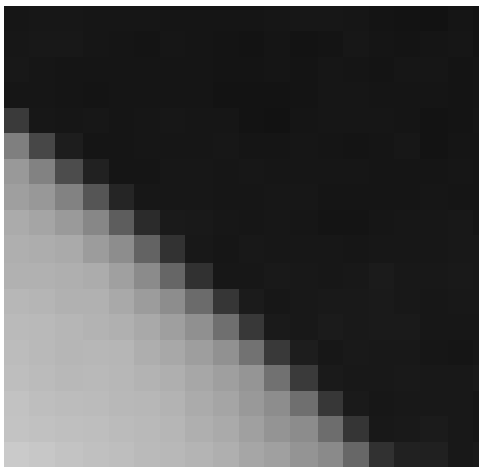


Figura 1.2

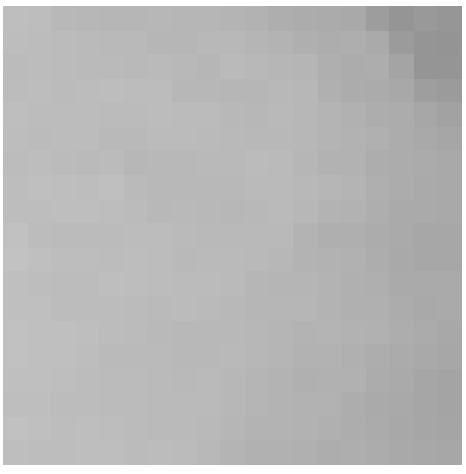


Figura 1.3

In Figura 1.2 e Figura 1.3 sono riportate gli ingrandimenti delle aree evidenziate in Figura 1.1.

1.2 Edge detection

L’individuazione dei contorni o edge detection è il metodo più comune per identificare le discontinuità nei livelli di grigio in un'immagine. Questo approccio rientra nel campo di ricerca del trattamento delle immagini e della visione artificiale, ed è rilevante in particolare nella branca del riconoscimento delle caratteristiche *(feature extraction*). L'operazione di riconoscimento dei contorni genera immagini contenenti molte meno informazioni rispetto a quelle originali, poiché elimina la maggior parte dei dettagli non rilevanti al fine dell'individuazione dei contorni. Le informazioni essenziali per descrivere la forma e le caratteristiche strutturali e geometriche degli oggetti rappresentati vengono invece conservate.

Per affrontare e risolvere correttamente il problema è necessario definire cosa s’intente per *bordo*. Come si osserva in Figura 1.4a, a causa di imprecisioni nell'acquisizione dell'immagine il bordo non presenta una netta transizione tra i livelli di grigio, ma è caratterizzato da una variazione graduale che può essere ben approssimata da un modello come quello in Figura 1.4b.

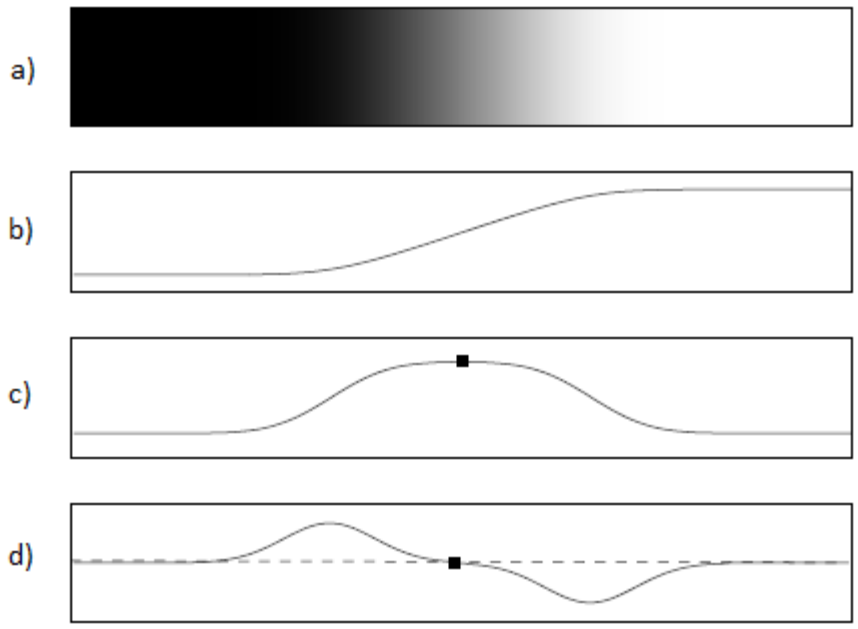


Figura 1.4

Per identificare i bordi, gli approcci principalmente utilizzati sono i metodi basati sulla ricerca dei punti estremanti (*search-based*) e metodi basati sull'attraversamento dello zero (*zero-crossing*). I metodi basati sulla ricerca riconoscono i contorni calcolando i massimi e i minimi della derivata del primo ordine dell'immagine (Figura 1.4c), di solito individuando la direzione in cui si ha il massimo gradiente locale. I metodi zero-crossing ricavano i punti in cui la derivata del secondo ordine passa per lo zero (Figura 1.4d), considerando la funzione laplaciana o un'espressione differenziale se la funzione è non-lineare. Un'importante problematica di queste tecniche è la loro elevata sensibilità al rumore. Per limitare questa fonte di errore le immagini vengono processate con filtri che permettono di eliminare o almeno diminuire i disturbi. Tecniche più complesse, come l'algoritmo di Canny, integrano con altre tecniche quelle già citate, al fine di ottenere migliori risultati.

Nei seguenti paragrafi sono riportate le informazioni di base e le motivazioni che spingono a utilizzare queste tecniche. Per maggiori dettagli si faccia riferimento a [1], [2].

1.2.1 Tecniche search-based

Per determinare l'intensità e la direzione di un edge nella posizione (x, y) si utilizza il vettore gradiente il quale contiene le derivate parziali dell'immagine calcolate lungo le direzioni principali. Il gradiente è un indice della velocità di variazione della grandezza a cui si applica: appare quindi naturale ricercare i punti di edge tra quelli caratterizzati dai valori elevati del gradiente.

La direzione del vettore gradiente

$$\nabla f = [g_x, g_y] = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

è quella di massima variazione di $f(x, y)$ nel punto (x, y). Tale direzione è data dall'angolo

$$\alpha(\mathrm{x}, \mathrm{y}) = \tan^{-1}\left(\frac{\mathrm{g_y}}{\mathrm{g_x}}\right)$$

Il vettore ha quindi direzione normale all'edge, verso positivo andando dalla regione più scura a quella più chiara e ampiezza $|\nabla f| = \sqrt{(g_x^2 + g_y^2)}$

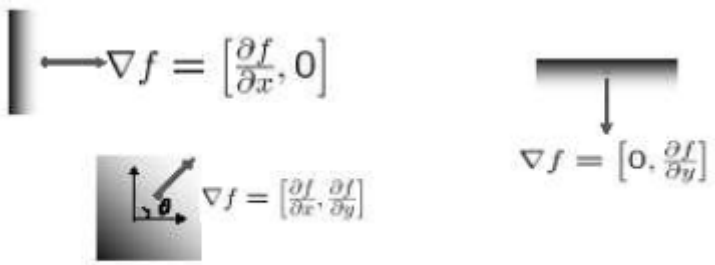


Figura 1.5

I metodi che utilizzano la derivata prima applicano all'immagine opportuni filtri che stimano il gradiente. La difficoltà nell'uso di questi filtri è la loro estrema sensibilità al rumore. Questa dipendenza può essere ridotta effettuando l'operazione detta di smoothing, che sarà discussa in seguito.

1.2.1 Tecniche zero-crossing

Altri operatori per il riconoscimento dei contorni si basano sul calcolo della derivata seconda dell'intensità, che si può approssimare con la rapidità di variazione del gradiente. I punti del contorno corrispondono alle coordinate in cui vi è il passaggio a zero della derivata seconda, che viene calcolata usando l'operatore Laplaciano definito come:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

1.2.1 Filtro di Smoothing

I filtri di smoothing vengono utilizzati per rimuovere piccoli dettagli in un'immagine e sono molto utili per ridurne il rumore. In entrambi i casi l'effetto complessivo è quello di rendere l'immagine più sfuocata di quella originale (*blurring*). Questi filtri effettuano una media dei valori di intensità di grigio dei pixel di una determinata area dell'immagine secondo opportuni pesi in modo da ridurre le brusche variazioni.

Analizzando il caso più semplice, ciò che viene fatto è applicare un filtro lineare, il quale si ottiene dalla convoluzione tra l'immagine ed una matrice costante detta maschera o finestra. Se le ipotesi fatte sul tipo di rumore sono esatte, questo può essere ridotto effettuando una media su pixel adiacenti (*neighborhood averaging*).

In altri casi, se per esempio i valori della maschera sono calcolati a partire dalla distribuzione Gaussiana 2D si parla di Gaussian smoothing. La funzione semplificata è:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2}\frac{x^2+y^2}{\sigma^2}}$$

Una Gaussiana non è mai nulla, ma è prossima a zero a partire da tre deviazioni standard dalla media, perciò troncando adeguatamente i valori si può ottenere una finestra di dimensioni opportune.

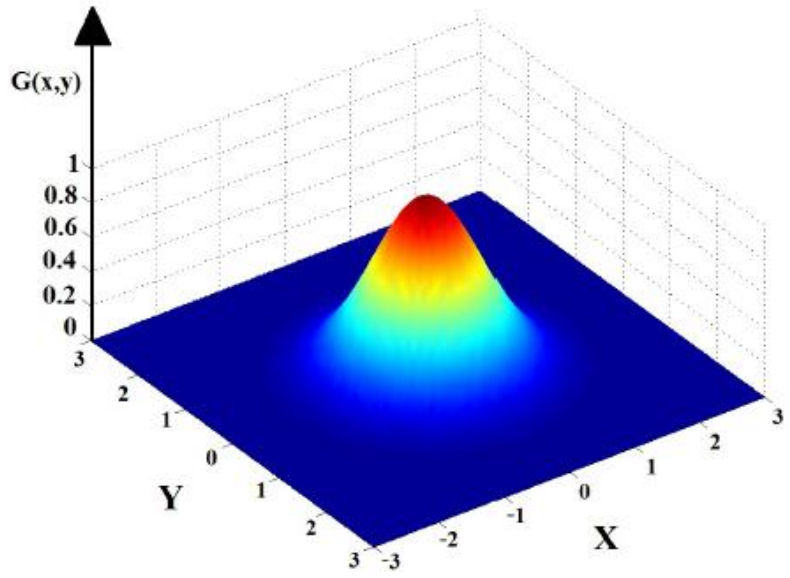


Figura 1.6

È necessario considerare che nelle zone dell'immagine caratterizzate da bordi, l'elaborazione provoca una distorsione che ha il risultato visivo di sfocare i contorni. Bisogna allora trovare il giusto compromesso tra riduzione del rumore e distorsione dimensionando opportunamente i pesi dei filtri.

In Figura 1.7 e Figura 1.8 viene riportato un confronto tra l'immagine originale e l'immagine ottenuta dopo un processo di smoothing.

Figura 1.7

Figura 1.8

1.2.2 Thresholding a doppia soglia

Esistono casi in cui le tecniche di identificazione del bordo falliscono. In questi casi i contorni identificati non sono mai netti, ma solitamente affetti da una o più distorsioni. Le cause più frequenti sono condizioni d'illuminazione che determinano la presenza di ombre o riflessi oppure un'imperfetta messa a fuoco dell'immagine dovuta a un limite della profondità di campo dell'ottica del dispositivo di acquisizione.

Consideriamo una tecnica basata sul gradiente: per stabilire se un pixel appartiene ad un edge oppure no vengono fissate due soglie T_{high} e T_{low}.

Se il relativo valore del gradiente:

- Supera la soglia T_{high}, allora è classificato come un bordo (edge);
- È minore della soglia T_{low}, viene annullato;
- È compreso tra T_{low} e T_{high}, viene conservato solo se è vicino a pixel classificati come edge.

Questa tecnica prende il nome di *operazione di Thresholding a doppia soglia*. In figura X è riportato un esempio di sogliatura con $T_{low} = 80$ e $T_{high} = 250$

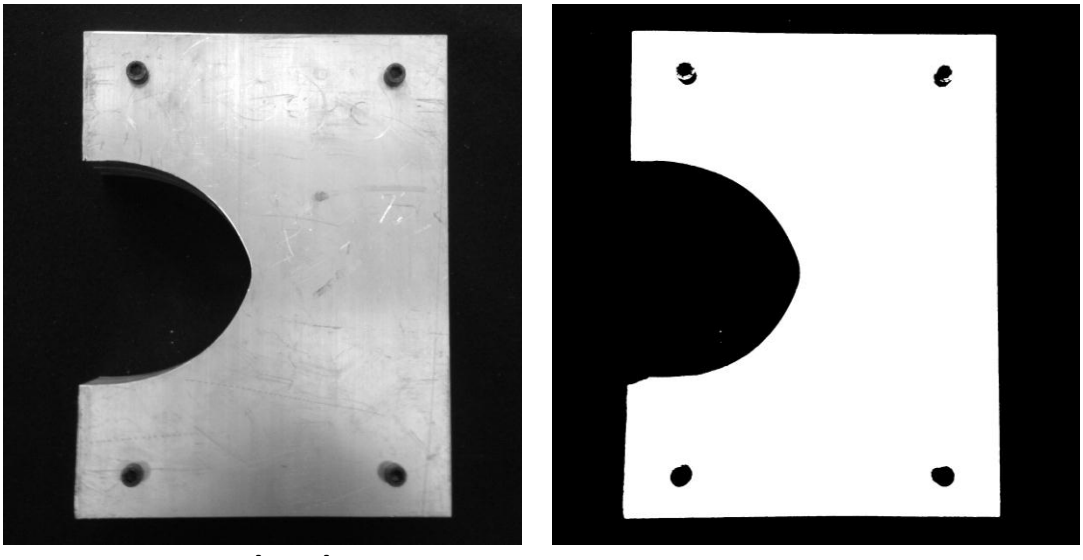


Figura 1.9

Il Canny edge detector è un metodo molto potente per la rivelazione dei bordi in un'immagine. Esso cerca di migliorare il risultato dell'elaborazione operando in quattro fasi:

a) *riduzione del rumore* per mezzo di un filtro gaussiano. Il risultato è un'immagine con una leggera "sfocatura", in cui nessun singolo pixel è affetto da disturbi di livello significativo.
b) Si effettua una *differenzazione* che permette di calcolare il gradiente dell'immagine. Dal momento che il contorno di un'immagine può puntare verso una direzione qualsiasi, l'algoritmo di Canny usa quattro filtri differenti al fine di individuare nell'immagine i contorni orizzontali, quelli verticali e quelli sulle due diagonali. Il filtro che ottiene il valore maggiore corrisponde al massimo gradiente di luminosità e identifica pertanto la direzione dell'edge per uno specifico pixel.
c) *soppressione dei non massimi*. Utilizzando la mappa dei gradienti è possibile stabilire i punti corrispondenti a massimi locali che saranno considerati come punti di un contorno e saranno presi in considerazione per l'ultimo step di elaborazione.
d) si usa un *Thresholding a doppia soglia* allo scopo di eliminare i pixel che presentano un gradiente minore della soglia bassa (dovuto probabilmente a rumore) e conservare quelli che superano la soglia alta (classificati come edge point). I pixel che presentano un gradiente compreso tra le due soglie vengono conservati solo se vicini ai pixel che hanno superato la soglia alta.

Seguono due immagini d'esempio di edge detection mediante algoritmo di Canny. Nell'immagine del cerchione (Figura 1.10) il valore di soglia minima è stato fissato a 50 mentre la soglia massima è 100. Nel secondo esempio $T_{low} = 80$ e $T_{high} = 150$

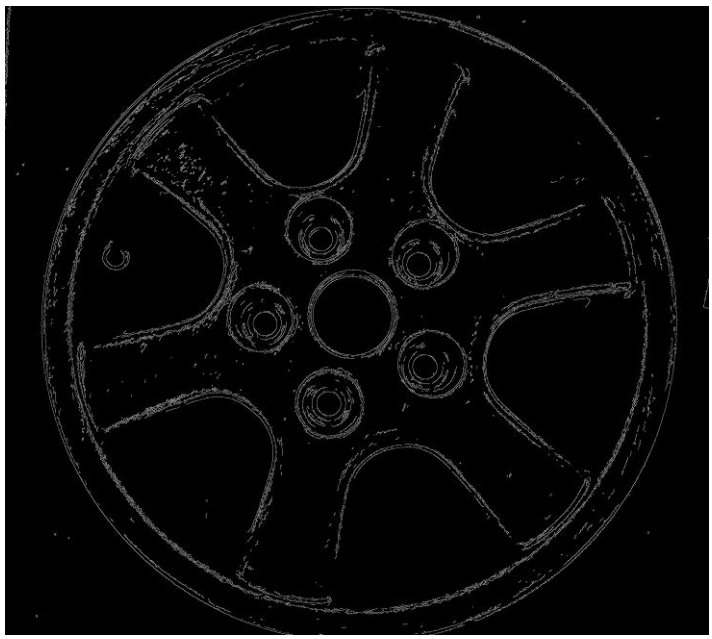


Figura 1.10

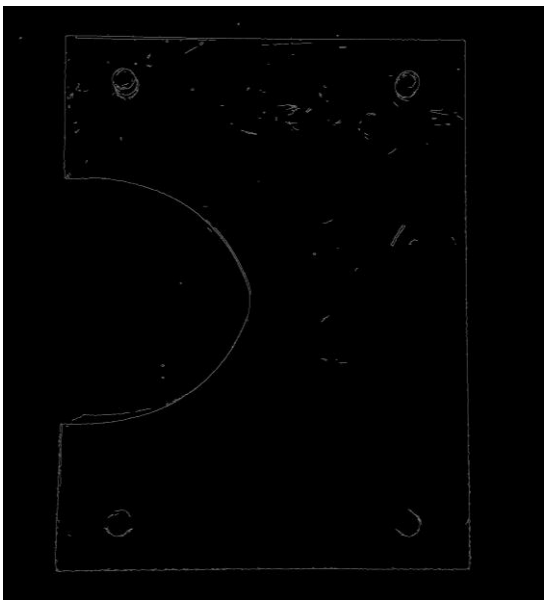


Figura 1.11

Le immagini originali sono la Figura 1.7 e la Figura 1.9.

1.3 Contour following

In ambito industriale i robot programmabili sono utilizzati per una varietà di applicazioni ripetitive. La programmazione dei movimenti del robot può essere un processo complicato e che richiede tempo per lo sviluppo. Semplificare la programmazione della traiettoria è dunque una necessità prioritaria per l'industria robotica.

Il robot, in applicazioni come saldatura, taglio, pulizia, sbavatura e lucidatura, deve compiere percorsi lungo il contorno del pezzo, effettuare cioè il *Contour Following*.

I metodi per la programmazione del percorso che si intende far compiere ad un robot possono essere classificati come *CAD based* e *non-CAD based*. Un tipico sistema basato su CAD consente agli utenti di selezionare una parte del modello CAD, come una superficie o un bordo, in riferimento al quale vengono generati i punti del percorso. Esistono tuttavia frequenti casi in cui non si dispone dei disegni CAD dell'oggetto di lavorazione. Vi è quindi la necessità di utilizzare sensori esterni per acquisire efficacemente le informazioni geometriche del pezzo, in particolare nella zona di interesse per la lavorazione. Negli ultimi anni i dispositivi più utilizzati a tale scopo sono sensori di forza integrati con sistemi di visione che hanno determinato lo sviluppo di numerose applicazioni industriali di contour following con controllo di forza e/o controllo di visione [3].

1.3.1 Contour following con controllo di forza e di posizione

In contesti industriali dove il rapporto costi / benefici, la semplicità, la robustezza e l'affidabilità devono essere attentamente bilanciati, può essere opportuno applicare solamente un controllo di forza integrato al controllo di posizione per risolvere il problema dell'inseguimento del contorno. Questa soluzione non necessita, infatti, né di costosi sistemi di visione, né delle competenze per il loro utilizzo, fornendo in molti casi risultati soddisfacenti.

Un esempio è il controllo proposto in [3], dove l'idea di base è insegnare al robot alcune posizioni approssimative lungo il percorso desiderato durante la prima fase di programmazione. Applicare in seguito, se necessario, degli algoritmi per effettuare

le regolazioni supplementari al percorso registrato e infine realizzare il contour following con un controllo di forza in retroazione che sfrutta la guida dei punti precedentemente memorizzati. I test con applicazioni reali hanno dimostrato la solidità e l'efficacia di tale sistema in ambienti industriali.

1.3.2 Contour following con controllo ibrido di forza e di visione

In operazioni di contour following planari, ovvero quando l'operazione avviene su un piano solitamente parallelo al piano di lavoro, il robot afferra un utensile mentre effettua l'inseguimento del bordo. In questo caso un controllo di forza può essere utilizzato per generare o modificare la traiettoria dell'end effector. La velocità di esecuzione della lavorazione, tuttavia, è limitata al fine di evitare la perdita di adesione, o la nascita di forze al contatto eccessive, soprattutto in corrispondenza di angoli o profili con curvature elevate.

Il controllo ibrido di forza e di visione consente di migliorare le prestazioni in questi termini, permettendo inoltre di mantenere la forza desiderata al contatto anche mentre si affronta un brusco cambio di direzione.

Tra le pubblicazioni scientifiche che trattano quest'argomento [4], [5] si vuole mettere in evidenza lo studio di J. Baeten e J. De Schutter [6]. Essi presentano un approccio valido per percorsi planari con utensili rotazionali, particolarmente utile per lavorazioni di una sola passata nel quale la calibrazione del pezzo che garantisce l'accuratezza del posizionamento è dispendiosa o impossibile. In tale approccio è prevista la presenza di una camera montata sull'end effector del robot che si mantiene sopra il contorno del pezzo mentre il sistema di regolazione controlla la forza al contatto. Un algoritmo di visione effettua l'edge detection in tempo reale e individua eventuali angoli. Se un angolo viene individuato, viene attivato un controllo a stati finiti che permette di affrontare lo spigolo con successo.

L'algoritmo di edge detection presente in questa pubblicazione scientifica è stato scelto come linea guida per lo sviluppo del progetto svolto in questa tesi.

Per identificare la posizione di un punto del bordo nello spazio immagine viene applicato su una riga o colonna dell'immagine acquisita (in scala di grigi) un operatore che combina un filtro di smoothing e un operatore differenziale.

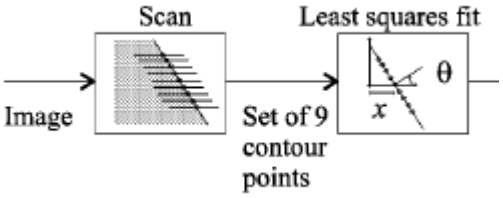


Figura 1.12

Successivamente in un'area di interesse dell'immagine si esegue una scansione dei successivi punti del contorno. In totale vengono estratti nove punti disposti simmetricamente rispetto al centro dell'immagine, le cui coordinate in pixel sono salvate in una struttura dati:

$$[X_p, Y_p], con\ X_p = [x_1, \dots, x_9]' e\ Y_p = [y_1, \dots, y_9]'$$

Valutando la retta di regressione lineare con il metodo dei minimi quadrati a partire set di coordinate è possibile determinare la posizione x_p^C [pixel] e l'orientamento θ^C[rad] del contorno nell'immagine.

Per una miglior comprensione è riportata in formato matriciale la retta di regressione e una figura che fornisce un esempio di questi parametri.

$$\begin{pmatrix} Y_p & \begin{matrix} 1 \\ \dots \\ 1 \end{matrix} \end{pmatrix} \begin{pmatrix} \tan(\theta^C) \\ x_p^C \end{pmatrix} = X_p$$

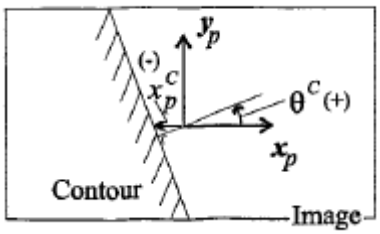


Figura 1.13

Numerose pubblicazioni scientifiche ([7], [8], [9]) nell'ambito del contour following si avvalgono delle tecniche proposte in [6]: questo fatto può essere interpretato come la garanzia dell'efficacia di questo algoritmo.

Capitolo 2

Il Setup Sperimentale

In questo capitolo verranno descritte le architetture hardware e software utilizzate nell'ambito di questo lavoro di tesi. Di seguito sono menzionati tutti i componenti hardware e in Figura 2.1 è rappresentata la loro connessione:

- Robot COMAU SMART SiX
- Unità di Controllo COMAU C4G Open
- uEye CAMERA UI-5240SE-M
- PC LINUX
- SERVER LINUX REALTIME

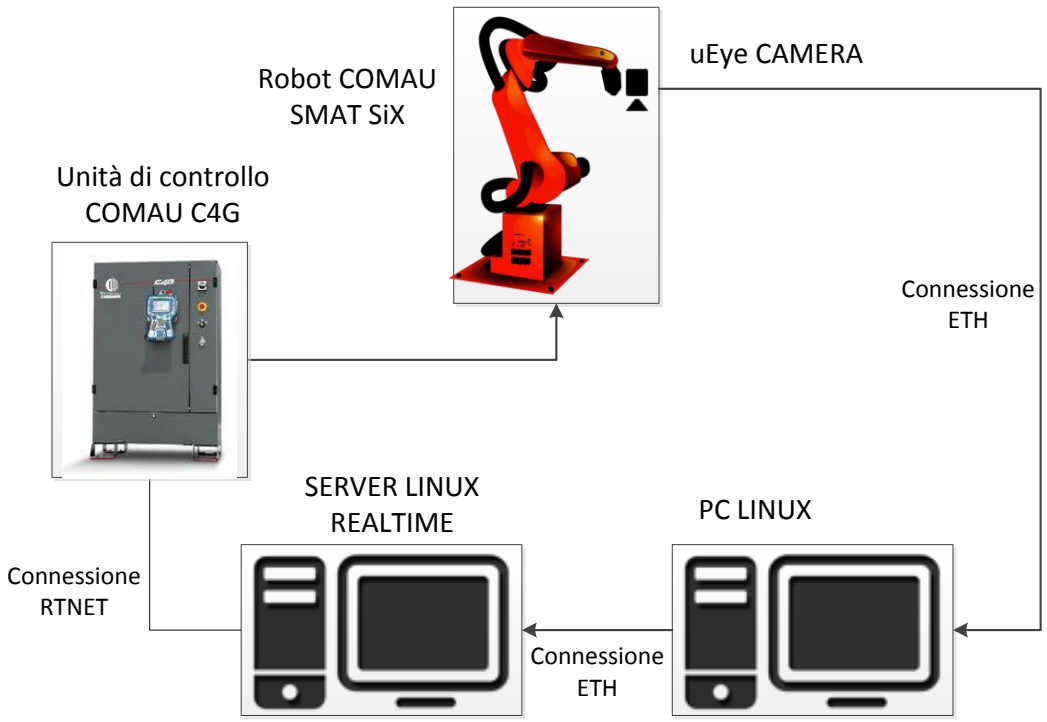


Figura 2.1

2.1 Il sistema di visione

L'acquisizione delle immagini avviene tramite una telecamera posta sull'end-effector del manipolatore e collegata alla scheda di rete Ethernet del computer con sistema operativo Linux. La camera uEye, modello UI-5240SE-M mono colore, è prodotta dalla azienda IDS (Imaging Development System) e dispone di un sensore ottico *CMOS ev2* con risoluzione 1.3 Megapixel (1280x1024 pixel) [10].

Il sensore è il componente principale di ogni camera digitale ed è costituito da un chip che converte la luce in cariche elettriche, che vengono poi trasformate in informazioni digitali. Il sensore può essere di tipo CCD (Charge-Coupled Device) o CMOS (Complementary Metal-Oxide-Semiconductor). Entrambe le tecnologie si basano sul fotodiodo, componente fotosensibile che per effetto fotoelettrico genera una carica quando colpito da luce incidente.

Nel caso del sensore CCD, la carica elettrica generata da ogni fotodiodo viene convertita in voltaggio e trasferita in uscita sotto forma di segnale analogico. Questo tipo di sensore offre molti vantaggi in termini di qualità.

In un sensore CMOS ogni fotodiodo è accompagnato sia da un convertitore, che trasforma l'energia luminosa in voltaggio, sia da circuiti di digitalizzazione che rendono il segnale di uscita digitale. Uno dei limiti più significativi dei sensori CMOS deriva dalla loro minore sensibilità alla luce.

Il sensore *CMOS e2v* combina i vantaggi delle due tecnologie: elevata sensibilità e uscita digitale.

Figura 2.2

Sul retro della camera uEye sono collocate una porta RJ45 per l'interfacciamento Ethernet e un connettore a 6 pin per l'alimentazione.

In Tabella 1 sono riportate tutte le caratteristiche di cui la telecamera dispone [10].

Interfaccia	GigE
Lenti montate	C-Mount
Tecnologia dei sensori	CMOS (e2v)
Modello	UI-5240SE-M
Risoluzione (h x v)	1280 x 1024
Risoluzione	1.3 Megapixel
Dimensione sensore	1/2"
Otturatore	Global
Max Frame rate in Freerun Mode	50 fps
Tempo di esposizione Max	9µs – 2000ms
Modalità Area of Interest (AOI)	orizzontale + verticale
Trigger I/O	1
Flash I/O	1
Modello sensore	EV76C560BB / EV76C560BC
Pixelpitch in µm	5.30
Dimensione ottica	6.784 x 5.427 mm
Classe di protezione	IP30
Dimensione H / W / L	34.00 mm, 44.00 mm, 43.50 mm
Peso	108.00 g
Alimentazione	12 - 24V

Tabella 1

2.1.1 Modalità di acquisizione delle immagini

Il sistema di visione supporta la cattura di singoli frame (snap) e quella di sequenze di frame (live). Entrambe le tecniche possono essere utilizzate, a seconda dell'applicazione, in modalità *Trigger Mode* o in modalità *Freerun,* di seguito descritte:

a) *Modalità Freerun*
   In questa modalità il sensore della telecamera cattura immagini in sequenza ad intervalli fissati (max 50 fps). L'esposizione dell'immagine corrente e la trasmissione dei dati immagine precedenti sono eseguite contemporaneamente in modo da poter raggiungere il frame rate massimo.

b) *Trigger mode*
   In questa modalità il sensore rimane in standby e si attiva solo quando riceve la richiesta di un immagine da un segnale di trigger. Al termine del tempo di esposizione il frame viene trasferito al PC.

Le seguenti figure mostrano lo schema di una sequenza di acquisizione delle immagini. L'esposizione del sensore, i tempi di lettura e i tempi di trasmissione dipendono dalle impostazioni correnti dei parametri della camera uEye. Il tempo di pre-processing dipende dalle funzioni API disponibili che l'utente intende attivare, come ad esempio la funzione di miglioramento dei bordi [11].

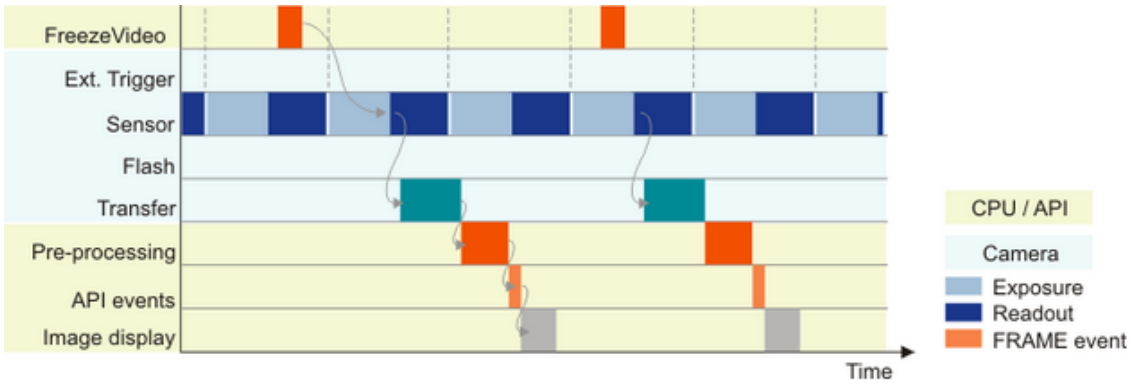


Figura 2.3: Cattura di singoli frame in Freerun mode

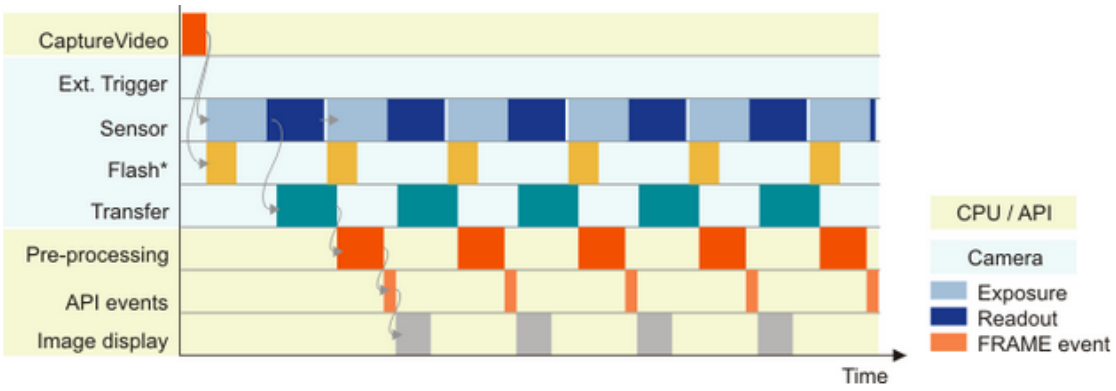


Figura 2.4: Cattura sequenza video in Freerun mode

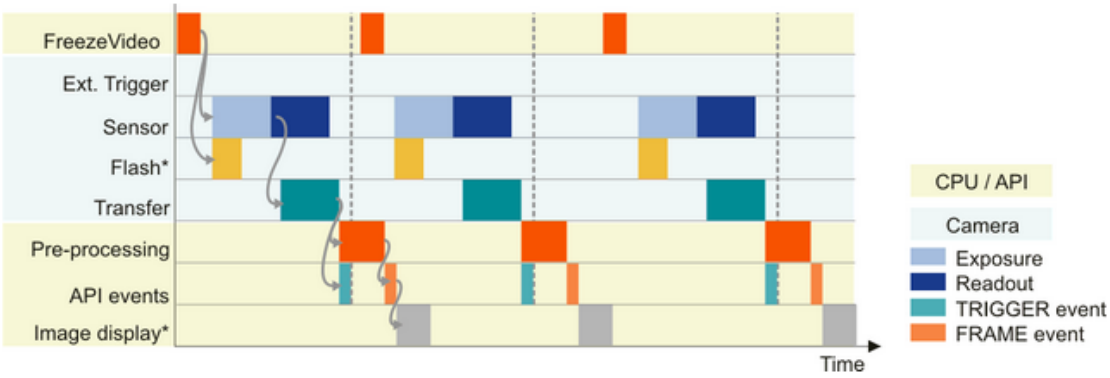


Figura 2.5: Cattura di un singolo frame in Trigger mode

Dopo aver valutato le opzioni disponibili si è giunti alla conclusione che, al fine di effettuare l'edge detection di un profilo metallico per il contour following, la tecnica più efficiente sia acquisire ciclicamente un singolo frame in trigger mode. Se si considera che per compiere l'edge detection si ha la necessità di processare l'immagine tra un'acquisizione e la successiva, risulta ottimale la scelta di trasmettere i dati e salvarli in memoria solo quando risulta strettamente necessario. La modalità Trigger permette inoltre di ottimizzare i tempi di cattura, come risulta evidente dal confronto diagrammi temporali di Figura 2.3: Cattura di singoli frame in Freerun mode e di Figura 2.5: Cattura di un singolo frame in Trigger mode.

2.1.2 Pacchetto software e programmazione con uEye SDK

Il pacchetto software fornito dalla casa costruttrice comprende driver (sia per ambiente Windows che per ambiente Linux) e SDK (Software Development Kit), ovvero un insieme di strumenti per l'integrazione e lo sviluppo di applicazioni. Tra linguaggi di programmazione supportati si trovano C, C++ e Visual Basic. Sono inoltre disponibili software commerciali, come *Halcon* [12], che dispongono dei driver per questo tipo di telecamere.

Per l'acquisizione e la gestione delle immagini è stato scelto un ambiente di sviluppo Linux per ragioni di omogeneità con il sistema Robot/Unità di Controllo/PC Linux Realtime già disponibile nel laboratorio dove è stata condotta la ricerca. La scelta di adottare un secondo calcolatore è stata obbligata dall'assenza di driver per Linux Realtime.

Per la programmazione è stato adottato il linguaggio C, in quanto software Open Source che evita investimenti in programmi commerciali e risulta essere più in linea con la politica di ricerca universitaria.

La procedura utilizzata nell'ambito del progetto che sfrutta le funzioni del SDK e che permette il setup della telecamera e l'acquisizione delle immagini è rappresentata nel flow chart in

Figura 2.6.

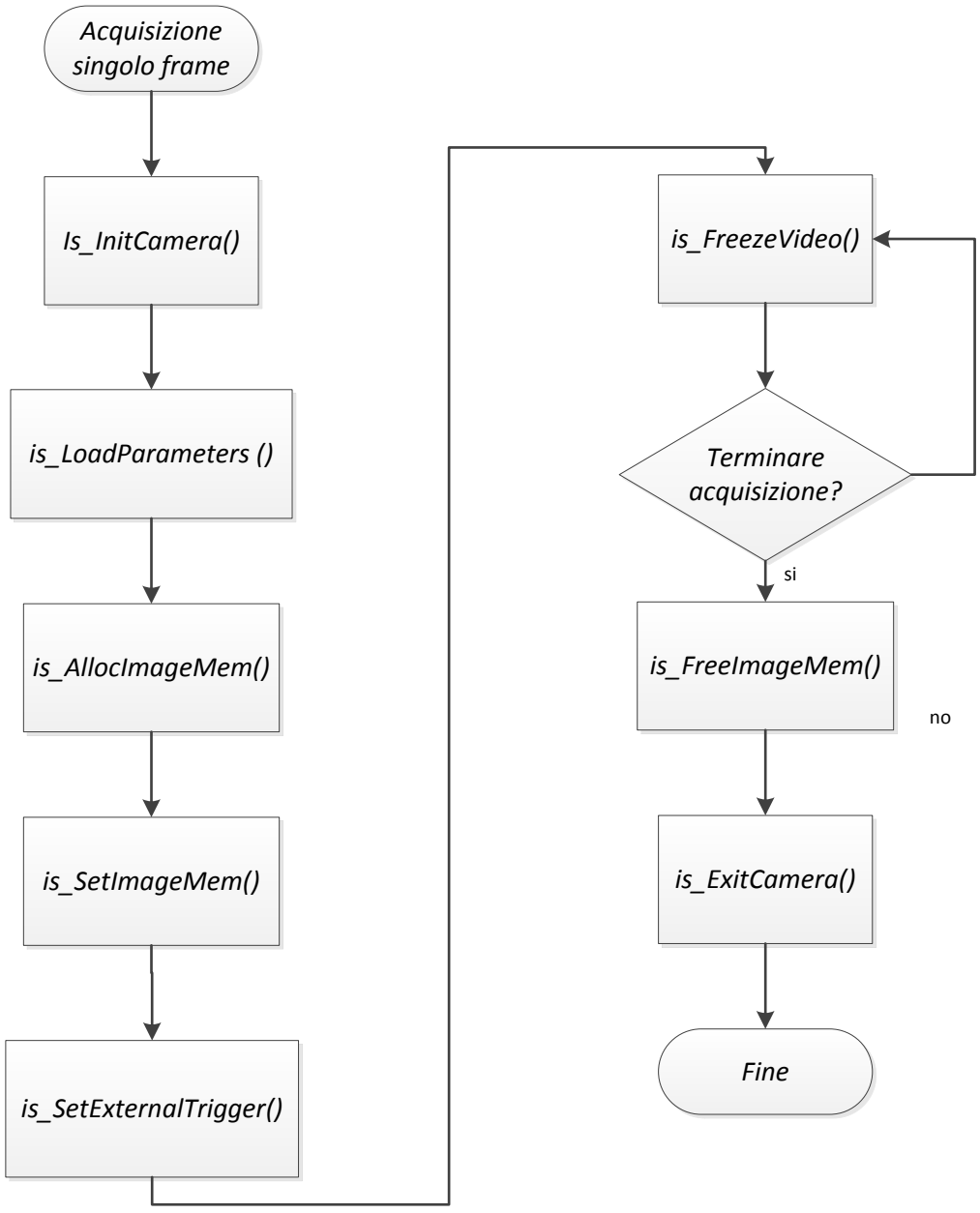


Figura 2.6: Flow chart di un algoritmo di setup camera e uEye e acquisizione dati

Di seguito viene riportata la sintassi e la descrizione delle funzioni in linguaggio C:

1. La funzione *is_InitCamera ()* avvia il driver e stabilisce la connessione alla telecamera. Dopo che l'inizializzazione ha avuto successo, questa funzione assegna l'ID della telecamera (*hCam*). Tutte le funzioni successive richiedono questo ID come primo parametro.

Sintassi:	`INT is_InitCamera (HIDS* hCam, HWND hWnd)`

2. La funzione *is_LoadParameters ()* carica da un file .ini tutti i parametri che sono stati scelti per la telecamera uEye, come ad esempio il tempo di esposizione, il frame rate ed il Pixel Clock. Nel progetto in esame sono stati scelti

```
Pixelclock = 71 [Mhz]
Framerate  = 50  [fps]
Exposure   = 4.75 [ms]
```

Sintassi:	`INT is_LoadParameters (HIDS hCam, char* pFilename)`

3. *isAllocImageMem()* alloca uno spazio immagine avendo a disposizione le dimensioni e la tipologia dell'immagine. Vengono assegnati *ppcImgMem* e *pid*, ovvero l'indirizzo alla zona di memoria ed un ID. La memoria appena allocata non risulta immediatamente utilizzabile, deve essere prima attivata dalla funzione *isSetImageMem().*

Sintassi:	`INT is_AllocImageMem (HIDS hCam,`
	`INT width, INT height, INT bitspixel,`
	`char** ppcImgMem, INT* pid)`

4. *isSetImageMem()* rende la memoria attiva e pronta a immagazzinare dati.

Sintassi:	`INT is_SetImageMem(HIDS hCam,char* pcImgMem,INT pid)`

5. Utilizzando *is_SetExternalTrigger ()*, è possibile attivare la modalità di trigger. L'immagine viene catturata immediatamente quando viene chiamata la funzione *is_FreezeVideo ()*.

Sintassi:	`INT is_SetExternalTrigger (HIDS hCam,` `INT nTriggerMode)`

6. Quando si chiama *is_FreezeVideo ()*, l'immagine catturata viene memorizzata nel buffer di immagine designata da *pcImgMem* e *pid* . Il parametro *Wait* permette di definire se trasmettere immediatamente l'immagine o aspettare che il primo frame si salvato in memoria.

Sintassi:	`INT is_FreezeVideo (HIDS hCam, INT Wait)`

7. Terminata l'acquisizione si libera lo spazio di memora allocato, si rimuove la gestione del driver e si chiude la connessione con la telecamera uEye disattivando *hCam* e rilasciando le strutture dati attivate.

Sintassi:	`INT is_FreeImageMem (HIDS hCam,`
	`char* pcImgMem, INT id)`
	`INT is_ExitCamera (HIDS hCam)`

Per maggiori dettagli sull'utilizzo pratico delle funzioni appena descritte e di tutte le funzioni disponibili si faccia riferimento al manuale disponibile sul sito online di IDS [11].

2.2 Il manipolatore e l'unità di controllo

Questo paragrafo è dedicato alla descrizione del robot COMAU SMART Six sul quale è stato montato il sistema di visione per applicazioni di "inseguimento del profilo di un oggetto metallico". Lo SMART SiX è prodotto dall'azienda italiana COMAU (*COnsorzio MAcchine Utensili*), società del gruppo Fiat con sede a Torino e fa parte della famiglia di robot adatta per applicazioni di manipolazione leggera e saldatura ad arco.

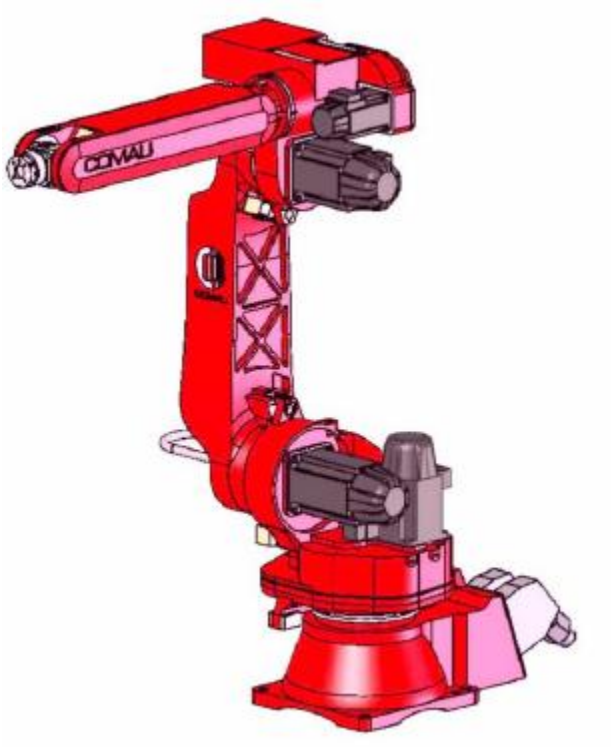


Figura 2.7

La struttura è del tipo antropomorfo con 6 gradi di libertà. La base del robot è fissa e su essa ruota attorno all'asse verticale (asse 1) la colonna che porta il motoriduttore dell'asse 2. Un braccio collega l'asse 2 all'avambraccio ed include i motoriduttori degli assi 3-4-5-6; all'estremità dell'avambraccio si trova il polso. I motori sono del tipo AC brushless ed integrano al loro interno il freno e l'encoder [13].

Tutte le caratteristiche del robot sono riportate in Tabella 2.

Struttura / n° assi	Antropomorfo / 6 assi
Carico al polso	6 kg(1)
Carico supplementare su avambraccio	10 kg(2)
Coppia asse 4	11,7 Nm
Coppia asse 5	11,7 Nm
Coppia asse 6	5,8 Nm
Corsa / (Velocità) – asse 1	+/- 170°(140°/s)
Corsa / (Velocità) – asse 2	+155°/-85°(160°/s)
Corsa / (Velocità) – asse 3	0°/-170°(170°/s)
Corsa / (Velocità) – asse 4	+/-210°(450°/s)
Corsa / (Velocità) – asse 5	+130°/-130°(375°/s)
Corsa / (Velocità) – asse 6	+/- 2700°(550°/s)
Sbraccio massimo orizzontale	1400 mm
Ripetibilità	+/- 0,05 mm
Peso robot	160 kg
Flangia attacco attrezzi	ISO 9409-1-40-4-M6

Motori	AC brushless
Sistema di misura della posizione	con encoder
Potenza totale installata	3 kVA / 4,5 A
Grado di protezione	IP65
Temperatura di esercizio	0 ÷ + 45 °C
Temperatura di immagazzinamento	-40 °C ÷ +60 °C
Colore robot (standard)	Rosso RAL 3020
Posizione di montaggio (3)	A pavimento / Soffitto (Inclinazione max 45°)

Tabella 2

Lo spazio operativo, ovvero l'insieme dei punti che l'organo terminale del manipolatore può raggiungere, è rappresentato in Figura 2.8, mentre in Figura 2.10 si possono vedere la vista del piano Y-Z e nel dettaglio lo schema dall'avambraccio .

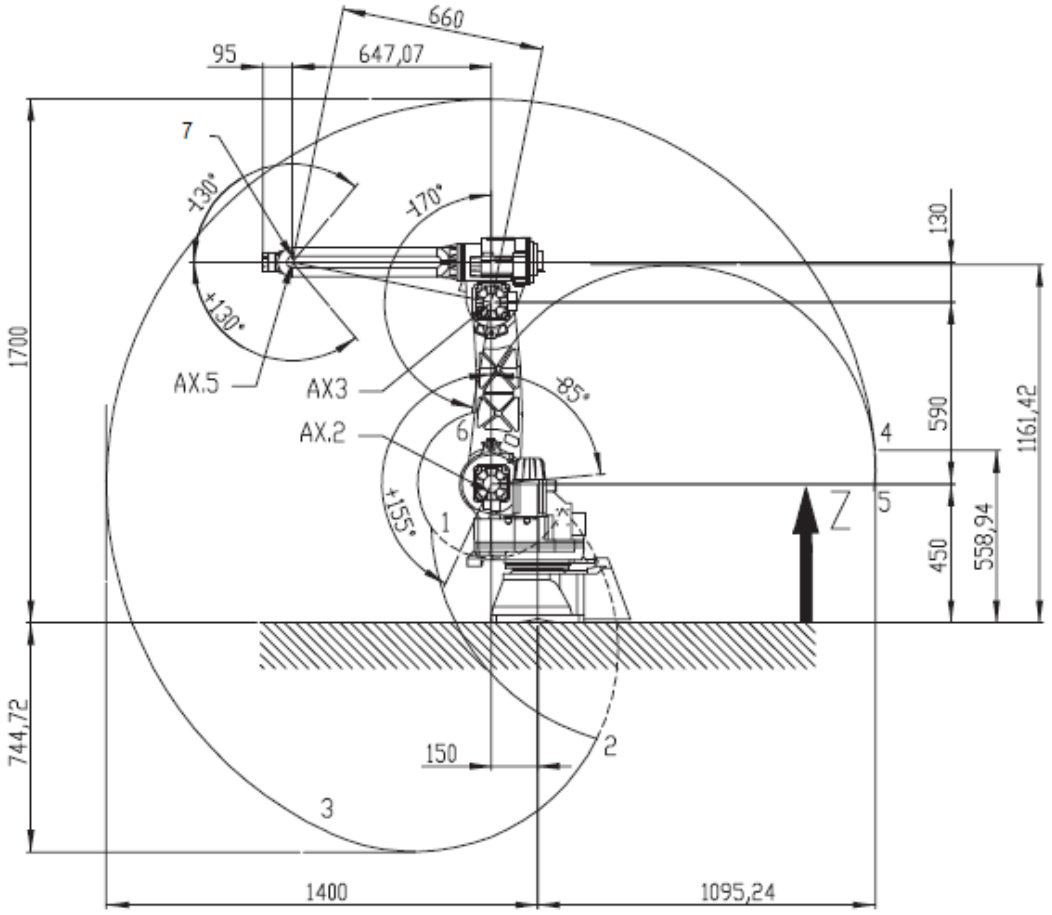


Figura 2.8

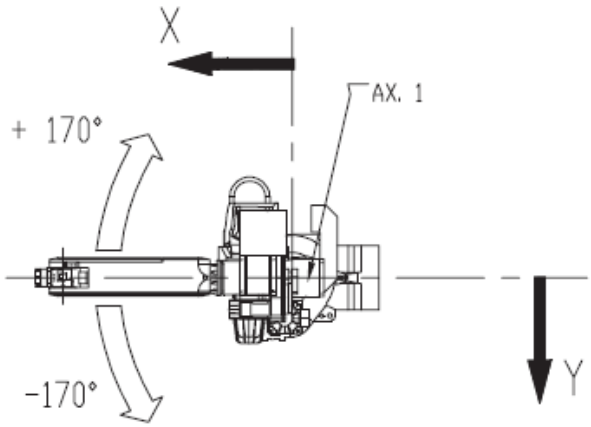


Figura 2.9

Pos	X	Z	Ax.2	Ax.3
	[mm]	[mm]	[deg]	[deg]
1	345,85	308,45	+30°	-170°
2	-192,03	-377,77	+155°	-100°
3	678,27	-682,88	+155°	-11,36°
4	-1095,24	558,94	-85°	-11,36°
5	-1093,69	428,31	-85°	0°
6	45,45	687,32	-85°	-170°

Tabella 3

Giunti in posizione di calibrazione (pos.7)					
Ax 1	Ax 2	Ax 3	Ax 4	Ax 5	Ax 6
0°	0°	-90°	0°	+90°	0°

Tabella 4

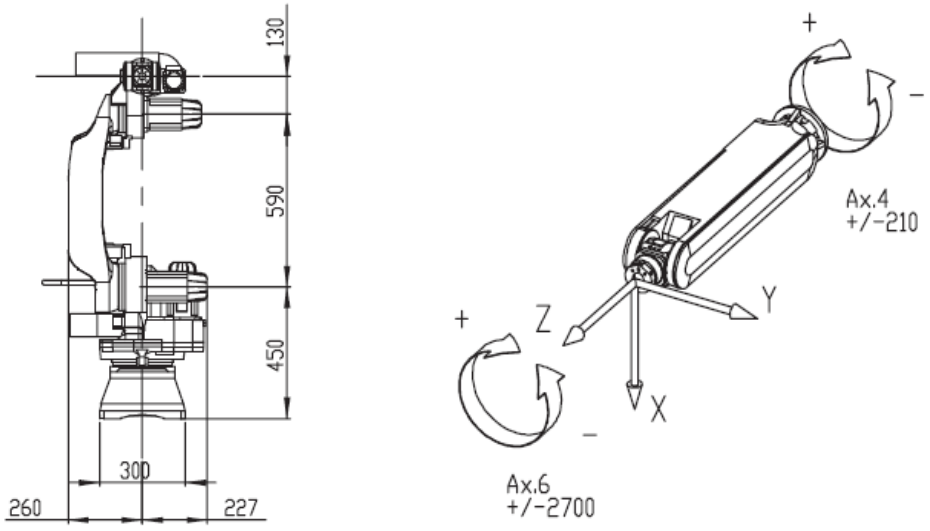


Figura 2.10

Il robot è abbinato all'Unità di Controllo C4G Open. La tecnologia di cui dispone è stata sviluppata da COMAU per semplificare l'uso dei manipolatori industriali nelle università, i centri di ricerca e le piccole e medie imprese. Robustezza e sicurezza sono garantite dai sistemi di supervisione e controllo che sono sempre attivi e che comprendono gli anelli di corrente, di velocità e di posizione dei giunti del manipolatore

Figura 2.11

Un pc esterno Realtime può direttamente comandare il robot utilizzando un protocollo di comunicazione Open Source per generare la traiettoria e/o fornendo i riferimenti di posizione e velocità dei giunti. Il linguaggio di programmazione è il PDL2 che consente l'utilizzo di istruzioni di movimento, di tutte le variabili di sistema C4G, la gestione degli Input/Output e del socket di comunicazione TCP/IP.

2.3 Architettura Linux

Due calcolatori con sistema operativo Linux completano il sistema raffigurato in Figura 2.1. Come accennato nel primo paragrafo il PC connesso all'unità di controllo è un server Realtime versione 10.04.4. La scelta di un calcolatore realtime nasce dalla necessità di fornire tempi di risposta prefissati e garanzie di sicurezza; l'unità di controllo interroga il server ogni 2 milisecondi chiedendo l'invio di un pacchetto che contenga la posizione e la velocità dei giunti del robot. Questa informazione è generata dal sistema di controllo in esecuzione su questa macchina.

In generale i sistemi di controllo robotici che integrano anche i sistemi di visione possono essere di due tipi. Nel caso in cui la visione e la manipolazione vengono

combinate in una tecnica ad anello aperto allora il sistema di visione guida il sistema di controllo posizionale; tale configurazione è denominata L*ook and move*. In caso contrario, se le misure visive possono essere utilizzate direttamente in un anello di retroazione per realizzare il controllo di posizione dell'organo terminale in anello chiuso, si tratta di asservimento visivo o Visual Servoing (VS). Quest'ultimo tipo di controllo è quello utilizzato nel caso in esame.

Le tipologie di tecniche VS si suddividono in due macro categorie:

- Image Based, basato sull'immagine (IBVS)
- Position Based, basato sulla posizione (PBVS)

L'image Based in Figura 2.13 è una legge di controllo basata sull'errore calcolato come differenza tra una determinata caratteristica sul piano immagine corrente e una posizione desiderata sempre sullo stesso piano immagine. Questa tecnica differisce dalla Position Based Visual Servoing Figura 2.12 che, invece di valutare l'errore sul piano immagine, ricostruisce la posizione dell'oggetto nello spazio 3D e con questa impartisce il comando al controllore che fa muovere il robot. In particolare, le features estratte dall'immagine sono usate, in connessione con un modello geometrico del target e con il modello della telecamera, per stimare la postura del target rispetto alla telecamera [14].

Si può comprendere che l'approccio Image Based può ridurre i tempi di calcolo e elimina eventuali errori dovuti alla modellizzazione del sensore di visione e dalla calibrazione delle telecamera.

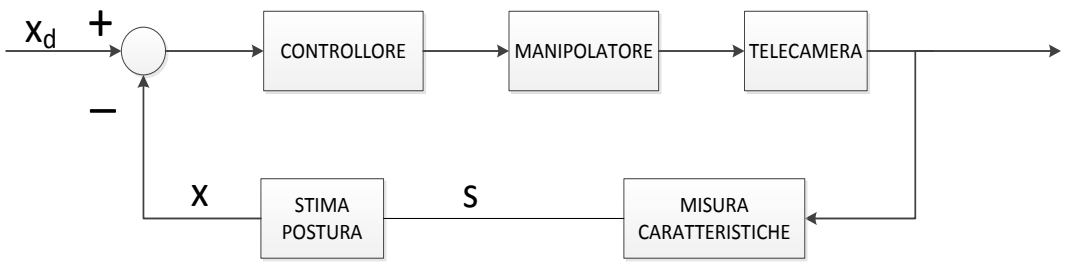
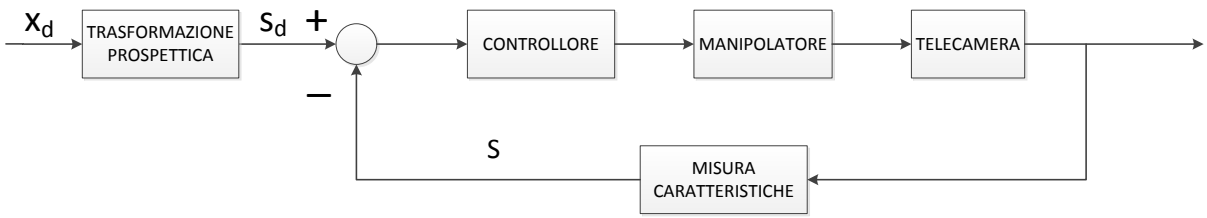


Figura 2.12: Position Based Visual Servoing



x_d = posizione desiderata in coordinate 3D

$s_{d=}$ = posizione desiderata in termini di image features

x = posizione in coordinate 3D

s = posizione in termini di image features

Figura 2.13: Image Based Visual Servoing

Esistono infine gli approcci ibridi che combinano alcuni aspetti del IBVS e PBVS. Nel sistema di controllo considerato l'errore di posizione è valutato sul piano immagine nelle coordinate x e y, mentre la coordinata z viene stimata nello spazio 3D grazie ad una triangolazione laser. In Figura 2.14: Approccio ibrido di un controllo Visual Servoing è mostrato lo schema a blocchi del controllo Visual Servoing adottato.

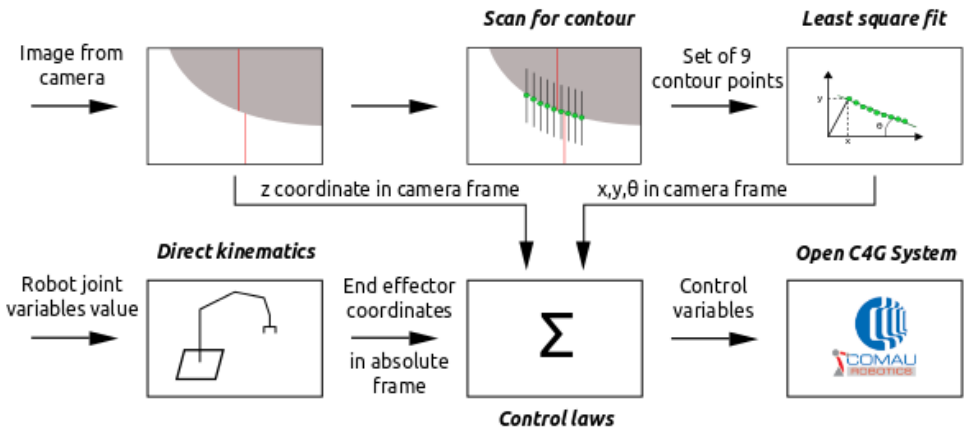


Figura 2.14: Approccio ibrido di un controllo Visual Servoing

Le informazioni utilizzate per il calcolo dell'errore delle coordinate x, y sono valutate e trasmesse al server Realtime tramite una connessione Client/Server di tipo TCP/IP dal software sviluppato in questa tesi ed eseguito sul secondo calcolatore. Su questo secondo PC è installato un sistema operativo Ubuntu Server versione 10.04.4.

2.4 Trasmissione dati Client/Server

L'architettura di rete Client/Server permette di interfacciare un computer client ad un server tramite un protocollo di comunicazione che, nell'applicazione in esame, è il *Transfer Control Protocol* o *TCP*. Tale protocollo permette di creare un canale di comunicazione affidabile tra i due processi garantendo il tempo di consegna e il controllo della congestione di rete evitando la perdita e gli errori dei pacchetti informativi trasmessi.

Con lo scopo di trasmettere la posa del contorno dal computer su cui si effettua l'image analysis al server Linux Realtime che gestisce il controllo del manipolatore, sono state create per entrambi i dispositivi le routine necessarie per la creazione di una connessione TCP ed il trasferimento di dati. Affinché vi possa essere una comunicazione tra i due calcolatori è necessario innanzitutto stabilire una connessione tra il client ed il server. Una volta stabilita vi potrà essere lo scambio di dati, terminato il quale la connessione verrà chiusa. Il canale di comunicazione sarà costituito

da un socket, ovvero la coppia indirizzo IP e porta, sia del mittente che del destinatario.

Una volta che è stato creato il canale di comunicazione (**socket**) il server si deve mettere in ascolto in attesa di una richiesta di connessione da parte di un client (**listen**). Nel momento in cui giunge una richiesta di connessione questa viene accettata dal server il quale instaura una connessione con il client richiedente (**accept**).

Nel processo client, una volta creata la socket, si può avanzare la richiesta di connessione al server (**connect**) ed in caso di esito positivo i due processi possono comunicare. Il client invierà i dati relativi alla posa del contorno, calcolata nelle routine descritte in precedenza, che il server riceverà e che saranno utilizzati dal sistema di controllo (**send receive**). La comunicazione non viene chiusa fin tanto che l'algoritmo è in esecuzione, solo al termine di quest'ultimo il canale di connessione costituito dalla socket verrà chiuso (**close**) [15]. Per una maggior in Figura 2.15 è mostrato lo schema di comunicazione Client/Server.

Nei capitoli seguenti verranno descritti i metodi e gli strumenti software utilizzati nell'image analysis per ottenere le informazioni utili per il calcolo dell'errore del sistema di controllo citato.

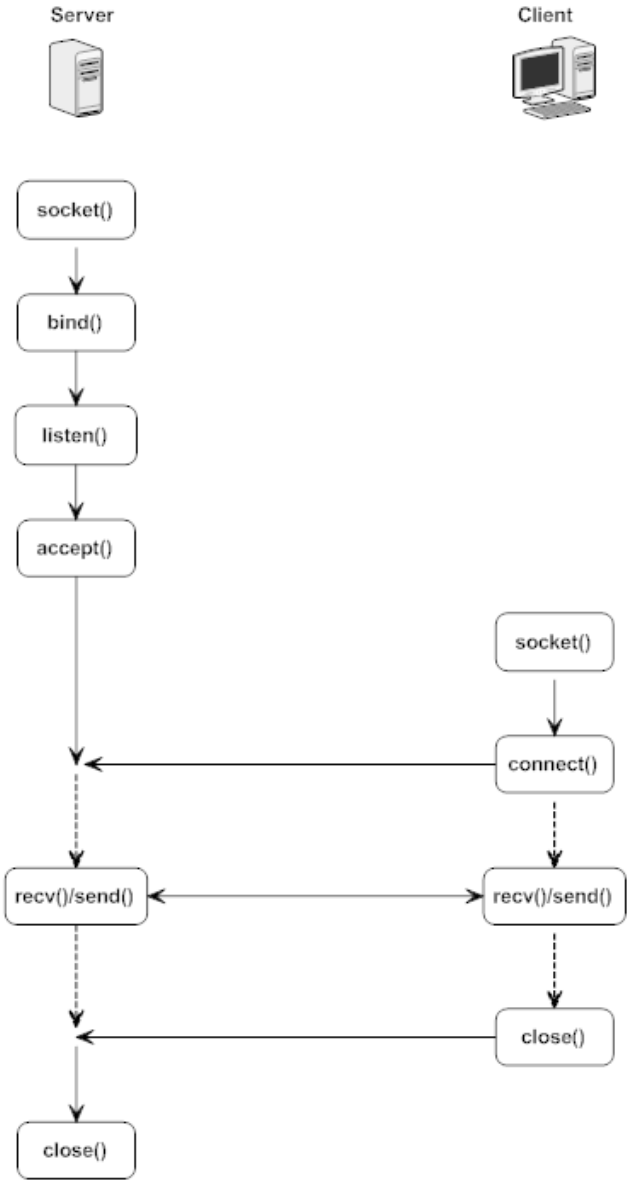

Server
Client
socket()
bind()
listen()
accept()
socket()
connect()
recv()/send()
recv()/send()
close()
close()

Figura 2.15

Capitolo 3

COMPUTER VISION e LIBRERIE OpenCV

La *visione artificiale* è la disciplina che studia modelli e metodi per abilitare le macchine ad acquisire, elaborare e analizzare informazioni estratte da immagini fisse o sequenze video di una scena reale. L'obiettivo è prendere decisioni in maniera automatica o semi-automatica riproducendo su calcolatori elettronici le capacità di percezione e comprensione della realtà della visione umana, quel senso che consente all'uomo di localizzare e riconoscere gli oggetti presenti in una scena e di percepire i rapidi mutamenti del mondo 3D.

La capacità di visione, intesa come acquisizione di immagini, dei sistemi ottici non riscontra particolari problemi grazie all'enorme sviluppo tecnologico in questo campo. I relativi sensori hanno ampiamente superato le capacità di sensibilità, velocità e risoluzione dell'occhio umano. Le problematiche maggiori in questo campo risiedono principalmente nell'interpretazione e nell'utilizzo delle informazioni acquisite da parte di un sistema elettronico. Convertire un'immagine in informazioni come un cervello umano sa fare è un problema tutt'altro che semplice e il campo di ricerca risulta essere ancora giovane, con solo trent'anni di esperienza.

La computer vision è presente e in continuo sviluppo in diversi settori. Esempi sono il campo medico e biomedico, quello della sicurezza con i sistemi di video sorveglianza, quello biometrico con i sistemi di face-detection, quello della modellizzazione tridimensionale e quello dei video games.

3.1 La Computer Vision in ambito industriale

In particolare nel campo industriale la visione artificiale ha riscosso un notevole successo in quanto ha permesso l'abbattimento dei costi e la completa automatizzazione del controllo di qualità, con migliori risultati.

I principali ambiti di applicazione riguardano:

- *Controllo di processo*: la computer vision gioca un ruolo fondamentale nella guida dei robot in quanto permette di controllarne l'orientamento e la posizione nel campo operativo, nonché di rilevare eventi e anomalie per una completa analisi del flusso produttivo.
- *Metrologia*: sfruttando la computer vision si possono effettuare misure automatiche in assenza di contatto tra oggetto e strumento di misura, garantendo accuratezza, ripetibilità e riproducibilità e un vantaggio in termine di riduzione dei tempi di processo.
- *Controllo di qualità*: l'uso di sistemi di computer vision consente il rilevamento ed il riconoscimento di difetti e la verifica del rispetto di tolleranze. Inoltre facilita il conteggio e la classificazione di prodotti in base a predeterminati parametri di qualità (misura, numero e tipologia di difetti).
- *Lettura di caratteri e codici*

Queste ed altre applicazioni vengono concretamente raggiunte grazie allo svolgimento di specifiche operazioni che mettono in gioco la capacità visiva del sistema; tra le più comuni si possono ricordare la *ricostruzione di scene*, il *video tracking*, l'*inseguimento di un contorno*, il *rilevamento di eventi*, l'*apprendimento* inteso come *Machine Learning* ecc. Tutto ciò implica l'integrazione hardware e software delle tecnologie dei sensori ottici nei sistemi di controllo.

3.2 La Computer Vision nel caso di studio

L'integrazione hardware del sistema di visione per il controllo del moto di un robot è stata descritta nel capitolo 3. In questo paragrafo si vogliono esporre le soluzioni a livello software utilizzate nel progetto di edge detection per il contour following di un profilo metallico.

Sono stati valutati due soluzioni software, entrambi adatti al conseguimento dell'obiettivo posto: *MATLAB* e le librerie *OpenCV*.

a) *MATLAB* è un linguaggio di alto livello ed un ambiente interattivo per il calcolo, la visualizzazione e la programmazione numerica. Consente di analizzare dati, sviluppare algoritmi, creare modelli e applicazioni. Il linguaggio, gli strumenti e le funzioni matematiche incorporate consentono di esplorare più approcci e di arrivare a una soluzione più velocemente rispetto all'uso di fogli di calcolo o di linguaggi di programmazione tradizionali quali C/C++ o

Java. È possibile usare MATLAB in un'ampia gamma di applicazioni, tra cui l'elaborazione di immagini e video.

b) Le librerie OpenCV (Open Computer Vision) forniscono gli strumenti, i formati di rappresentazione dei dati e gli algoritmi comunemente utilizzati per l'analisi dell'immagine. Il progetto OpenCV nasce inizialmente da un gruppo di ricerca e sviluppo della Intel; è una libreria disponibile per tutti i sistemi POSIX (Linux/BSD/UNIX/MacOsX) e piattaforme Microsoft Windows scritta in C e C++ e costituita da oltre 500 funzioni utili nel campo dell'image processing e della computer vision. Punti di forza di questa libreria sono la completa portabilità e la capacità di soddisfare le più svariate esigenze di trattamento delle immagini [16].

Le principali motivazioni che hanno spinto ad utilizzare le librerie OpenCV in linguaggio C sono state la completezza e l'alta efficienza. L'enorme disponibilità di funzioni avanzate per l'image processing e la capacità di elaborare le informazioni acquisite e fornire i risultati con tempi di computazione efficienti sono, infatti, requisiti fondamentali per la realizzabilità dell'obiettivo imposto.

3.3 Librerie OpenCV

Nonostante Intel abbia avviato lo sviluppo delle OpenCV, la libreria è "open" e "free" e tutto il suo codice o parte di esso può essere utilizzato o incorporato in altre applicazioni, sia commerciali che di ricerca. I contributi di sviluppo alla libreria provengono dalle più svariate realtà: l'elenco dei credits presente nella pagina ufficiale contiene nomi di ricercatori e docenti universitari di tutto il mondo. Questo aspetto rappresenta la garanzia di buona qualità del codice e degli algoritmi applicati.

3.3.1 Informazioni pratiche

Il sito internet principale a cui si è fatto riferimento è quello creato dal laboratorio di ricerca robotica "Willow Garage" [17], impegnato nello sviluppo di hardware e programmi open source per applicazioni robotiche. Il sito fornisce tutte le istruzioni per l'installazione della libreria OpenCV per tutti i tipi di piattaforma e l'installazione dei pacchetti necessari al suo completo funzionamento.

Per la documentazione completa sulla libreria si è fatto riferimento al sito [18] e a [16].

3.3.2 Organizzazione della libreria

La libreria è suddivisa in molti moduli in continuo sviluppo, ciascuno contenente centinaia di funzioni che permettono di implementare applicazioni di visione in diverse aree (i già citati settori medico, sicurezza, interfaccia utente, calibrazione delle telecamere e robotica). I moduli fondamentali sono:

- CXCORE è la sotto libreria principale e fondamentale. Include tutte le funzioni di inizializzazione delle strutture dati utilizzate con le rispettive inizializzazioni, l'algebra lineare e le funzioni di base per la lettura, la scrittura e la memorizzazione dei dati.
- CV contiene le funzioni che riguardano l'analisi delle immagini, tutte funzioni di image processing e include le routine di object detection.
- HIGHGUI Include funzioni GUI, funzioni per il caricamento e il salvataggio di file immagine, per la gestione delle telecamere e l'acquisizione video.
- MLL (Machine Learning Library). La computer vision spesso va a pari passo con l'apprendimento automatico (o machine learning). MLL OpenCV fornisce gli strumenti fondamentali per qualsiasi problema di apprendimento.

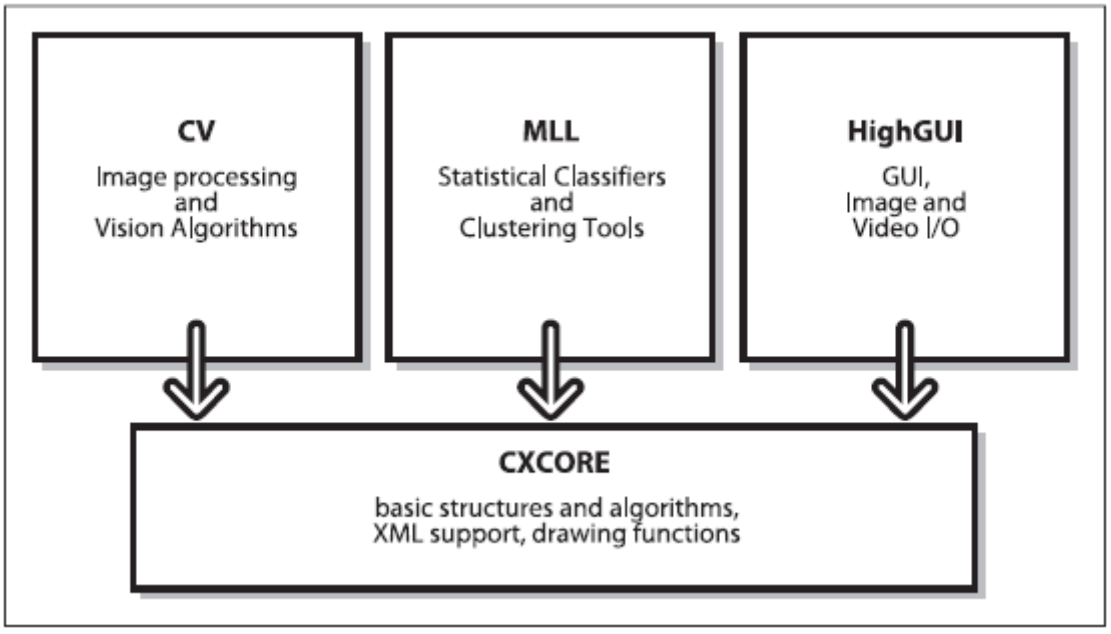


Figura 3.1

3.3.3 Panoramica delle funzioni più comuni

Per introdurre le principali strutture utilizzate e le funzioni più comuni viene riportato un esempio con flow chart (in Figura 3.2) che rappresenta un algoritmo di acquisizione, creazione, modifica e salvataggio di immagini. In particolare questo

algoritmo acquisisce da disco un immagine a colori e ne crea una copia in scala di grigi salvandola su disco.

Come si può notare tutte le funzioni della libreria OpenCV sono accomunate dal prefisso "*cv*", mentre i nomi delle strutture di dati utilizzate hanno prefisso "*Cv*" (con la C maiuscola) ad eccezione della struttura *IplImage*, ereditata dalla libreria IPL di Intel. Verrà presentata di seguito la sequenza di istruzioni e le relative descrizioni. Per maggiori dettegli sulla sintassi delle singole funzioni e i parametri disponibili si richiama la documentazione ufficiale della libreria OpenCV [18].

1. *Struttura IplImage*
   È l'acronimo di Image Processing Library Image, ossia il formato standard utilizzato da Intel per rappresentare un'immagine nelle API IPL e OpenCV. Tramite questa struttura dati è possibile gestire completamente tutto ciò che ruota intorno alle immagini, come caricamento e salvataggio, conversione di formato, elaborazione, filtraggio e visualizzazione. Di seguito vengono esplicitati tutti i campi della struttura commentando quelli piu importanti.

```
typedef struct IplImage {
      int nSize;
      int ID;
      int nChannels;//numero di canali dell'immagine
      int alphaChannel;
      int depth;//definisce lo spazio di memoria
                  allocato per pixel
      char colorModel[4];
      char channelSeq[4];
      int dataOrder;
      int origin;
      int align;
      int width;//larghezza dell'immagine (in pixel)
      int height;//altezza dell'immagine (in pixel)
      struct IplROI* roi;//selezione sotto immagine
      struct IplImage* maskROI;//puntatore a sotto
                                 immagine selezionata
      void* imageId;
      struct IplTileInfo* tileInfo;
      int imageSize;
      char* imageData;//puntatore ai dati
                        dell'immagine
      int widthStep;//passo in bit di larghezza (in
                      base alla depth)
      int BorderMode[4];
      int BorderConst[4];
      char* imageDataOrigin;
} IplImage;
```

Figura 3.2

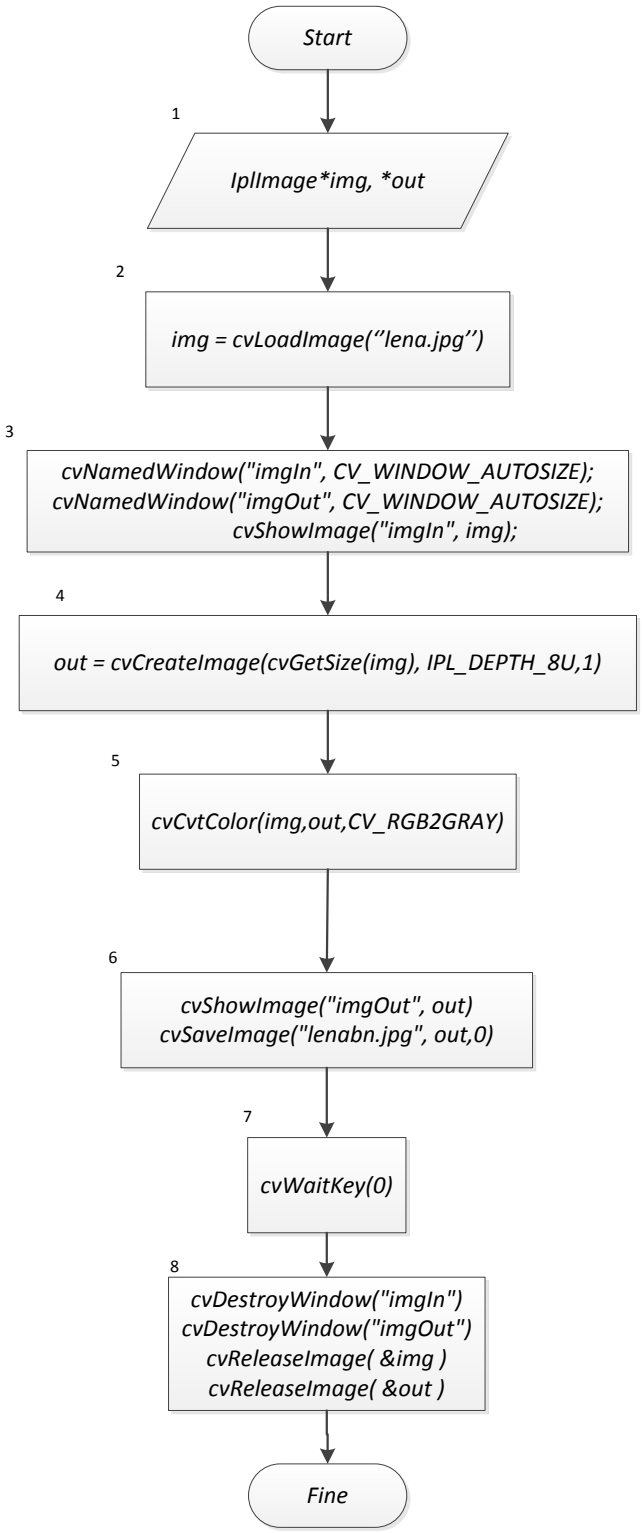


Figura 3.3: Flow chart di un algoritmo di acquisizione, creazione e modifica di immagini con OpenCV

2. `cvLoadImage()`
   questa è una funzione di alto livello che carica l'immagine e ne determina il formato basandosi sul nume del file. Automaticamente alloca la memoria necessaria per la struttura dati. *CvLoadImage* Può acquisire una vasta varietà di formati immagine, che include BMP, DIB, JPEG, JPE, PNG, e TIFF. La funzione restituisce un puntatore alla struttura dati *IplImage* allocata in memoria.
3. *cvNameWindows()* e *cvShowImage()*
   cvnameWindows() è un'altra funzione di alto livello che permette di mostrare a video una finestra che può contenere un immagine. Viene assegnato un nome alla finestra, ad esempio "ImgIn", che sarà utilizzato dalle chiamate che interagiscono con essa, come ad esempio cvShowImage() che richiede il nome della finestra e la struttura IplImage per caricare l'immagine nella finestra assegnata. Il secondo parametro di cvNameWindows() definisce infine le proprietà della finestra.
4. *cvCreateImage()*
   Questa routine alloca un struttura dati immagine assegnando la dimensione dell'immagine, il pixel depth in bit e il numero di canali dell'immagine che possono essere 1 o 3.
5. *cvCvtColor()*
   cvCvtColor() converte un'immagine a colori in scala di grigi. Sostituendo il terzo parametro della funzione è possibile effettuare il processo inverso.
6. *cvSaveimage()*
   In questa fase dell'algoritmo viene mostrato a video la nuova immagine in scala di grigi. Con la funzione cvSaveImage() è possibile salvare su disco l'immagine passata assegnando il nome e un formato tra quelli disponibili.
7. *cvWaitKey()*
   questa funzione chiede al programma di interrompersi per un tempo variabile. Se viene passato un valore positivo il programma aspetterà per un intervallo di tempo pari a questo valore in millisecondi. Se il valore è pari a zero o negativo il programma aspetterà fino alla pressione di un pulsante da tastiera.

8. *cvDestroyWindow()* e *cvReleaseImage()*
   Entrambe le funzioni permettono di liberare la memoria allocata dalle funzioni di creazione immagine e di finestra.

Il risultato finale dell'algoritmo eseguito al calcolatore è mostrato in Figura 3.5: Immagine processata.

Figura 3.4: Immagine di origine

Figura 3.5: Immagine processata

3.3.4 Codice completo dell'esempio

```
#include <cv.h>
#include <highgui.h>

int main(int argc, char* argv[])
{
	// Apri il file "lena.jpg".
	IplImage* img = cvLoadImage("lena.jpg",CV_LOAD_IMAGE_UNCHANGED);
	if (!img) {
		printf("Errore: Non é stato possibile aprire il file  \n");
		exit(1);
	}
	cvNamedWindow("imgIn", CV_WINDOW_AUTOSIZE);
	cvShowImage("imgIn", img);

	IplImage* out=cvCreateImage(cvGetSize(img), IPL_DEPTH_8U,1);

	cvCvtColor(img,out,CV_RGB2GRAY);

	// Salva l'immagine in scala di grigi in un file
	cvSaveImage("lenabn.jpg", out,0);

	cvNamedWindow("imgOut", CV_WINDOW_AUTOSIZE);
	cvShowImage("imgOut", out);

	cvWaitKey(0);

	// Libera le risorse
	cvDestroyWindow("imgIn");
	cvDestroyWindow("imgOut");
	cvReleaseImage( &img );

	return 0;
}
```

Capitolo 4

SVILUPPO DELL'ALGORITMO DI VISIONE

Questo capitolo illustra le parti principali in cui si articola l'algoritmo di edge detection per l'identificazione online del contorno di un pezzo metallico, e spiega come viene effettuato il calcolo dei parametri necessari al sistema di controllo Visual Servoing per eseguire il contour following. Data la molteplicità delle possibili soluzioni al problema, saranno descritte esclusivamente quelle metodologie che hanno portato allo sviluppo di un algoritmo in codice C che soddisfi le richieste di ottimizzazione dei tempi di calcolo.

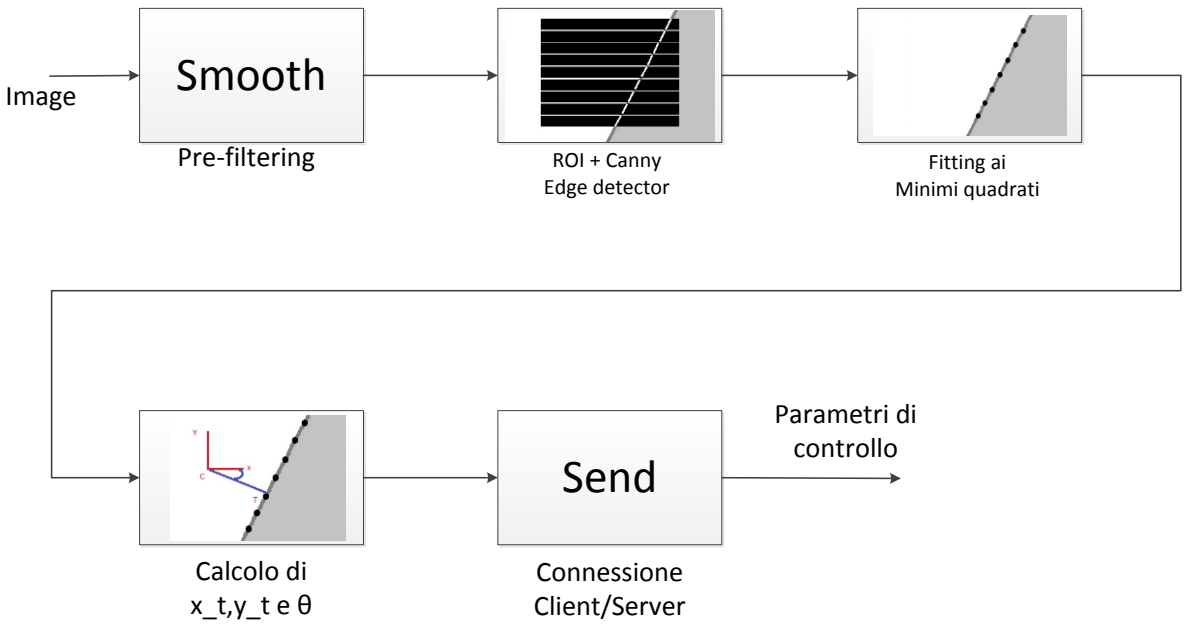


Figura 4.1: Schema del processo di sviluppo dell'algoritmo

La Figura 4.1 fornisce una panoramica dei passi seguiti. L'algoritmo è stato formulato secondo una logica che prende spunto dallo studio di J. De Schutter e J. Baeten [6] presentato nel primo capitolo e mira ad acquisire ciclicamente un frame video su cui effettuare l'image analysis.

Nello specifico, gli obiettivi dell'algoritmo sono:

1. estrarre un set di punti dal profilo del workpiece nel piano immagine (*edge detection*)
2. ottenere la retta di regressione lineare dai punti estratti (*fitting dei dati*)
3. valutare la distanza minima e l'orientamento della retta interpolante rispetto al centro immagine (*posa del contorno*)

Una volta completata questa operazione, i valori di distanza minima e orientamento sono trasmessi, tramite connessione Client/Server di tipo TCP, al server Realtime che gestisce il sistema di controllo Visual Servoing.

4.1 Edge detection

L'interpretazione di immagini acquisite consente di ottenere informazioni utili per il raggiungimento dell'obiettivo di controllo, ovvero il contour-following. Queste informazioni prendono il nome di *image features*, o caratteristiche dell'immagine, e sono ottenute selezionando, con opportuni metodi prentati nel primo capitolo, solo una parte dei dati messi a disposizione dal sensore ottico.

Il primo passo è quello di effettuare un pre-filtraggio o preprocessing. Qui viene utilizzata la tecnica di smoothing, che consiste nello sfocare un'immagine sia per far emergere oggetti di interesse, nel caso in esame il workpiece, che per ridurre dettagli irrilevanti come rumori. La principale fonte di rumore riscontrata nelle fasi di progetto è stata l'instabilità della sorgente luminosa, dovuta all'illuminazione ambientale (luce solare) e all'illuminazione artificiale a cui si aggiungono in misura minore errori di trasmissione dei dati.

Lo smoothing viene effettuato da un *filtro mediano* che calcola la media dei valori dei pixel di un'intorno in modo che gli oggetti piccoli vengono inglobati nello sfondo mentre gli oggetti grandi diventano più semplici da individuare.

La funzione C, inserita nella libreria OpenCV, che effettua questo processo è *cv_Smooth():*

sintassi:
```
void cvSmooth(const IplImage* src, IplImage* dst,
              int smoothtype=CV_MEDIAN, int param1=3,
              int param2=0, double param3=0, double
              param4=0)
```

Per maggiori dettagli sui parametri si consulti la relativa documentazione [18].

Il risultato di questa tecnica fornisce l'immagine di input su cui effettuare l'identificazione e la visualizzazione del contorno del pezzo. Gli approcci seguiti per affrontare questo problema sono stati due.

Il primo prevede l'utilizzo di funzioni specifiche della libreria OpenCv come *cv_FindCountour()*, la quale analizza tutti contorni dell'immagine e li immagazzina in una struttura dati. Poiché l'obiettivo di questo passo è incentrato sulla sola estrapolazione di un set di punti del contorno del pezzo nell'intorno del centro immagine, si è scelto un secondo approccio computazionalmente più efficiente, di seguito descritto.

Dall'immagine pre-filtrata si scelgono un numero di regioni di interesse (ROI) pari al numero di punti da estrarre, allo scopo di restringere il campo d'azione delle funzioni presenti in OpenCV a delle sotto-aree del frame totale. Risulta essere di grande importanza la scelta sia delle dimensioni delle ROI, sia delle loro posizione all'interno delle spazio immagine. Seguono le funzioni base per la definizione di una ROI.

sintassi:
```
void cvSetImageROI(IplImage* image, CvRect rect)

CvRect rect = cvRect( int x, int y,
                      int width, int height )
```

Per quanto riguarda l'estensione delle aree, l'idea è quella di disporre di ROI con altezza fissata e larghezza dinamica, per poter essere ampie a sufficienza per contenere il contorno del pezzo nelle fasi iniziali, quando la posizione del contorno non è ancora stata rilevata, e allo stesso tempo essere ridotte una volta identificato il contorno, per minimizzare i tempi di esecuzione delle funzioni che saranno applicate in seguito.

Disponendo le ROI su linee orizzontali così da ottenere punti equidistanti verticalmente l'uno dall'altro, si applica iterativamente ad ognuna l'algoritmo di Canny.

sintassi:	`void cvCanny(const IplImage* image, IplImage* out, double threshold1, double threshold2, int aperture_size=3)`

Terminata la procedura si ottiene un'immagine con ROI binarie (Figura 4.2), dove ciascun pixel all'interno di ogni area di interesse è marcato come appartenente o non appartenente ad un contorno.

Per estrapolare i punti di interesse da ogni singola ROI si considerano tutti i punti posti sull'ordinata del centro dell'area e si valutano le ascisse dei punti corrispondenti ai pixel bianchi del contorno. Le coordinate estrapolate da ogni regione vengono salvate all'interno di una struttura dati appositamente dichiarata.

$$X_p = \begin{bmatrix} x_{p1} \\ x_{p2} \\ \dots \\ x_{pn} \end{bmatrix} \; Y_p = \begin{bmatrix} y_{p1} \\ y_{p2} \\ \dots \\ y_{pn} \end{bmatrix}$$

Per una maggior comprensione viene riportata in Figura 4.2 un'immagine che mostra il risultato alla fine di questa procedura. Nel frame si nota il profilo curvo di una razza del cerchione usato per l'esempio. Nel centro immagine il set di ROI binarie che contengono il profilo del cerchione identificato incrociato da delle rette orizzontali passanti per centro delle aree messe in evidenza per la comprensione del passaggio relativo all'estrapolazione del set di punti del contorno.

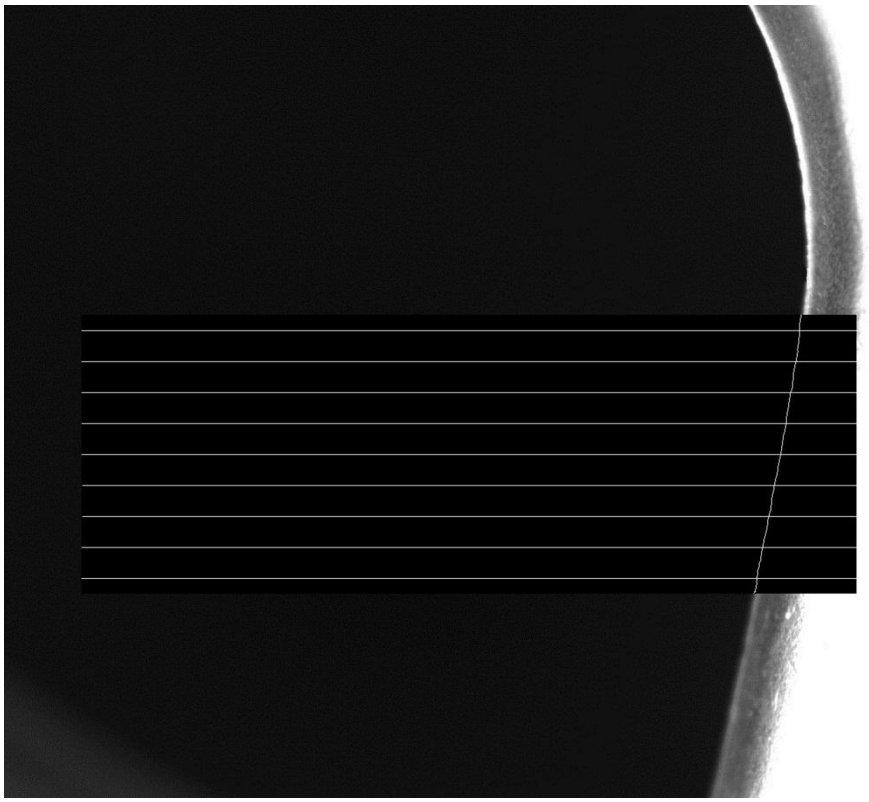


Figura 4.2

4.2 Fitting ai minimi quadrati

Disponendo della struttura dati contenente le coordinate del set di punti del contorno si effettua un fitting ai minimi quadrati per ottenere la retta interpolante dalla quale ricavare i parametri di controllo.

Lo scopo è quello di determinare una funzione lineare che sintetizzi opportunamente l'andamento del contorno. Nel caso dei minimi quadrati tale funzione è detta *retta di regressione* di Y rispetto a X

$$Y = m\,X + q \qquad (1)$$

ed è determinata univocamente se *n*, il numero di dati disponibili, è superiore al numero di parametri, in questo caso due.

Considerate le *n* coppie (x_{pi}, y_{pi}) allora i coefficienti sono:

$$m = \frac{n\sum(x_{pi}y_{pi}) - \sum x_{pi}\sum y_{pi}}{n\sum x_{pi}^2 - \sum(x_{pi})^2} \qquad q = \frac{\sum y_{pi}\sum x_{pi}^2 - \sum x_{pi}\sum(x_{pi}y_{pi})}{n\sum x_{pi}^2 - \sum(x_{pi})^2}$$

Per implementare questo metodo nell'algoritmo sviluppato in linguaggio C viene importata la libreria numerica *GNU Scientific Library (GSL)*, una libreria open source che fornisce una vasta gamma di procedure matematiche, tra cui il fitting ai minimi quadrati di cui viene riportata la sintassi [19].

sintassi:
```
gsl_fit_linear (const double * X, const int xstride,
                const double * Y, const int ystride,
                int n, double * q, double * m,
                double * cov00, double * cov01,
                double * cov11, double * sumsq)
```

Avendo a disposizione l'equazione della retta interpolante (1) si valuta la retta perpendicolare e passante per il centro immagine

$$Y = -1/m\, X$$

e risolvendo il seguente sistema lineare di seguito riportato è possibile ottenere le coordinate (x_t, y_t) del punto T, intersezione tra la retta interpolante e la sua perpendicolare.

$$\begin{cases} Y = m\, X + q \\ X = -m\, Y \end{cases} \qquad \begin{cases} Y = -m^2 Y + q \\ X = -m\, Y \end{cases} \qquad \begin{cases} Y = y_t = \dfrac{q}{1+m^2} \\ X = x_t = -\dfrac{mq}{1+m^2} \end{cases}$$

$$\theta = -\, arctan\,(1/m)$$

È possibile notare che l'equazione della retta perpendicolare è stata esplicitata in funzione di *Y* per evitare che degeneri per *m*=0. In Figura 4.3 è riportato il risultato ottenuto al termine di questa nuova procedura per far comprendere.

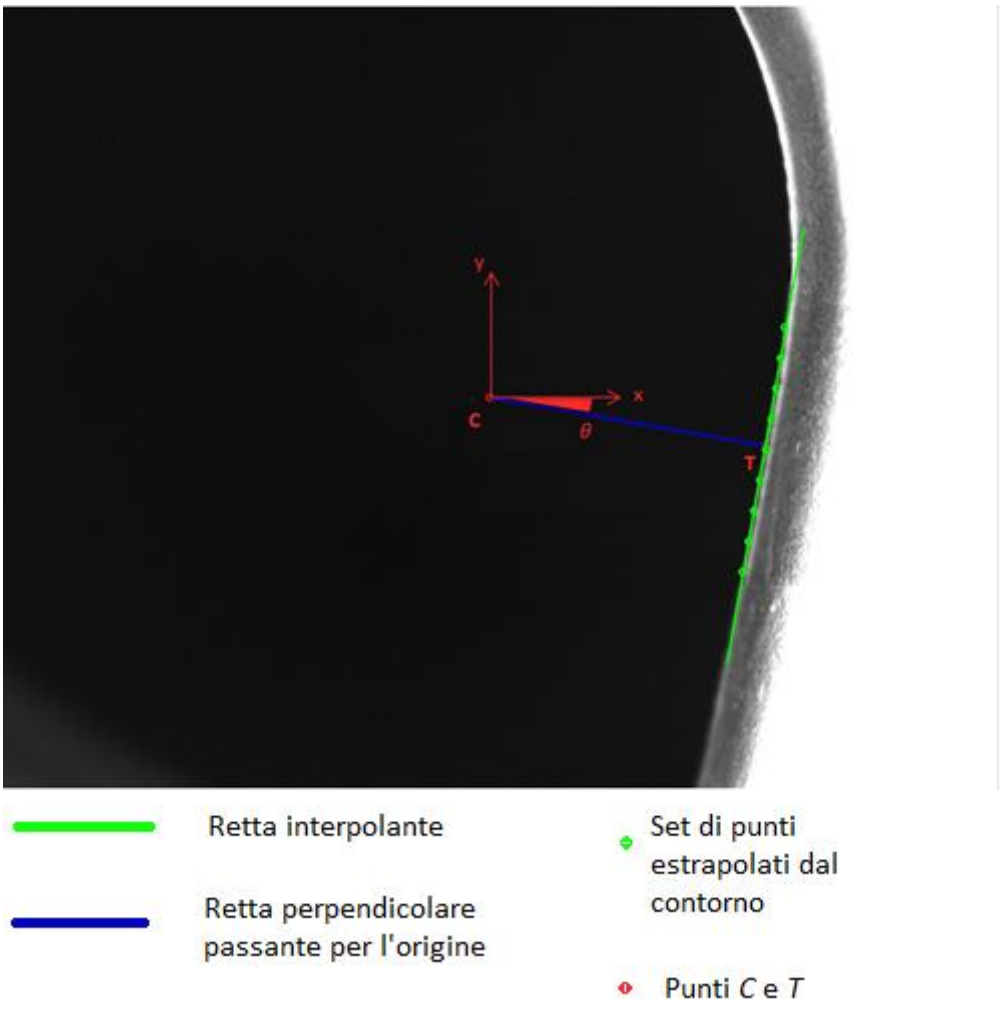


Figura 4.3

Le coordinate del punto *T* (x_T, y_T) insieme all'angolo θ permettono di definire la distanza minima dal centro immagine *C* e l'orientamento rispetto all'asse delle ordinate, ovvero la posa del contorno.

Questi parametri calcolati online vengono utilizzati dal controllo Visual Servoing per controllare il movimento del robot durante il contour following.

4.3 Trasmissione dati Client/Server

Con lo scopo di trasmettere la posa del contorno dal computer su cui si effettua l'image analysis al server Linux Realtime che gestisce il controllo del manipolatore, sono state create per entrambi i dispositivi le routine necessarie per la creazione di una connessione TCP ed il trasferimento di dati.

Semplici esempi di Client/Server reperibili in letteratura sono stati la base delle funzioni C appositamente create per tale scopo. A queste si è aggiunta la *serializzazione* della struttura dati della posa del contorno grazie all'inserimento della libreria *TPL* [20]. Questo miglioramento consente di gestire autonomamente la codifica e la decodifica dei dati prima e dopo il trasferimento dei pacchetti. Il vantaggio principale è la possibilità, se necessario, di cambiare la struttura dati oggetto della comunicazione apportando semplici modifiche al posto di sostituire completamente le istruzioni di codifica e decodifica.

4.4 Valutazione del tempo medio

L'algoritmo di visione è stato corredato delle istruzioni necessarie a valutare il tempo medio di un ciclo durante l'esecuzione, al fine di verificare a posteriori se l'applicazione sviluppata soddisfi le richieste riguardanti l'ottimizzazione dei tempi di calcolo.Nello specifico,un ciclo di esecuzione è composto da:

- L'acquisizione delle immagini
- L'elaborazione dei dati
- Il trasferimento tramite connessione Client/Server

A tale scopo si è resa necessaria l'introduzione della libreria *sys/time.h* e della struttura *Timeval* che consentono di valutare un intervallo di tempo con la risoluzione del microsecondo [21].

Sintassi:

```
int gettimeofday(struct timeval *tp, NULL);

struct timeval
{
      time_t tv_sec;            /* Seconds.  */
      long int tv_usec;         /* Microseconds.  */
}
```

La funzione *gettimeofday()* permette di ottenere il tempo corrente espresso come secondi e microsecondi. La procedura di calcolo del tempo medio di un ciclo di esecuzione (T_m) consiste nell'ottenere il tempo corrente all'inizio (T_i) e alla termine dell'esecuzione dell'algoritmo (T_f) per avere a disposizione l'intera durata di esecuzione dell'algoritmo ($T = T_f - T_i$). Una variabile contatore (n) viene incrementata di un'unità ad ogni ciclo e infine la media aritmetica fornisce il risultato richiesto:

$$T_m = \frac{T}{n}$$

Capitolo 5

RISULTATI SPERIMENTALI

In questo capitolo verranno descritte le prove sperimentali che sono state eseguite al fine di valutare la bontà dell'algoritmo sviluppato in questo lavoro di tesi. Prima di testare l'intero anello di controllo per il contour following si vuole verificare il corretto funzionamento dell'applicazione. Per questo motivo si effettuano due tipi di prove:

- una prima valutazione dell'algoritmo in anello aperto con il manipolatore in controllo manuale
- prove complete con il controllo Visual Servoing, solo dopo aver appurato la correttezza del software

5.1 Valutazione con controllo manuale

Sul server Realtime viene caricato un semplice programma per verificare la corretta trasmissione dei dati generati dall'algoritmo di visione.

I parametri di setup dell' uEye camera necessari per avere il miglior compromesso tra attenuazione dei rumori, buona qualità dell'immagine e minimizzazione del tempo di esposizione sono:

- Pixelclock = 71 [Mhz] (valore massimo disponibile)
- Framerate = 50 [fps] (valore massimo disponibile)
- Exposure = 4.75 [ms]

Il manipolatore robotico viene messo in movimento da un operatore mediante teach pendant. Vengono vincolate la coordinata z, fissata ad un valore di 0.72 metri, e gli angoli di rotazione φ e ψ dell'organo terminale. Le coordinate libere per il moto sono x, y e l'angolo θ dell'end effector. Il robot lavorerà perciò su un piano parallelo al piano di lavoro.

Con questi gradi di libertà è possibile seguire manualmente il contorno di un pezzo metallico generico mostrato in Figura 5.1 in cui è stato evidenziato il profilo inseguito.

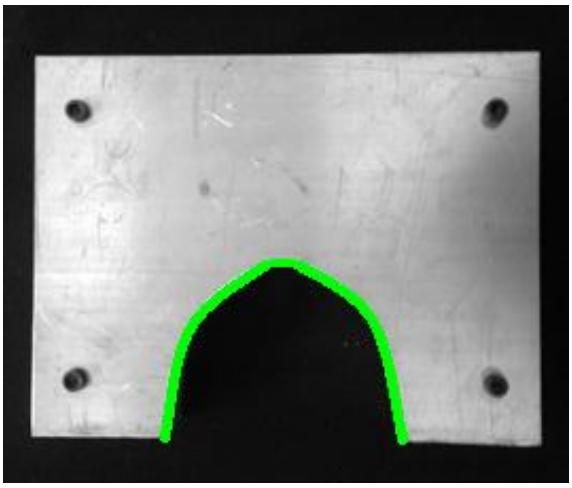


Figura 5.1: Profilo di un generico pezzo metallico

Vengono riportati nelle figure seguenti alcuni frame prelevati della prova ad intervalli di tempo successivi. Nel corso dell'intero test il set di punti è stato estratto correttamente dal processo di edge detection. Le caratteristiche ideali del pezzo (superficie chiara e piana, spigoli del profilo evidenti e ad angolo retto) hanno consentito un'identificazione netta del contorno e i disturbi luminosi esterni (ambientali e artificiali), se pur presenti, non hanno alterato le misure.

Sarà possibile definire in seguito il grado di robustezza ai disturbi verificando i risultati della prova di inseguimento del profilo del cerchione.

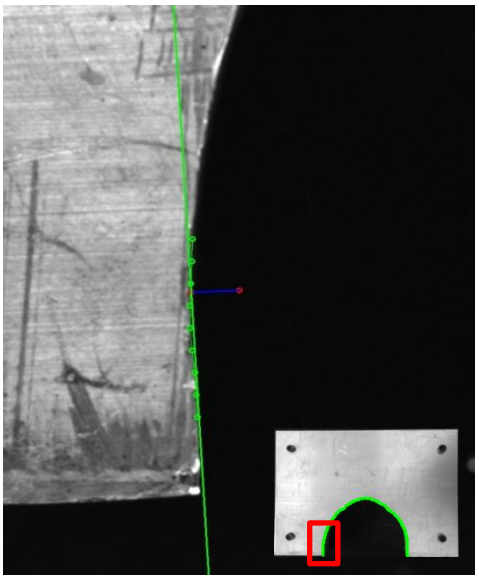


Figura 5.3

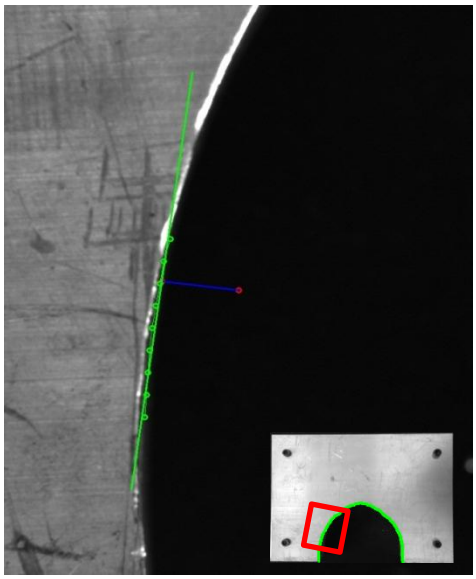


Figura 5.2

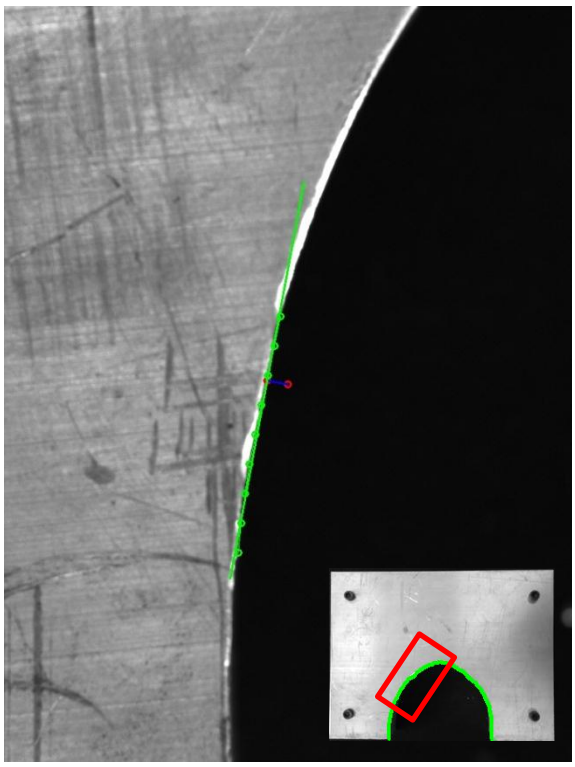


Figura 5.5

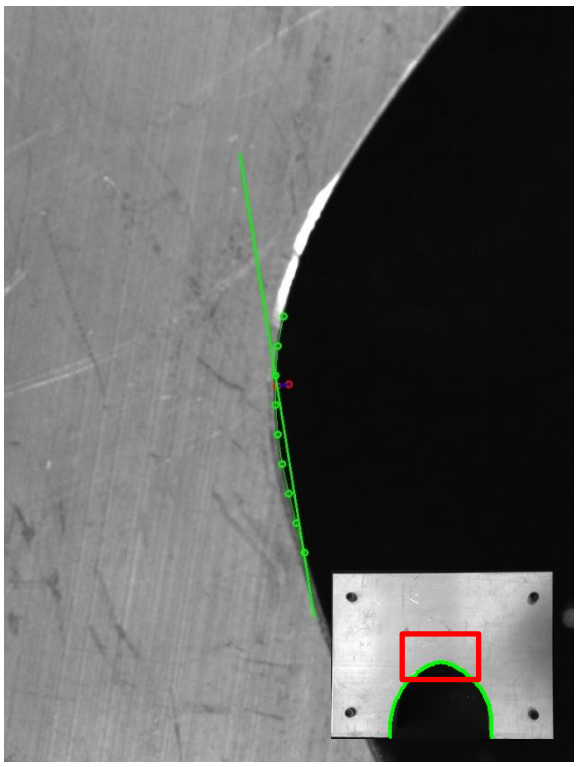


Figura 5.7

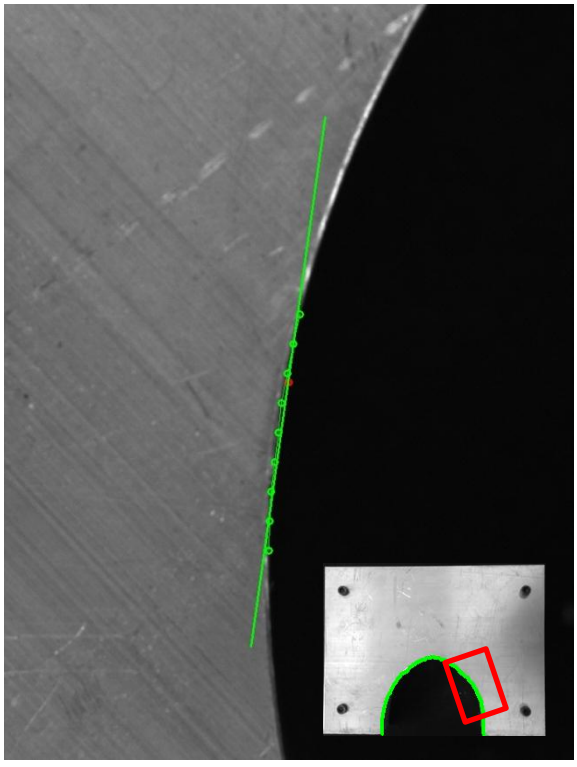


Figura 5.4

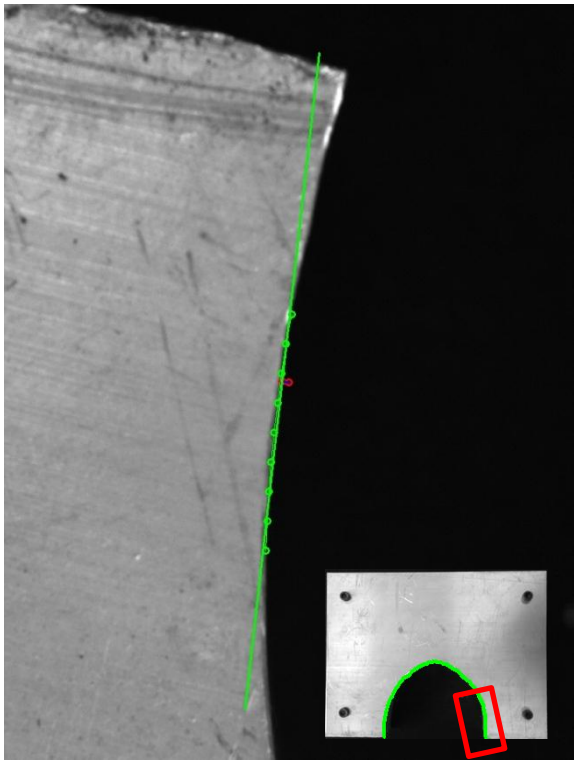


Figura 5.6

Al termine del test, il tempo medio di un ciclo di esecuzione è di 30 millisecondi. Questi sono così suddivisi:

- ≈ 26ms per l'acquisizione delle immagini in modalità Trigger mode. Di questi sono prefissati il tempo di campionamento della camera (un frame ogni 20ms) e l'exposure time pari a 4.75ms. Questo significa che il trasferimento, l'allocamento della memoria e il caricamento dei dati non influiscono significativamente sul tempo di acquisizione.
- ≈ 4ms per l'image analysis, processo che comprende l'operazione di filtraggio del rumore, l'edge detection, il fitting dei dati estrapolati e la valutazione della posa del contorno.
- ≈ 0.5 ms è in media il tempo di invio dei dati tramite connessione client/server di tipo TCP/IP

Da questi risultati possiamo affermare come accennato in precedenza che l'algoritmo di visione sfrutta a pieno le potenzialità della telecamera. Sono quindi soddisfate le richieste di ottimizzazione dell'algoritmo.

5.2 Contour following di un profilo metallico

Valutate l'efficacia e la stabilità di funzionamento dell'algoritmo, il software è stato inserito nel controllo per il contour following. Lo schema a blocchi e i frame in fig ### mostrano una modifica effettuata al software: il contorno viene identificato orizzontalmente sul piano dell'immagine e non più in verticale. Questo per migliorare l'integrazione tra il sistema di visione e la triangolazione laser, utile al sistema di controllo Visual Servoing di tipo ibrido per stimare la coordinata z della terna dell'end effector in coordinate cartesiane.

Viene riportato in Figura 5.8 lo schema a blocchi del regolatore.

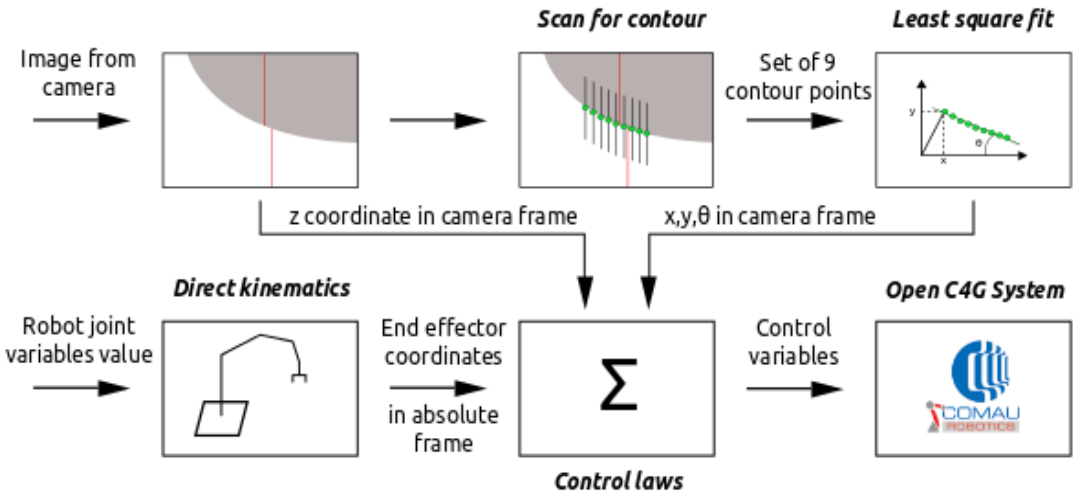


Figura 5.8

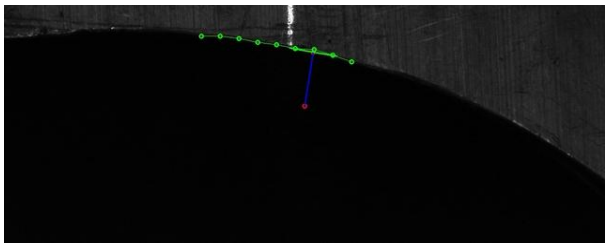
I riferimenti di posizione e orientamento del contorno sono:

- $x_T^\circ = y_T^\circ = 0$ (coordinate del centro camera)
- $\theta^\circ = 90°$ (parallelo all'asse delle ascisse del centro camera)

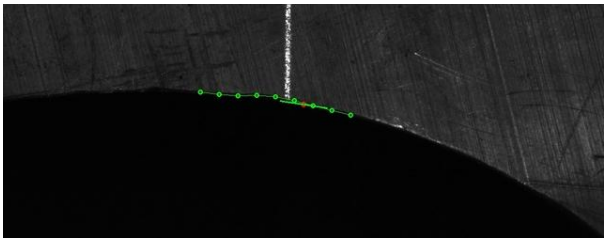
Questa prova ha nuovamente come oggetto dell'inseguimento il profilo metallico generico in Figura 5.1 ed il setup sperimentale è il medesimo del test precedente.

Si può osservare dalla sequenza di frame mostrati di seguito e prelevati, uno ogni 50 cicli di esecuzione, che il profilo è stato correttamente identificato e l'errore azzerato dopo la prima cattura.

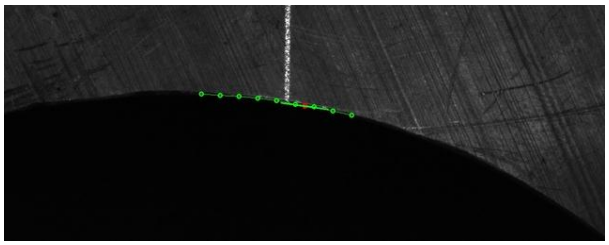
L'orientamento della stima del contorno subisce una leggera variazione nella frazione di profilo con raggio di curvatura più elevato (dal Frame 1.7 al Frame1.11), ma non vi è perdita di contatto.



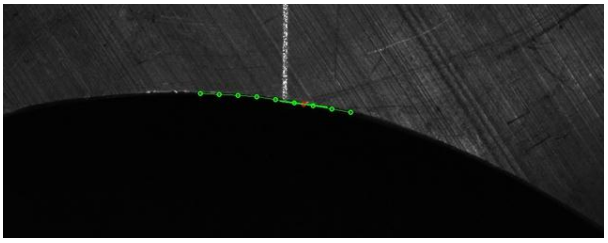
Frame 1.1



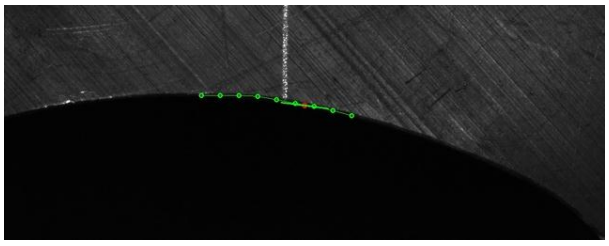
Frame 1.1



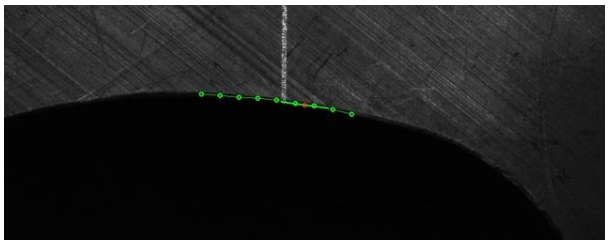
Frame 1.3



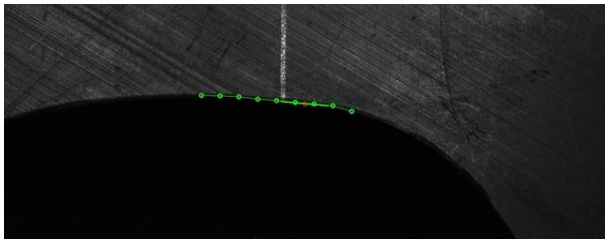
Frame 1.4



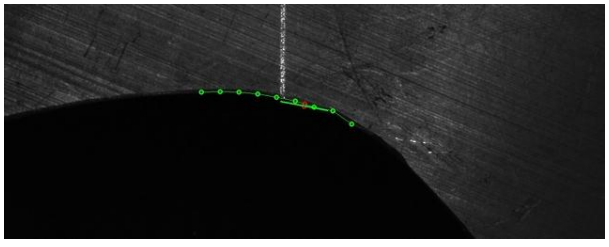
Frame 1.5



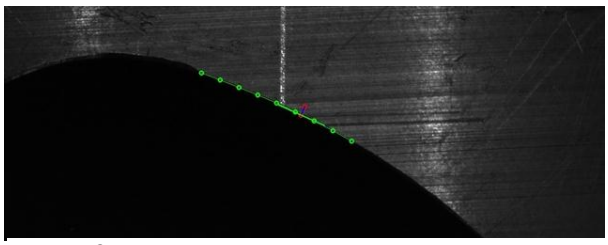
Frame 1.2



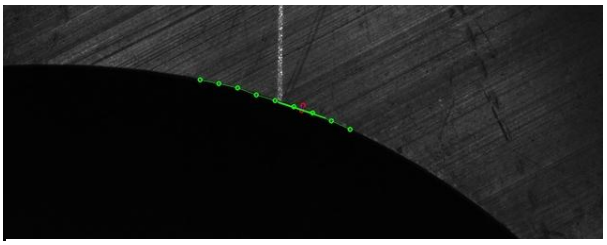
Frame 1.7



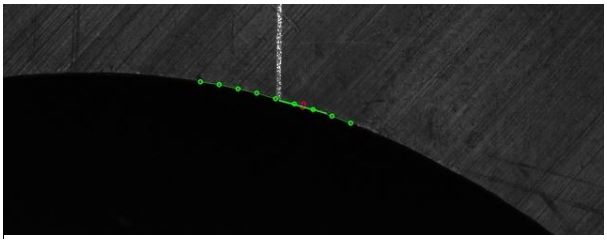
Frame 1.8



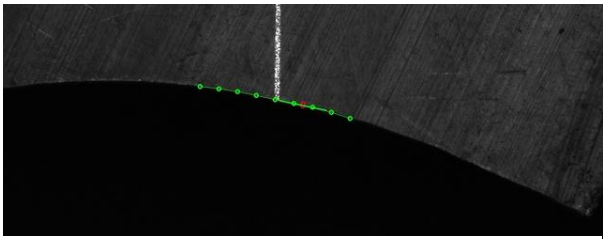
Frame 1.9



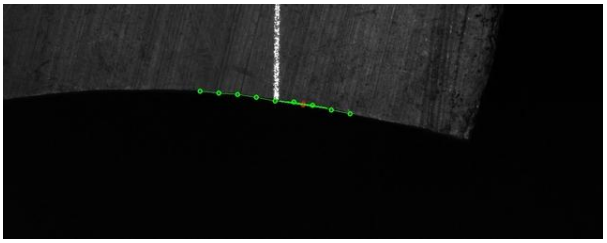
Frame 1.10



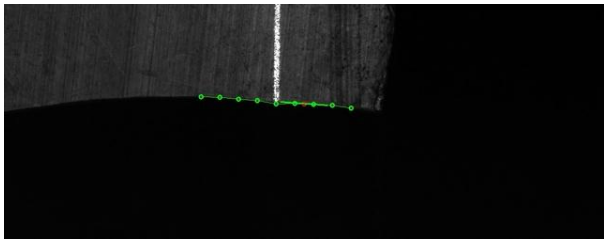
Frame 1.11



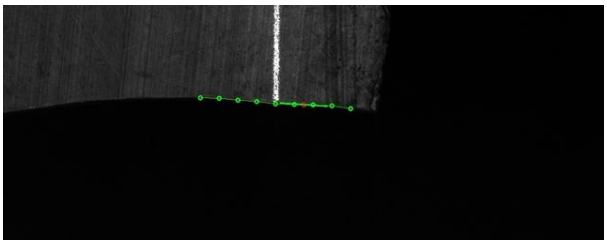
Frame 1.12



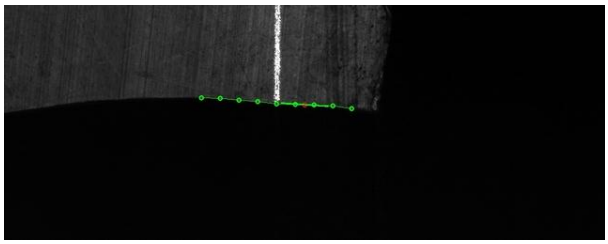
Frame 1.13



Frame 1.14



Frame 1.35



Frame 1.16

5.3 Contour following di un cerchione in lega di alluminio

Riportiamo infine i risultati di un test di contour following di un cerchione in lega di alluminio, oggetto di lavorazione (sbavatura) nel progetto Fidelio. In Figura 5.9 viene mostrata la vista dall'alto del cerchione appoggiato sul piano di lavoro in cui viene evidenziato il tratto inseguito durante la prova in esame.

Figura 5.9

L'unica modifica al setup sperimentale è il valore della coordinata z dell'organo terminale che è stato fissato a 0.975 metri.

Sono di seguito riportati gli andamenti degli errori in pixel tra coordinate desiderate e coordinate misurate ottenute dall'algoritmo di visione sviluppato in questa tesi. Possiamo notare come gli errori delle coordinate prese in considerazione siano di pochi pixel rispetto alle dimensioni dell'immagine 1280x1024 pixel. Da qui possiamo giudicare la bontà del controllo Visual Servoing.

Se si osservano i picchi degli andamenti degli errori delle coordinate di posizione (x_T, y_T) in Figura 5.10 e Figura 5.11 si può notare come questi presentino delle ampiezze contenute tranne negli ultimi istanti della prova. Al contrario l'errore di orientamento (Figura 5.12) presenta diversi picchi isolati sia negli istanti iniziali che negli finali.

Si può affermare che queste ampie variazione nelle misure sul piano immagine sono dovute a errori nell'estrapolazione dei punti del contorno da parte dell'algoritmo di edge detection, come visibile nei Frame 2.1, Frame 2.2 e Frame 2.16.

Questi errori sono dovuti alla presenza di ombre e riflessi causati dalla superficie bombata del cerchione nell'intorno del contorno da seguire.

La buona riuscita della prova non è comunque compromessa. Gli andamenti degli errori sono prossimi al valore nullo e nella sequenza di frame si può notare che non vi è perdita di contatto.

In Figura 5.13 si può osservare l'andamento delle coordinate x,y della terna camera nello spazio cartesiano ottenute tramite trasformazione prospettica delle coordinate nello spazio immagine. È possibile osservare che traiettoria percorsa corrisponda al profilo inseguito.

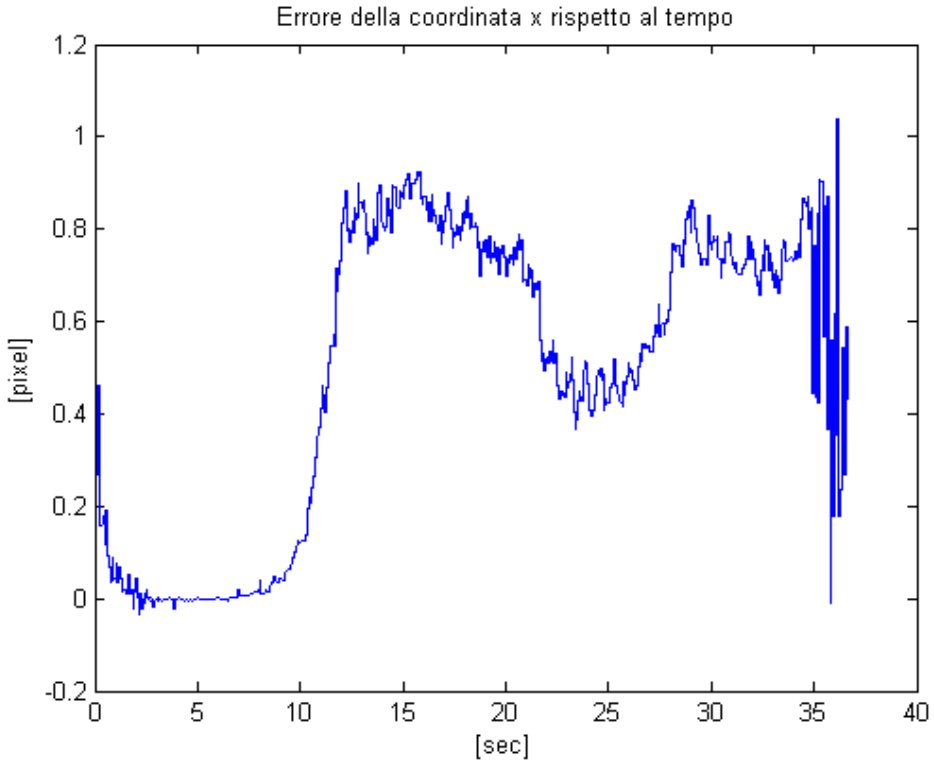


Figura 5.10

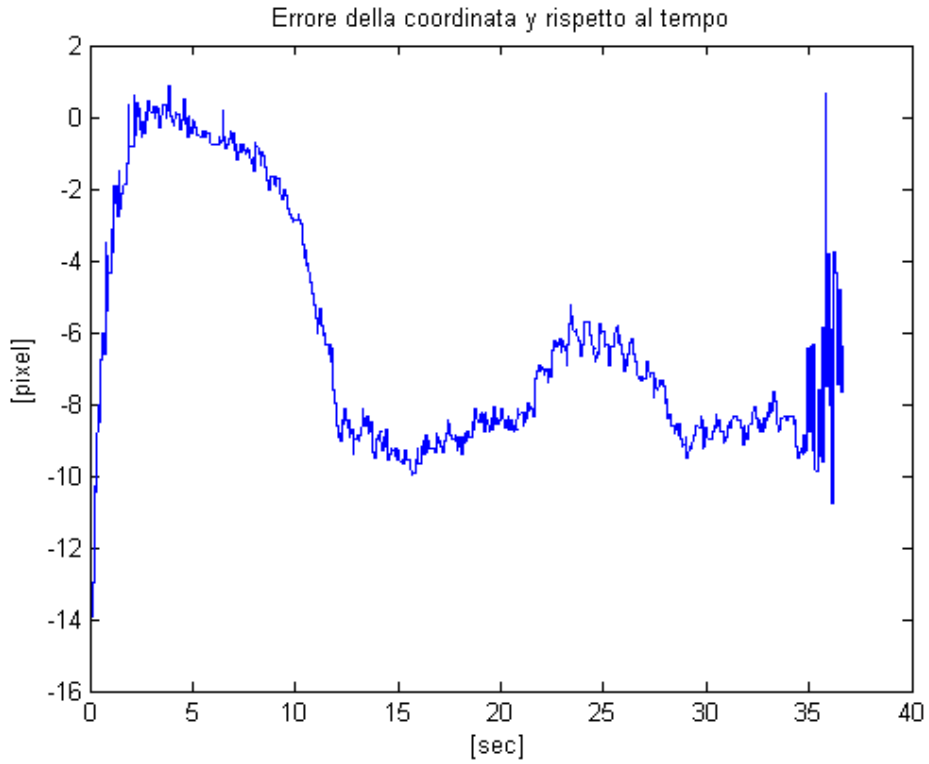


Figura 5.11

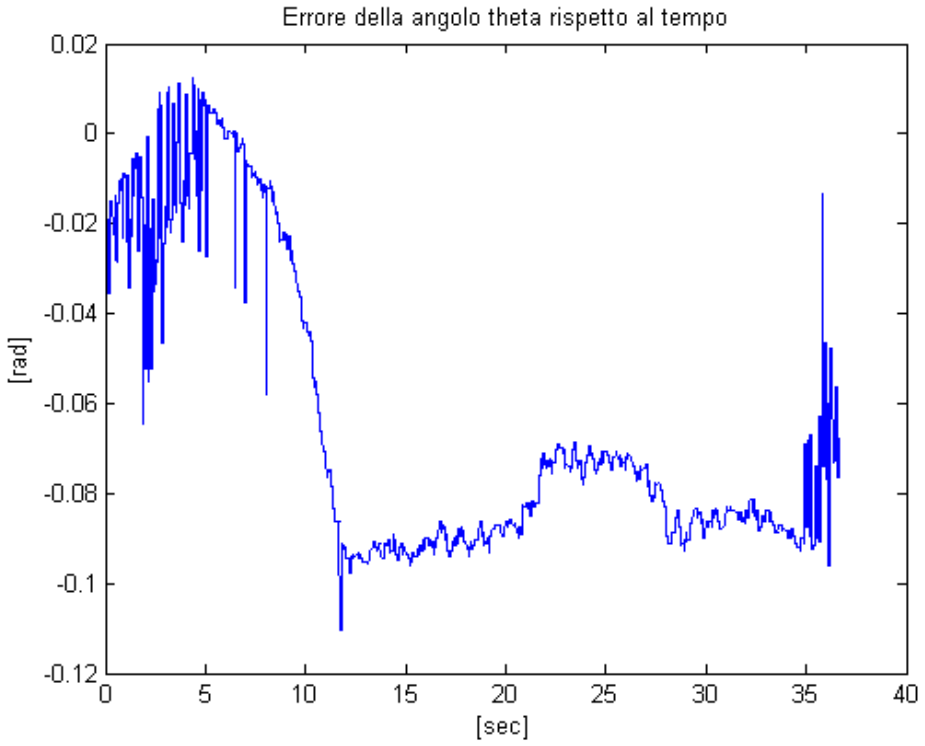


Figura 5.12

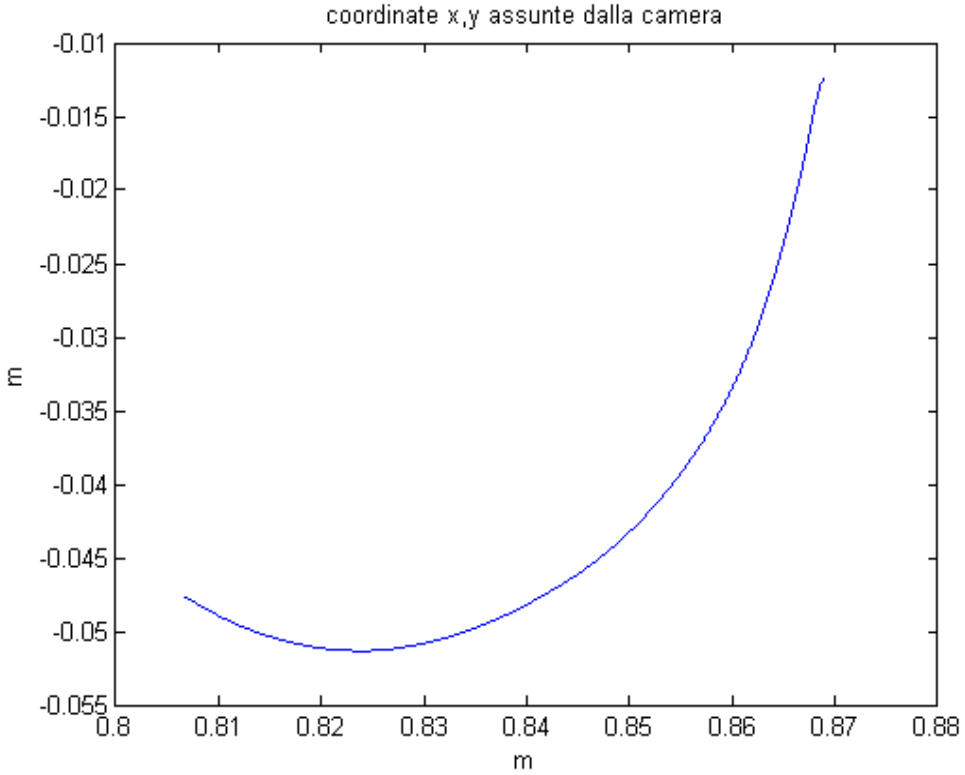


Figura 5.13

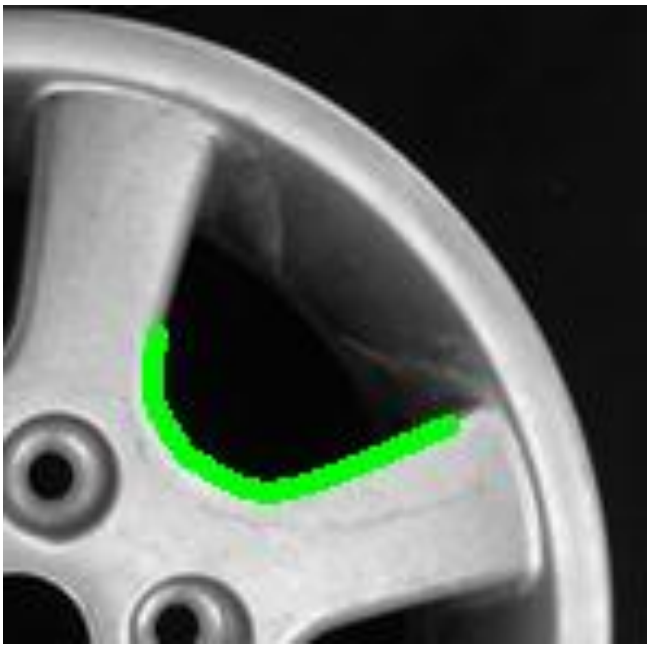
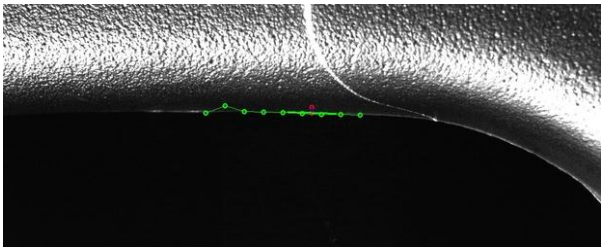
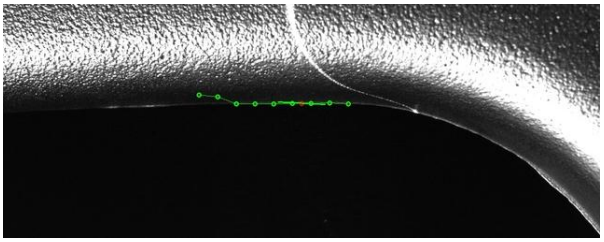


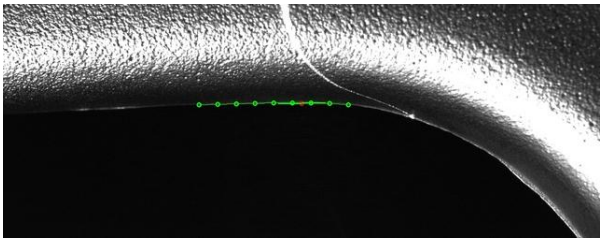
Figura 5.14



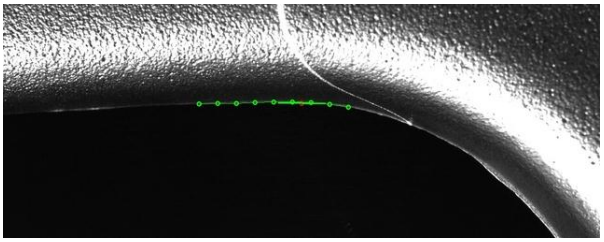
Frame 2.1



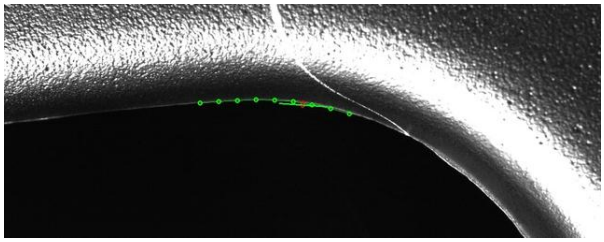
Frame 2.2



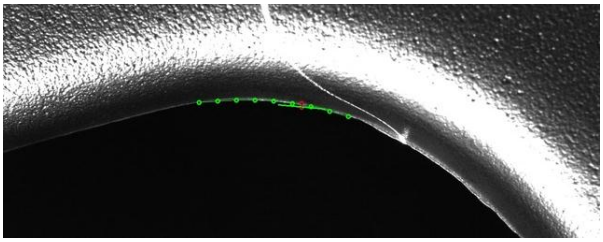
Frame 2.3



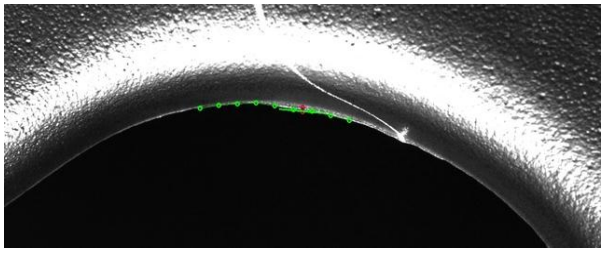
Frame 2.4



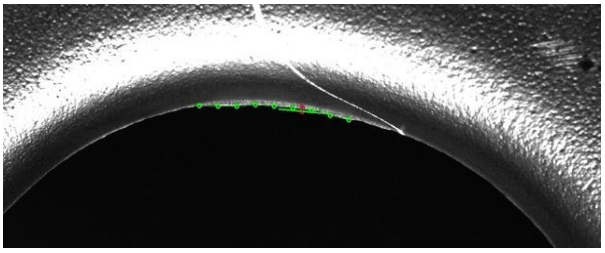
Frame 2.5



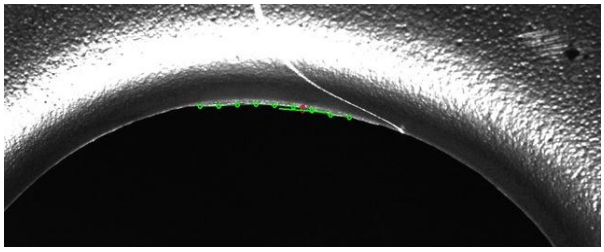
Frame 2.6



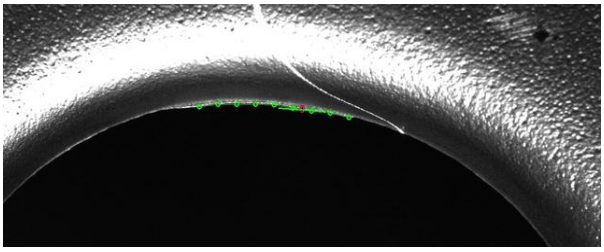
Frame 2.7



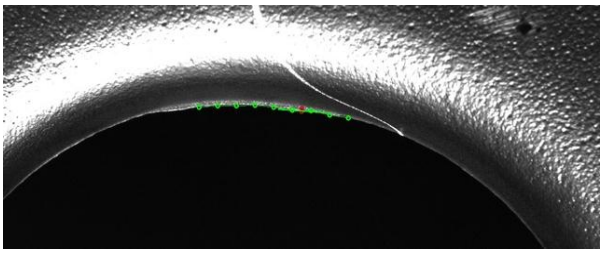
Frame 2.8



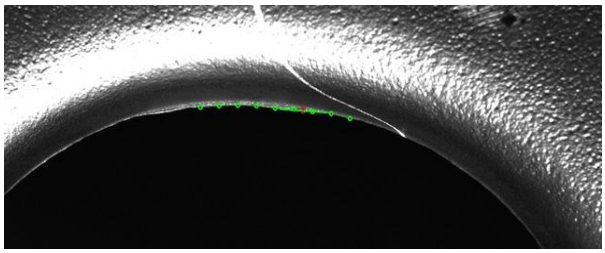
Frame 2.9



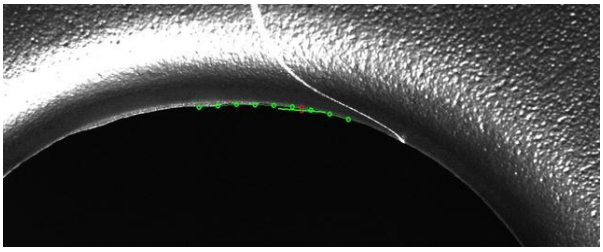
Frame 2.10



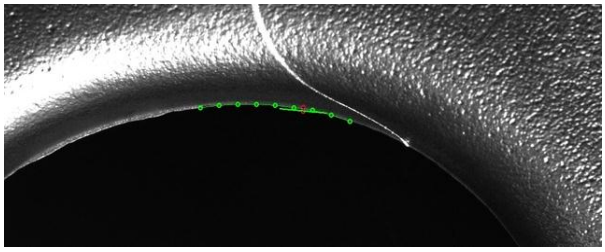
Frame 2.11



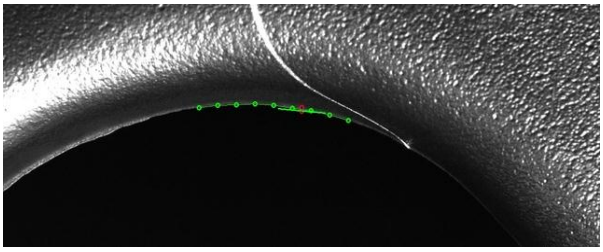
Frame 2.12



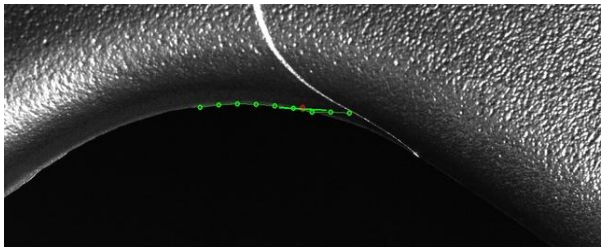
Frame 2.13



Frame 2.14



Frame 2.15



Frame 2.16

Capitolo 6

Conclusioni

In questa tesi è stato realizzato un algoritmo di visione artificiale inserito in un sistema di controllo Visual Servoing di manipolatori robotici finalizzato al conseguimento del contour following.

Il primo obiettivo raggiunto in questo progetto è stato la realizzazione di un'applicazione in ambiente Linux per l'acquisizione di frame video dalla telecamera montata su un robot in configurazione Eye-In-Hand.

L'algoritmo è stato completato con l'integrazione di funzionalità di visione artificiale messe a disposizione dalla libreria OpenCV, che hanno consentito di stimare la posa del profilo metallico inseguito, minimizzando il carico computazionale in maniera significativa, consentendo di sfruttare a pieno le caratteristiche tecniche della telecamera in dotazione e di fornire in tempi minimi le misure di posizione al sistema di controllo.

I risultati sperimentali riportati nel capitolo precedente e ottenuti attraverso prove di contour following dimostrano le potenzialità dell'applicazione realizzata in termini di carico computazionale, ma hanno evidenziato scarsa robustezza ai disturbi (ombre e riflessi) forniti dalle sorgenti luminose nel caso di prove con profili che presentano superfici arrotondate (es. cerchione in lega di Alluminio).

Sviluppi futuri prevedono l'introduzione di una sorgente luminosa posizionata sopra il banco di prova e progettata ad hoc per eliminare le variazioni luminose e i disturbi dovuti all'illuminazione esterna al sistema (ambientale e artificiale).

Bibliografia

[1] R.C.Gonzalez and R.E.Woods: *Digital Image Processing, 3rd Ed.*, Prentice Hall, 2008.

[2] A.Bovik, *The essential guide to Image Processing*, Academic Press, 2009.

[3] Jianjun Wang, Hui Zhang, Zhang, «A force control assisted robot path generation system,» in *Automation Science and Engineering, 2008. CASE 2008. IEEE International Conference*, 23-26 Aug. 2008.

[4] J. Pomares, F. Torres, « *Movement-flow-based visual servoing and force control fusion for Manipulation Tasks in unstructured environments* » *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS,* 2005.

[5] H. Koch, A. Konig, A. Weigl-Seitz, K. Kleinmann, J. Suchy, « *Force, acceleration and vision sensor fusion for contour following tasks with an industrial robot* » *Robotic and Sensors Environments (ROSE),* 2011 IEEE International Symposium on 17-18 Sept. 2011

[6] J. De Schutter, J. Baeten, «*Hybrid vision/force control at corners in planar robotic-contour following*», Mechatronics, IEEE/ASME Transactions on Jun 2002.

[7] A.C. Leite, F. Lizarralde, Liu Hsu «*Hybrid vision-force robot control for tasks on unknown smooth surfaces*» in *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, May 2006.

[8] Chien Chern Cheah, S.P. Hou, Yu Zhao, J.-J.E. Slotine, «*Adaptive Vision and Force Tracking Control for Robots With Constraint Uncertainty*» *IEEE/ASME TRANSACTIONS ON MECHATRONICS,* Jun 2010.

[9] Sang-Wook Jeon, Doo-Sung Ahn, Hyo-Jeong Bae, Chang-Woo Hong, «*Object Contour Following Task based on Integrated Information of Vision and Force sensor*» in *International Conference on Control, Automation and Systems*, Seou,

Korea, 2007.

[10 Imaging Development Systems GmbH, «http://www.ids-imaging.com» [Online].

[11] IDS, «uEye Camera Manual,» [Online]. Available: http://www.ids-imaging.de/frontend/files/uEyeManuals/Manual_eng/uEye_Manual/index.html.

[12] MVTec Software GmbH, «HALCON the power of Machine Vision,» [Online] Available: http://www.mvtec.com/halcon/.

[13] COMAU Robotics, «Manuale di istruzioni SMART SiX - Specifiche Tecniche».

[14] B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo, Robotica - Modellistica, pianificazione e controllo, McGraw-Hill.

[15] A. Berson, Client/Server Architecture, McGraw-Hill, 1996.

[16] G. Bradsky e A.Kaehler, Learning OpenCV – Computer vision with the OpenCV Library, O'REILLY.

[17] W. Garage, «http://opencv.willowgarage.com,» [Online].

[18] «OpenCV documentation,» [Online]. Available: http://docs.opencv.org .

[19] «GSL - GNU Scientific Library,» [Online].

Available: http://www.gnu.org/software/gsl/.

[20] «tpl library» [Online]. Available: http://tpl.sourceforge.net/.

[21] «The GNU C Library,» [Online].

Available: http://www.gnu.org/software/libc/manual.