Alma Mater Studiorum · Università di Bologna
Campus di Cesena
Scuola di Ingegneria e Architettura

Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Studio e implementazione di tecniche di Hand Eye Calibration di un Braccio Robotico

Tesi di Laurea in
Tecniche Avanzate Per L'analisi Delle Immagini E Visione

Relatore:
Prof.
Alessandro Bevilacqua

Correlatori:
Ing. Luca Tersi
Ing. Alessandro Gherardi
Ing. Giuseppe Casadio

Presentata da:
Andrea Ridolfi

Sessione III
Anno Accademico 2014/2015

Indice

0.1 Abstract

La tesi indaga il problema di calibrazione Hand Eye, ovvero la trasformazione geometrica fra i diversi sistemi di riferimento della camera e dell'attuatore del robot, presentando il problema ed analizzando le varie soluzioni proposte in letteratura. Viene infine presentata una implementazione realizzata in collaborazione con l'azienda SpecialVideo, implementata utilizzando l'algoritmo proposto da Konstantinos Daniilidis, il quale propone una formulazione del problema sfruttando l'utilizzo di quaternioni duali, risolvendo simultaneamente la parte rotatoria e traslatoria della trasformazione. Si conclude il lavoro con una analisi dell'efficacia del metodo implementato su dati simulati e proponendo eventuali estensioni, allo scopo di poter riutilizzare in futuro il lavoro svolto nel software aziendale, con dati reali e con diversi tipi di telecamere, principalmente camere lineari o laser.

Capitolo 1

Introduzione

Nel momento in cui si applicano tecniche di visione artificiale alla robotica, montando una telecamera a bordo di un braccio robotico, nasce intrinsecamente la necessità di conoscere la relazione fra la telecamera utilizzata e il robot sulla quale la camera è fisicamente montata. Tale relazione consiste nella posizione e orientazione, più brevemente denominata posa, del centro ottico della telecamera, o occhio, relativamente al robot su cui è rigidamente fissata, la mano, da cui nasce il nome con cui è comunemente conosciuto il problema: Hand Eye Calibration.

Conoscere la posa della camera rispetto al robot è di fondamentale importanza per l'utilizzo del robot stesso in almeno due situazioni:

- Muovere il robot con precisione all'interno del piano di lavoro con misure prelevate dal sensore. Si pensi ad esempio al caso in cui si vogliano manipolare con precisione oggetti sul piano di lavoro. Per farlo occorre esprimere una posizione vista dalla telecamera, e quindi rispetto al suo sistema di riferimento, in coordinate comprensibili al robot e quindi rispetto al suo sistema di riferimento; necessariamente occorre essere in possesso della posa in maniera più precisa possibile fra telecamera e robot stesso, in modo da poter esprimere posizioni del mondo reale viste dalla camera nel sistema di riferimento del robot.

- Muovere il sensore con precisione tramite il robot. Si pensi ad una situazione diametralmente opposta, in cui cioè si voglia muovere il

sensore ottico, ad esempio per la scansione di un oggetto tramite camera laser, sfruttando la mobilità del robot al quale il sensore è fissato. Per poter effettuare movimenti di precisione occorre conoscere la stessa trasformazione del caso precedente con la maggiore accuratezza possibile.

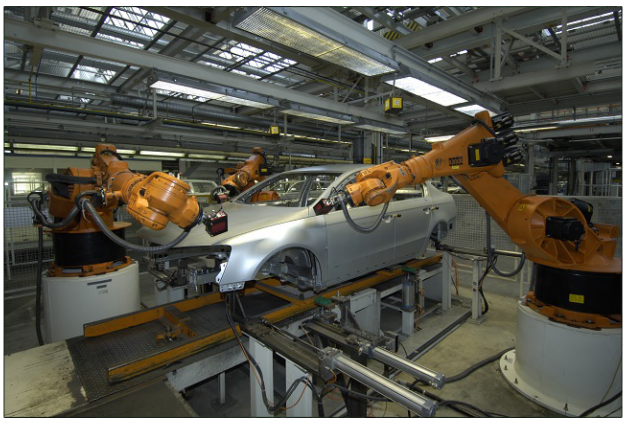
Per calcolare la calibrazione richiesta occorre necessariamente utilizzare tecniche di visione artificiale, in quanto spesso la camera è parte integrante della mano del robot, rendendo impossibile una misurazione diretta della posa. Quanto più il processo di calibrazione è accurato, tanto più sarà precisa la posa stimata e di conseguenza sarà più preciso il robot nei movimenti, dando la possibilità di effettuare operazioni più complicate e precise.

Negli anni sono state proposte diverse soluzioni, descritte nel capitolo 2, nel quale viene illustrato formalmente il problema di calibrazione e viene fornita una breve spiegazione teorica sul funzionamento delle soluzioni proposte in letteratura, oltre ad essere presentata la soluzione correntemente adottata da SpecialVideo. Viene poi proposta, nel capitolo 3, una soluzione basata sulla teoria analizzata in precedenza, descrivendone il processo di design e le librerie di cui si è trovato necessario l'utilizzo, con l'obiettivo di ottener un risultato il più possibile modulare, portabile e mantenibile, rendendo il codice realizzato il più facilmente possibile integrabile all'interno dell'applicativo aziendale. Nel quarto capitolo viene descritta, più nel dettaglio, l'implementazione, presentando le interfacce dell'applicativo e l'utilizzo dello stesso.

Nel capitolo 5 vengono presentati i risultati dei test effettuati sull'algoritmo implementato soggetto a diversi tipo di utilizzo e di rumore nei dati acquisiti, aggiungendo alcune considerazioni sui risultati ottenuti.

Vengono infine tratte le conclusioni sul lavoro svolto, il quale prevederà un'estensione con l'obiettivo di utilizzare quanto già creato con altre tipologie di telecamere, ad esempio camere lineari o laser, utilizzando diversi tool di calibrazione, diversi dalla classica scacchiera, dando la possibilità di rendere la calibrazione più generica e precisa.

(a)



(b)

Figura 1.1: Un esempio di funzionamento di robot in ambito industriale. Nel caso di una catena di montaggio automobilistica (a), ad esempio, è di fondamentale importanza utilizzare robot con sistemi di visione ben calibrati, per garantire il perfetto assemblaggio dei componenti. Aggiungere un sistema di visione al robot permette di ottenere un perfetto assemblaggio adattandosi, in real time, ad eventuali spostamenti od imperfezioni, percepiti dalla camera, che non sarebbero altrimenti possibili utilizzando lo stesso movimento meccanicamente identico ripetuto per ogni vettura. Un caso opposto, in cui si ha la necessità di muovere un robot con precisione per spostare, conseguentemente, la camera montata a bordo di esso, si può avere ad esempio, sempre nel caso automobilistico, nel caso di controllo qualità (b) dei difetti di una automobile.

Capitolo 2

Analisi del problema e del contesto

Viene, in questo capitolo, presentato inizialmente il problema in questione, dopodiché viene presentato un breve riassunto sull'evoluzione del problema di Hand Eye calibration nel tempo, presentando in ordine cronologico alcune delle varie pubblicazioni che hanno proposto diversi approcci o formulazioni risolutive. Verrà analizzato il percorso effettuato, dalla prima formulazione di Tsai e Lenz (1989) fino alle più moderne formulazioni come quella proposta in seguito da Heller, Havlena e Tomas (2015), mettendo in luce vantaggi e svantaggi nell'utilizzo dell'una o dell'altra soluzione, creando un percorso il più possibile lineare e semplice da seguire per il lettore. Verranno inoltre introdotti concetti matematici quali quaternioni[A] e numeri duali[B], approfonditi in appendice. Infine vengono presentate le precedenti soluzioni utilizzate da SpecialVideo per risolvere il problema.

2.1 Hand Eye Calibration di un braccio robotico

In visione artificiale ed in robotica un concetto fondamentale risulta essere quello della posa di un oggetto ovvero la combinazione della tra-

slazione e rotazione di un oggetto relativa ad un determinato sistema di riferimento.

Nel caso in cui si abbia una camera montata a bordo di un robot si possono riconoscere quattro pose fondamentali:

- Posa Flangia-Base del robot. Questa è la posa fondamentale di un robot in quanto rappresenta la sua estremità rispetto alla base fissa sulla quale si muove. Genericamente tale posa viene fornita tramite il software di controllo del robot stesso, fornito dall'azienda creatrice del robot.

- Posa Scacchiera-Camera. Nel caso di una camera è possibile stimare la posizione di un tool di calibrazione, ottenendo la posa della scacchiera rispetto al centro ottico della telecamera, o viceversa. Tale stima avviene tramite tecniche di calibrazione standard facilmente implementabili e tramite il riconoscimento di un determinato oggetto di calibrazione, solitamente una scacchiera, geometricamente noto a priori.

- Posa Camera-Flangia. Ottenere la posa in questione equivale a risolvere il problema di Hand Eye calibration, in quanto conoscere la posa della camera relativa alla flangia ci da la possibilità di esprimere agevolmente la quarta e ultima posa del sistema robot-telecamera.

- Posa Scacchiera-Base del robot. Questa posa è una composizione delle tre trasformazioni precedenti. Conoscere le tre trasformazioni precedenti permette al sistema di esprimere in coordinate relative al sistema di riferimento del robot pose relative alla camera.

Viene mostrato in figura 2.1 un tipico caso d'esempio in cui si richiede il calcolo della calibrazione Hand Eye. Il robot in figura possiede sei gradi di libertà, potendosi muovere liberamente, con la camera fissata all'ultimo punto mobile del robot, chiamato flangia, alla quale vengono poi rigidamente fissati i diversi tool a seconda del compito specifico del robot.

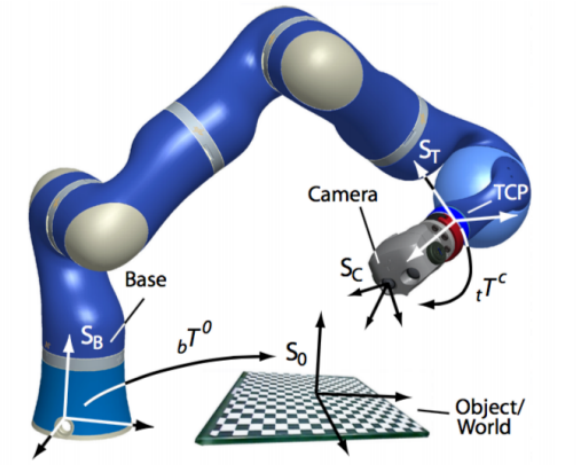


Figura 2.1: Il disegno raffigura un caso ideale di Hand Eye calibration, in cui un robot con camera rigidamente fissata ha la possibilità di acquisire immagini di un oggetto di calibrazione noto, in questo caso una scacchiera. *Courtesy of Horaud and Dornaika.*

Avendo a disposizione un oggetto di calibrazione, comunemente una scacchiera, del quale sono conosciute a priori le caratteristiche geometriche, è possibile muovere il robot, e con esso la camera, allo scopo di ottenere diverse immagini, in diverse posizioni, ritraendo l'oggetto di calibrazione.

Da queste immagini è possibile prelevare, analizzando la distorsione della scacchiera causata dalla trasformazione prospettica, le posa della camera rispetto all'oggetto di calibrazione. Questa fase viene chiamata estrazione dei parametri estrinseci, comunemente presentati sotto forma di una matrice di rototraslazione 4×4

$$\begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}$$

dove R rappresenta una matrice di rotazione 3×3 e t un vettore di traslazione tridimensionale.

La posizione del robot rispetto al proprio sistema di riferimento viene invece prelevata direttamente dal robot stesso, in maniera differente

a seconda del modello del robot, affidandosi alle specifiche fornite dal costruttore ed al suo ambiente software.

Questo insieme di informazioni permettono di ottenere la formulazione classica del problema di Hand Eye Calibration nella forma di un sistema di equazioni omogenee nella forma:

$$AX = XB$$

Nella formulazione classica, ponendo due posizioni 1 e 2 del sistema camera-robot, si rappresenta con A la matrice di rototraslazione frutto della trasformazione fra le due pose del sensore calcolata con:

$$A = A_2 A_1^{-1}$$

ed in maniera analoga con B la trasformazione fra le pose della flangia calcolata con:

$$B = B_2^{-1} B_1$$

X infine rappresenta la soluzione al problema, ovvero la trasformazione fra centro ottico del sensore e flangia.

2.2 Problema e approcci per la risoluzione

Vengono di seguito presentati varie pubblicazioni, ordinate cronologicamente, con diverse soluzioni per il problema. Il primo approccio risolutivo fu presentato da Tsai e Lenz (1989)[25][26] i quali propongono una soluzione al problema nella sua formulazione $AX = XB$ presentata nella sezione precedente, il quale prevede di risolvere il sistema di equzioni decomponendo il problema in parte rotatoria e traslatoria, risolvendole in sequenza. Successivamente Horaud e Dornaika (1995)[20] propongono una diversa formulazione del problema equivalente a quella precedente, introducendo inoltre i quaternioni sfruttandone le proprietà per proporre due nuovi approcci risolutivi utilizzabili in entrambe le formulazioni, uno, in maniera simile a quanto proposto in precedenza, risolvendo separatamente parte rotatoria e traslatoria, l'altro, più performante, risolvendo contemporaneamente per rotazione e traslazione. La soluzione successiva è quella proposta da Daniilidis (1999)[13], che sfrutta i quaternioni

duali risolvendo il problema algebricamente tramite scomposizione per valori singolari, stimando contemporaneamente rotazione e traslazione senza ricorrere a tecniche di minimizzazione non lineari. Strobl e Hirzinger (2006)[24] risolvono il problema studiando la metrica delle rotazioni e traslazioni, ottenendo una funzione errore alternativa per risolvere il problema tramite minimizzazione non lineare. Infine si presenta una recente formulazione del problema proposta da Heller *et al.*(2015)[18][19][17] la quale pone una formulazione completamente nuova rispetto alle precedenti, utilizzando la tecnica del Branch and Bound per risolvere il problema, minimizzando una funzione errore ricavata dai vincoli epipolari derivati dalle immagini.

2.2.1 Tsai and Lenz (1989), using 3D Machine vision

I primi a studiare il problema ed a proporre la formulazione $AX = XB$ presentata in precedenza, utilizzando tecniche avanzate di visione artificiale, sono, nel 1989, Tsai e Lenz[25][26]. Fino ad allora lo stato dell'arte prevedeva l'utilizzo dei parametri del modello cinematico del robot insieme ai parametri Hand/Eye, dovendo risolvere un sistema non lineare con oltre 30 incognite. Tale problema aveva gli svantaggi di avere un alto costo computazionale, oltre alla necessità di avere una buona soluzione di partenza per garantire la convergenza verso la soluzione ottimale.

La soluzione da loro proposta prevede di separare la parte di calibrazione del modello del robot con la parte di Hand Eye calibration ed evitare un problema di ottimizzazione non lineare globale con un così alto numero di incognite. Per raggiungere lo scopo si utilizza un setup del problema classico come visto in figura 2.1, il quale consiste in una comune scacchiera simile a quelle utilizzate per la calibrazione oltre al robot ed alla camera, con quest'ultima fissata rigidamente alla flangia.

L'algoritmo risolutivo si basa sulla rappresentazione di una trasformazione come insieme della rotazione di angolo θ attorno ad un asse passante per l'origine con direzione dei coseni $[n_1, n_2, n_3]$ seguita da una traslazione T, così rappresentata:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R \begin{bmatrix} x \\ y \\ z \end{bmatrix} + T$$

o in maniera equivalente:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

dove (x, y, z) e (x', y', z') sono le coordinate di un punto qualsiasi rispettivamente prima e dopo aver subito la trasformazione.

R è una matrice ortonormale $3x3$ di primo tipo ($det(R) = 1$)

$$R = \begin{bmatrix} n_1^2 + (1 - n_1^2)\cos\theta & n_1 n_2(1 - \cos\theta) - n_3 \sin\theta & n_1 n_3(1 - \cos\theta) + n_2 \sin\theta \\ n_1 n_2(1 - \cos\theta) + n_3 \sin\theta & n_2^2 + (1 - n_2^2)\cos\theta & n_2 n_3(1 - \cos\theta) - n_1 \sin\theta \\ n_1 n_3(1 - \cos\theta) - n_2 \sin\theta & n_2 n_3(1 - \cos\theta) + n_1 \sin\theta & n_3^2 + (1 - n_3^2)\cos\theta \end{bmatrix}$$

Avendo R un autovalore unitario, ed essendo il vettore $P_R \equiv [n_1 n_2 n_3]^T$ l'asse di rotazione è possibile esprimere la rotazione R tramite il suo autovettore, il quale è composto dal vettore P_R scalato di una funzione di θ, usando una versione modificata della conosciuta formula di Rodrigues.

$$P_r = 2 \sin\frac{\theta}{2} [\ n_1 \quad n_2 \quad n_3\ ]^T \quad con \quad 0 \leq \theta \leq \pi$$

É quindi possibile esprimere la rotazione R come funzione di P_r

$$R = \left(1 - \frac{|P_r|^2}{2}\right) + \frac{1}{2}\left(P_r P_r^T + \alpha \cdot Skew(P_r)\right)$$

con $\alpha = \sqrt{4 - |P_r|^2}$ e $Skew(P_r)$ matrice traccia così definita:

$$Skew(V) = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix}$$

É possibile sfruttare questa rappresentazione della matrice di rotazione, insieme ad undici lemmi[1] di natura geometrica ed algebrica esposti nel dettaglio nel paper, per arrivare all'algoritmo seguente, in grado di risolvere agevolmente il problema:

1. Per ogni coppia di stazioni i, j, corrispondenti ad uno spostamento del robot dalla posizione i alla posizione j, comporre il sistema di equazioni con P'_{cg} come incognita

   $$Skew(P_{gij} + P_{cij}) \cdot P'_{cg} = P_{cij} - P_{gij}$$

   Dove, P_{cij} rappresenta la trasformazione fra le due stazioni della camera, P_{gij} la trasformazione fra le due stazioni della flangia, come mostrato in figura 2.2. Essendo $Skew(P_{gij} + P_{cij})$ sempre singolare, sono sufficienti almeno due stazioni per risolvere univocamente il sistema per P'_{cg} risolvendo per minimi quadrati.

2. Calcolare $\theta_{R_{cg}} = 2\tan^{-1}|P'_{cg}|$.

3. Calcolare P_{cg}

   $$P_{cg} = \frac{2P'_{cg}}{\sqrt{\left(1 + P'_{cg} \cdot P'^{T}_{cg}\right)}}$$

   Dove P_{cg} rappresenta la parte rotatoria della trasformazione camera flangia.

4. Infine calcolare la componente traslatoria T_{cg} costruendo il sistema

   $$(R_{gij} - 1)T_{cg} = R_{cg}T_{cij} - T_{gij}$$

   risolvendo per T_{cg} per minimi quadrati.

Seppure computazionalmente molto rapido, Tsai e Lenz individuano nella fase di test alcuni fattori di critica importanza per migliorare l'accuratezza dell'algoritmo, fornendo alcune linee guida da seguire al setup del sistema per migliorare la precisione del risultato.

[1] Non riportati per brevità all'interno della tesi, consultabili nel paper *Real Time Versatile Robotics Hand/Eye Calibration Using 3D Machine Vision*[25][26] di Tsai e Lenz.

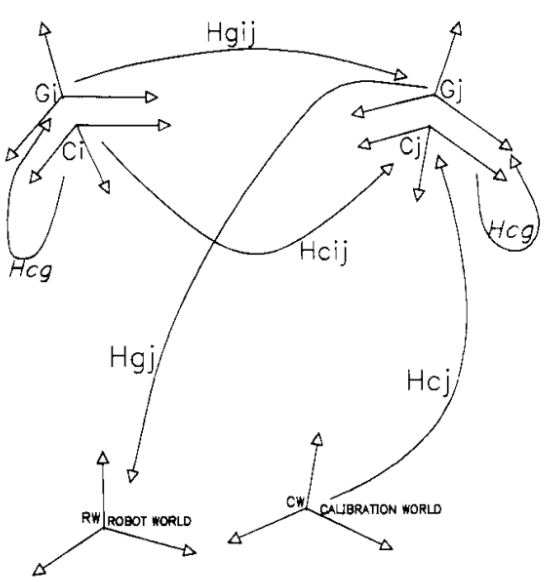


Figura 2.2: Il disegno raffigura il setup per l'algoritmo di Tsai e Lenz, con i nomi delle trasformazioni nei diversi sistemi di riferimento.

In primo luogo occorre porre attenzione alla scelta delle stazioni, le quali devono avere la massima rotazione possibile fra le une e le altre (il più possibile ortogonali). Tale necessità è dovuta al fatto che la rotazione fra camera e gripper viene calcolata a partire dalla differenza fra le rotazioni nelle diverse stazioni.

Un secondo fattore critico è la rotazione stessa nelle stazioni, un valore maggiore porta un errore minore nella stima della rotazione della posa camera-flangia.

Un altro fattore critico risiede nella distanza fra camera e blocco di calibrazione, diminuendo la quale si diminuisce l'errore di traslazione. Maggiore è la distanza fra blocco e camera, maggiormente l'effetto di una rotazione rispetto al blocco di calibrazione influisce sulla stima della distanza, influenzando di conseguenza la stima fra camera e flangia.

Per motivi analoghi un altro fattore cruciale risiede nella trasformazione fra diverse stazioni della flangia. Seppure non sia importante la posizione assoluta rispetto al centro del robot è importante che la trasformazione fra una stazione e la seguente sia il più precisa possibile. Tsai

e Lenz hanno notato, nei loro test, che l'errore fra stazioni successive nella rotazione influisce in maniera molto maggiore rispetto ad un errore nella traslazione. Questo nasce come conseguenza del fatto che la soluzione proposta da Tsai e Lenz prevede di disaccoppiare la parte rotatoria da quella traslatoria, calcolando quest'ultima a partite dalla stima della prima. Di conseguenza errori nel calcolo delle rotazioni si propagano nel calcolo delle traslazioni, aumentando ulteriormente l'errore finale. A tal proposito soluzioni successive risolvono il problema stimando la trasformazione rotatoria e traslatoria contemporaneamente, ottenendo risultati più precisi.

Per ottenere un valore preciso viene quindi consigliato di utilizzare stazioni molto differenti fra loro, soprattutto riguardo alle rotazioni, cercando di mantenere il più possibile la camera vicino al blocco di calibrazione. Inoltre fondamentale importanza risiede in una corretta calibrazione della camera e del robot, da effettuarsi con metodi tradizionali in maniera più accurata possibile, in quanto la calibrazione Hand Eye viene determinata a partire da queste basi.

É importante sottolineare tali osservazioni effettuate da Tsai e Lenz in quanto, seppure le tecniche di calibrazione si siano evolute nel tempo, adottando metodologie ed algoritmi più robusti, performanti ed efficaci, tali tecniche, accortezze ed osservazioni rimangono tuttavia ottimi principi da seguire in qualsiasi ambito riguardante la calibrazione di un robot o di una camera, portando a risultati migliori e più precisi.

2.2.2 Horaud and Dornaika (1995), using unit quaternions

Nel 1995 Radu Horaud e Fadi Dornaika propongono una nuova soluzione, diversa dalla classica risoluzione della matrice di equazioni omogenee nella forma di

$$AX = XB$$

in una forma differente:

$$MY = M'YB$$

Il vantaggio di questa alternativa formulazione del problema di Hand Eye Calibration consiste nel rimuovere dal problema la necessità dell'ottenere esplicitamente i parametri intrinseci ed estrinseci della telecamera. La formulazione proposta prevedere infatti l'utilizzo diretto della matrice di trasformazione prospettica 3×4 M e M' associate a due posizioni della camera rispetto all'oggetto di calibrazione.

La formulazione proposta è tuttavia matematicamente equivalente alla formulazione classica, come verrà dimostrato in seguito.

Nella formulazione classica, la trasformazione della camera fra le due pose viene esplicitata tramite la decomposizione della matrice prospettica 3×4 M_i, ottenuta tramite la calibrazione, in intrinseci ed estrinseci:

$$M_i = CA_i = \begin{pmatrix} \alpha_u & 0 & u_0 & 0 \\ 0 & \alpha_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} R_{Ai} & t_{Ai} \\ 0 & 1 \end{pmatrix} \tag{2.1}$$

La prima matrice rappresenta i parametri intrinseci della telecamera[2], con α_u e α_v lunghezza focale e u_0 e v_0 centro dell'immagine, mentre la seconda matrice rappresenta i parametri estrinseci, ovvero la posa della camera rispetto all'oggetto di calibrazione.

La nuova formulazione proposta evita questa decomposizione. Definiamo Y come la matrice di trasformazione fra l'oggetto di calibrazione e la mano del robot, o flangia, mentre X rappresenta come in precedenza la trasformazione camera flangia.

Risulta immediato ottenere:

$$X = A_1 Y$$

dove A_1 è la matrice di trasformazione fra la camera e l'oggetto di calibrazione.

Sostituendo X ottenuta da quest'ultima equazione e $A = A_2 A_1^{-1}$ alla formulazione classica $AX = XB$ si ottiene:

$$A_2 Y = A_1 Y B$$

[2]Si suppone in questa rappresentazione di descrivere la camera tramite il modello pin-hole, nella realtà sono necessari parametri aggiuntivi per descrivere la distorsione radiale e prospettica dovuta all'utilizzo di lenti ottiche.

che a sua volta, premoltiplicando per C ed utilizzando eq. (2.1) con $i = 1, 2$ si ottiene:

$$M_2 Y = M_1 Y B \tag{2.2}$$

equivalente alla formulazione classica originale.

Tuttavia in questa equazione l'incognita Y è la trasformazione fra la flangia del robot a cui la camera è fissata e l'oggetto di calibrazione invece che fra flangia e camera. Tale scelta porta ad una semplificazione del problema, in quanto non viene più assunto l'utilizzo di una camera pin-hole perfetta, evitando il problema di decomposizione della matrice di trasformazione in intrinseci ed estrinseci.

Equivalenza della nuova formulazione rispetto alla originale

Non solo la formulazione proposta da Horaud e Dornaika è equivalente alla formulazione originalmente proposta da Tsai e Lenz, ma viene inoltre risolta con un approccio utilizzabile anche nella formulazione originale.

Tsai e Lenz propongono come visto in precedenza di risolvere l'equazione $AX = XB$ scomponendo l'equazione in un sistema di equazioni, una matriciale dipendente solamente da rotazioni ed una vettoriale dipendente sia da rotazioni che da traslazioni:

$$R_A R_X = R_X R_B \tag{2.3}$$

$$(R_A - I)t_X = R_X t_B - t_A \tag{2.4}$$

Per risolvere l'eq. (2.3) si sfruttano le proprietà algebriche e geometriche delle matrici di rotazione. Riscrivendo l'equazione come

$$R_A = R_X R_B R_X^T$$

R_A e R_B hanno lo stesso autovalore, essendo matrici simili; hanno inoltre un autovalore unitario essendo matrici di rotazione.

Chiamiamo gli autovettori assiociato all'autovalore unitario n_A e n_B, possiamo a questo punto ottenere

$$R_A R_X n_B = R_X R_B n_B = R_X n_b$$

postmoltiplicando per n_b. Si trova quindi che l'autovettore n_A, associato all'autovalore unitario, è uguale a

$$n_A = R_X n_B \tag{2.5}$$

Per concludere, la risoluzione della formulazione classica del problema equivale al risolvere l'eq. (2.5) e l'eq. (2.4).

Viene ora mostrato come la formulazione proposta da Horaud e Dornaika sia matematicamente equivalente alla formulazione classica, fornendo perciò un approccio in grado di risolvere entrambe.

Viene esplicitata la matrice 3 introdotta nell'eq. (2.1):

$$M = \begin{pmatrix} \alpha_u r_{11} + u_0 r_{31} & \alpha_u r_{12} + u_0 r_{32} & \alpha_u r_{13} + u_0 r_{33} & \alpha_u t_x + u_0 t_z \\ \alpha_v r_{21} + v_0 r_{31} & \alpha_v r_{22} + v_0 r_{32} & \alpha_v r_{23} + v_0 r_{33} & \alpha_v t_y + v_0 t_z \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix}$$

la quale può essere scritta come:

$$M_i = \begin{pmatrix} N_i & n_i \end{pmatrix}$$

dove N_i è una matrice 3×3 e n_i è un vettore tridimensionale.

Essendo N_i invertibile nel caso generale, visto che $(r_{11}\ r_{12}\ r_{13})^T$, $(r_{11}\ r_{12}\ r_{13})^T$ e $(r_{11}\ r_{12}\ r_{13})^T$ sono ortogonali e $\alpha_u \neq 0, \alpha_v \neq 0$, l'equazione (2.2) può essere scomposta in

$$N_2 R_Y = N_1 R_Y R_B \tag{2.6}$$

e

$$N_2 t_Y + n_2 = N_1 R_Y t_B + N_1 t_Y + n_1. \tag{2.7}$$

Introducendo $N = N_1^{-1} N_2$ possiamo ottenere dalla eq. (2.6)

$$N R_Y = R_Y R_B$$

$$N = R_Y R_B R_Y^T$$

Due proprietà di N vengono sottolineate:

1. N è prodotto di tre matrici di rotazione, essendo quindi essa stessa una matrice di rotazione, per la quale vale la proprietà $N^{-1} = N^T$.

2. Essendo R_Y una matrice ortogonale, l'equazione sopra rappresenta una trasformazione simile, e quindi, analogamente con quanto succedeva nella formulazione originale, N e R_B hanno gli stessi autovalori, dei quali uno unitario.

Possiamo quindi scrivere

$$NR_Y n_B = R_Y R_B n_B = R_Y n_B$$

da cui si ricava

$$n_N = R_Y n_B$$

dove con n_N e n_B si intendono gli autovettori associati all'autovalore unitario rispettivamente della matrice N e B.

Premoltiplicando l'eq. (2.7) per N_1^{-1} si ottiene

$$(N - I)t_Y = R_Y t_B - t_N$$

con

$$t_N = N_1^{-1}(n_2 - n_1)$$

A questo punto è immediato riconoscere che queste ultime equazioni sono identiche alla classica formulazione fornita in precedenza.

Soluzione proposta

Avendo dimostrato che le due formulazioni sono identiche, Horaud e Dornaika procedono a proporre diversi approcci per la risoluzione del problema in entrambe le sue formulazioni.

Le due formulazioni possono essere generalizzate nella forma di due set di equazioni

$$v' = Rv$$

$$(K - I)t = Rp - p'$$

Dove R e t sono i parametri da stimare (rotazione e traslazione), v', v, p' e p sono vettori tridimensionali, K è una matrice ortogonale 3×3 e I è la matrice identità.

Per ogni movimento dell'apparato Hand Eye si ottiene una coppia di equazioni, ed almeno due spostamenti sono necessari per stimare R e t. Nel caso generale, avendo n pose, si ottengono $2n$ equazioni, risolte minimizzando due funzioni rappresentanti l'errore della soluzione:

$$f_1(R) = \sum_{i=1}^{n} \|v_i' - Rv_i\|^2 \tag{2.8}$$

$$f_2(R,t) = \sum_{i=1}^{n} \|Rp_i - (K_i - I)t - p_i'\|^2 \tag{2.9}$$

A questo punto vengono proposte due soluzioni:

1. R then t. La rotazione viene stimata per prima minimizzando f_1 nell'eq. (2.8). Una volta ottenuta la rotazione la minimizzazione di f_2 nell'eq.(2.9) rispetto a t è un problema lineare risolvibile tramite minimi quadrati.

2. R and t. Rotazione e traslazione vengono stimate contemporaneamente tramite minimizzazione di $f_1 + f_2$. Il problema così posto non è lineare, fornisce tuttavia una soluzione più stabile.

Rotation then Translation

Per minimizzare f_1 nell'eq. (2.8) si rappresenta la rotazione come un quaternione unitario (vedere appendice A). Tramite questa rappresentazione si può scrivere

$$Rv_i = q * v_i * \bar{q}$$

Inoltre, sfruttando la proprietà per cui per due quaternioni r e q vale l'uguaglianza

$$\|r * q\|^2 = \|r\|^2 \|q\|^2$$

e riscrivendo la moltiplicazione fra quaternioni sfruttando l'eq (A.1) possiamo riscrivere

$$\begin{aligned}\|v' - q * v_i * \bar{q}\|^2 &= \|v'_i - q * v_i * \bar{q}\|^2 \|q\|^2 \\ &= \|v'_i * q - q * v_i\|^2 \\ &= (Q(v'_i)q - W(v_i)q)^T (Q(v'_i)q - W(v_i)q) \\ &= q^T A_i q\end{aligned}$$

con A_i matrice 4×4 simmetrica

$$A_i = (Q(v'_i) - W(v_i))^T (Q(v'_i) - W(v_i))$$

La funzione errore può quindi essere riscritta come:

$$\begin{aligned}f_1(R) &= f_1(q) \\ &= \sum_{i=1}^{n} \|v'_i * q - q * v_i\|^2 \\ &= \sum_{i=1}^{n} q^T A_i q \\ &= q^T \left(\sum_{i=1}^{n} A_i \right) \\ &= q^T A q\end{aligned}$$

da minimizzare con il vincolo che q sia un quaternione unitario.

Questo problema di minimizzazione può essere risolto utilizzando il metodo dei moltiplicatori di Lagrange ottenendo:

$$\min_q f_1 = \min_q (q^T A q + \lambda(1 - q^T q))$$

La quale ha una soluzione in forma chiusa:

$$Aq = \lambda q$$

Il quaternione unitario, e quindi la rotazione, che minimizza la funzione errore f_1 è perciò l'autovettore di A associato al suo autovalore minore (positivo). Ottenuta la rotazione, risolvere f_2 per trovare la traslazione minore è un semplice problema lineare, risolvibile tramite il metodo dei minimi quadrati.

Rotation and Translation

Per evitare il problema che già Tsai e Lenz hanno notato, dovuto al calcolo della stima della traslazione a partire dalla stima della rotazione, viene formulato un nuovo approccio in grado di stimare contemporaneamente rotazione e traslazione, portando quindi ad una soluzione più stabile.

Il problema si può sintetizzare nella minimizzazione della funzione

$$min_{q,t}(f_1 + f_2)$$

la quale richiede la minimizzazione della somma dei quadrati di funzioni non lineari.

Utilizzando i quaternioni unitari per rappresentare le rotazioni, la funzione da minimizzare può essere scritta come

$$min_{q,t}(f(q,t) + \lambda(1 - q^T q)^2) \tag{2.10}$$

con

$$\begin{aligned} f(q,t) &= \lambda_1 f_1(q) + \lambda_2 f_2(q,t) \\ &= \lambda_1 \sum_{i=1}^{n} \|v_i' - q * v_i * \bar{q}\|^2 + \lambda_2 \sum_{i=1}^{n} \|q * p_i * \bar{q} - (K_i - I)t - p_i'\|^2 \end{aligned}$$

che ha anch'essa la forma di una somma di quadrati di funzioni non lineari e $\lambda(1-q^T q)^2$ è una funzione che garantisce che q (un quaternione) abbia modulo unitario.

Ci sono due alternative per risolvere il problema di minimizzazione dell'eq. (2.10). La prima possibilità è quella di risolvere utilizzando metodi standard di minimizzazione non lineare, come il metodo di Levenberg-Marquardt o il metodo di Newton. In alternativa si può provare a semplificare l'espressione della funzione errore da minimizzare, così come è stato fatto per f_1, sfruttando alcune proprietà dei quaternioni.

L'equazione per f_1 è già stata semplificata, e si può applicare un procedimento analogo per semplificare f_2.

Infatti f_2 è la sommatoria di termini

$$\|q * p_i * \bar{q} - (K_i - I)t - p_i'\|^2$$

da cui ricaviamo

$$\|q * p_i * \bar{q} - (K_i - I)t - p_i'\|^2 \|q\|^2 = \|q * p_i - (K_i - I)t * q - p_i' * q\|^2$$

Rappresentando matricialmente la moltiplicazione per quaternioni si può ricavare una forma scomposta della funzione f_2:

$$f_2(q,t) = q^T(\sum_{i=1}^{n} \beta_i)q + t^T(\sum_{i=1}^{n} \zeta_i)t + (\sum_{i=1}^{n} \delta_i)t + (\sum_{i=1}^{n} \xi_i)Q(q)^T W(q)t$$

Con le matrici 4×4 β_i e ζ_i

$$\begin{aligned} \beta_i &= (p_i^T p_i + p_i'^T p_i')I - W(p_i)^T Q(p_i') - Q(p_i')^T W(p_i) \\ \zeta_i &= K_i^T K_i - K_i - K_i^T + I \end{aligned}$$

E i vettori 1×4 δ_i e η_i

$$\begin{aligned} \delta_i &= 2p_i'^T(K_i - I) \\ \xi_i &= -2p_i^T(R_{Bi} - I) \end{aligned}$$

Eliminando le sommatorie sostituendo $\beta = \sum_{i=1}^{n} \beta_i$, $\zeta = \sum_{i=1}^{n} \zeta_i$, $\delta = \sum_{i=1}^{n} \delta_i$, $\xi = \sum_{i=1}^{n} \xi_i$ e con A già ricavato da f_1 si ottiene il problema di minimizzazione non lineare seguente:

$$\min_{q,t}(q^T)(A + \beta)q + t^T \zeta t + \delta t + \xi Q(q)^T W(q)t + \lambda(1 - q^T q)^2)$$

Il problema consiste in una somma di 5 termini nel quale occorre stimare 7 parametri, 4 per il quaternione unitario e 3 per la traslazione. Un tale problema di minimizzazione può essere risolto utilizzando tecniche classiche di ottimizzazione come tecniche del tipo *trust regions*.

Analisi dei risultati

Horaud e Dornaika analizzano la stabilità e la precisione dei due approcci risolutivi comparati all'approccio originale utilizzato da Tsai e Lenz, cercando di misurare qualitativamente quale metodo dia risultati migliori.

Nell'analisi del problema si giunge all'individuazione di due principali cause di errore: errori associati alla calibrazione della camera ed errori associati al movimento del robot. A causa di questi errori i movimenti del robot e della camera sono conosciuti con un determinato grado di incertezza rispetto al loro movimento reale.

Per simulare tali errori vengono generati artificialmente diversi set di dati, sporcati successivamente con un errore generato da una distribuzione uniforme o gaussiana, analizzando il risultato ottenuto sui diversi set dai diversi approcci presentati a confrontandone le prestazioni rispetto al numero di stazioni utilizzate, alla quantità di errore del set o al tipo stesso di errore.

Il vantaggio della nuova formulazione proposta consiste nel poter confrontare i tre approcci[3] fra di loro su entrambe le formulazioni, essendo queste equivalenti, per poter analizzare nel dettaglio il risultato.

Il primo risultato interessante è il quasi identico comportamento del metodo Tsai-Lenz rispetto al primo approccio proposto. Il motivo risiede nella scomposizione fra rotazione e traslazione, tecnica utilizzata in entrambi i metodi. Nonostante la stima della rotazione avvenga in modi diversi l'errore che inevitabilmente viene commesso durante tale stima si propaga successivamente nella stima della traslazione, portando a simili risultati.

Per lo stesso motivo il secondo approccio proposto, stimando contemporaneamente rotazione e traslazione, porta a risultati nettamente migliori, come si può vedere in figura 2.3 indipendentemente dal tipo di rumore (uniforme o gaussiano) utilizzato per sporcare i data set. Se ne conclude che la scomposizione del problema in rotazione e traslazione porta necessariamente all'inserimento di errori, in quanto l'errore commesso nel primo step si propaga successivamente. Tale nozione era già

[3] L'approccio originale visto nel capitolo precedente con i due proposti da Horaud e Dornaika.

stata considerata da Tsai e Lenz, che difatti suggeriscono una serie di accortezze nella scelta delle stazioni, quali ad esempio la scelta di angoli drasticamente differente da una stazione alla successiva, allo scopo di minimizzare il più possibile gli errori in fase di acquisizione minimizzandone la propagazione negli step successivi.

In tutti i casi la formulazione $MY = M'YB$ produce risultati migliori alla formulazione classica del problema $AX = XB$. Tale miglioramento è dovuto, in maniera analoga a quanto succede con la scomposizione fra traslazione e rotazione nei differenti approcci risolutivi, alla rimozione della scomposizione della trasformazione camera-mondo fra parametri intrinseci ed estrinseci, evitando di conseguenza l'inserimento di errori, i quali inevitabilmente portando ad una propagazione degli stessi nelle fasi di calcolo successive amplificandone l'inesattezza.

Tale risultato non sorge inaspettato, in quanto una maggiore generalizzazione e semplificazione del problema in grado di ridurre il numero di calcoli necessari, mantenendo tuttavia il problema equivalente all'originale, diminuisce la propagazione degli errori, migliorandone di conseguenza l'accuratezza.

Horaud e Dornaika compiono anche alcuni test su dati reali, dove tuttavia i risultati ottenuti si avvicinano fra di loro più del previsto. Tale risultato è principalmente dovuto alla non assoluta precisione dei robot utilizzati per i test, i quali portano evidentemente un errore significativo, portando le tre soluzioni ad allinearsi. Tuttavia il metodo non lineare risulta in ogni caso leggermente migliore rispetto ai precedenti, fornendo una soluzione più precisa.

La nuova formulazione proposta invece, a differenza della formulazione classica, porta un errore leggermente maggiore nella stima della rotazione, ma un netto miglioramento nella parte traslatoria della trasformazione. Questo risultato non deve sorprendere, in quanto le due parti vengono stimate contemporaneamente con un certo errore, che nel caso del robot reale utilizzato è maggiore nella sua componente traslatoria. Nella formulazione classica la parte rotatoria, molto precisa, viene stimata senza considerare la componente traslatoria, meno precisa, ottenendo perciò un risultato molto accurato. Tale risultato non è tuttavia esente da errore, il quale viene propagato nella componente traslatoria,

di per se imprecisa, portando ad un errore notevole. Nella formulazione proposta da Horaud e Dornaika invece la parte traslatoria, meno precisa, viene valutata contemporaneamente alla parte rotatoria, portando un errore modesto in entrambe le parti della trasformazione, ottenendo in conclusione un risultato migliore.

In figura 2.3 si possono vedere nel dettaglio i risultati ottenuti da Horaud e Dornaika, pubblicati nel loro paper.

Nella seconda colonna è presente l'errore di rotazione, calcolato come somma dei quadrati degli errori assoluti di J prove:

$$e_{rot} = \sqrt{\frac{1}{J}\sum_{j=1}^{J} \|\tilde{R}_j - R\|^2}$$

mentre nella terza colonna è presente l'errore effettuato nel calcolo della traslazione, anch'esso calcolato come somma dei quadrati degli errori assoluti in J prove:

$$e_{tr} = \frac{\sqrt{\frac{1}{J}\sum_{j=1}^{J} \|\tilde{R}_j - R\|^2}}{\|t\|}$$

2.2.3 Daniilidis (1999), using dual quaternions

Kostantinos Daniilidis, partendo da quanto visto in precedenza, propone un ulteriore miglioramento agli algoritmi di Hand Eye calibration introducendo l'utilizzo dei quaternioni duali, presentati in appendice B. Questa idea nasce dal fatto che gli algoritmi esistenti non trattano in maniera unificata traslazione e rotazione che invece, similmente a quanto già dimostrato in precedenza, comporta un miglioramento della stima delle trasformazioni. La parametrizzazione per quaternioni duali, controparte algebrica delle *screw*, come visto in appendice B, permette di ottenere una nuova soluzione di calibrazione simultanea di rotazione e traslazione utilizzando la scomposizione per Valori Singolari.

Daniilidis riprende il lavoro di Horaud e Dornaika, i primi a proporre un approccio di minimizzazione non lineare in grado di calcolare contemporaneamente i vettori di traslazione e rotazione, proponendo tuttavia

$AX = XB$	$\sum \|R_A R_X - R_X R_B\|^2$	$\frac{\sum \|(R_A - I)t_X - R_X t_B + t_A\|^2}{\sum \|R_X t_B - t_A\|^2}$
Tsai-Lenz	0.0006	0.032
Closed-form solution	0.0005	0.029
Non-linear optimization	0.0003	0.019

$AX = XB$	$\sum \|R_A R_X - R_X R_B\|^2$	$\frac{\sum \|(R_A - I)t_X - R_X t_B + t_A\|^2}{\sum \|R_X t_B - t_A\|^2}$
Tsai-Lenz	0.0014	0.036
Closed-form solution	0.0014	0.023
Non-linear optimization	0.0017	0.015
$MY = M'YB$	$\sum \|NR_Y - R_Y R_B\|^2$	$\frac{\sum \|(N - I)t_Y - R_Y t_B + t_N\|^2}{\sum \|R_Y t_B - t_N\|^2}$
Tsai-Lenz	0.0031	0.0021
Closed-form solution	0.0015	0.001
Non-linear optimization	0.0013	0.0006

$AX = XB$	$\sum \|R_A R_X - R_X R_B\|^2$	$\frac{\sum \|(R_A - I)t_X - R_X t_B + t_A\|^2}{\sum \|R_X t_B - t_A\|^2}$	CPU time
Tsai-Lenz	0.014	0.23	0.08
Closed-form solution	0.036	0.223	0.06
Non-linear optimization	0.258	0.058	0.21
$MY = M'YB$	$\sum \|NR_Y - R_Y R_B\|^2$	$\frac{\sum \|(N - I)t_Y - R_Y t_B + t_N\|^2}{\sum \|R_Y t_B - t_N\|^2}$	
Tsai-Lenz	0.038	0.039	0.06
Closed-form solution	0.035	0.037	0.08
Non-linear optimization	0.04	0.034	0.25

Figura 2.3: Tabelle contenente i risultati pubblicati da Horaud e Dornaika. La prima tabella mostra i risultati ottenuti su un dataset composto da 17 stazioni, fornito da Francois Chaumette di IRISA, la seconda e terza tabella mostrano i risultati ottenuti su due diversi set forniti da LIFIA composti rispettivamente da 7 e 6 stazioni. Nel primo dataset è stato possibile utilizzare soltanto la formulazione classica in quanto il dataset era composto dai parametri estrinseci; nei due dataset successivi è stato possibile utilizzare la nuova formulazione, in quanto si avevano a disposizione le intere matrici prospettiche. Nella terza tabella viene inoltre mostrata, nella la quarta colonna, il tempo di calcolo impiegato in secondi. *Courtesy of Horaud and Dornaika.*

di rappresentare la trasformazione sfruttando la *screw theory*, introdotta inizialmente nell'Hand Eye calibration da Chen[10] nel 1991.

Tramite i quaternioni duali viene provato che

1. il calcolo della trasformazione Hand Eye, se rappresentata tramite quaternioni duali, è indipendente da angolo di rotazione e traslazione sull'asse di rotazione dei movimenti della camera e della mano, bensì dipende unicamente dai parametri dell'asse di rotazione, come provato da Chen[10], e che

2. le incognite della *screw* possono essere stimate contemporaneamente tramite scomposizione per valori singolari.

L'utilizzo di scomposizione per valori singolari porta Daniilidis ad essere il primo a proporre un algoritmo in grado di risolvere simultaneamente rotazione e traslazione senza utilizzare tecniche di minimizzazione non lineare.

Trasformazione di una retta con quaternioni duali

É nota la possibilità di esprimere la rotazione di un punto $\vec{p}$ tramite un quaternione unitario nella forma $\vec{p_a} = q\vec{p_b}\bar{q}$. L'utilizzo di tale formula permette di esprimere una concatenazione di rotazioni tramite una semplice moltiplicazione di quaternioni. Tuttavia non esiste una rappresentazione simile in grado di rappresentare una trasformazione rigida generica che includa anche una traslazione all'interno di essa. É possibile tuttavia rappresentare in maniera simile una trasformazione rigida generica analogamente a quanto succede con le rotazioni pure introducendo la nozione di quaternione duale, applicabile ad una retta invece che ad un punto.

Una retta nello spazio con direzione $\vec{l}$ passante per un punto $\vec{p}$ può essere rappresentata da una tupla $(\vec{l}, \vec{m})$ dove $\vec{m}$ viene chiamato momento della retta ed è uguale a $\vec{p} \times \vec{l}$. Il momento della retta è normale al piano tra retta e origine ed ha valore assoluto pari alla distanza da retta ad origine. I vincoli $\|\vec{l}\| = 1$ e $\vec{l}^T\vec{m} = 0$ garantiscono che i gradi di libertà di una retta arbitraria siano 4.

Applicando una rotazione R e una traslazione $\vec{t}$ ad una retta $(\vec{l_b}, \vec{m_b})$ si ottiene la retta trasformata $(\vec{l_a}, \vec{m_a})$:

$$\begin{aligned}\vec{l_a} &= R\vec{l_b} \\ \vec{m_a} &= \vec{p_a} \times \vec{l_a} \\ &= (R\vec{p_b} + \vec{t}) \times R\vec{l_b} \\ &= R(\vec{p_b} \times \vec{l_b} + \vec{t}) \times R\vec{l_b} \\ &= R\vec{m_b} + \vec{t} \times R\vec{l_b}\end{aligned}$$

É possibile riscrivere tale trasformazione utilizzando i quaternioni, rappresentando $\vec{l}$ come un quaternione $l = (0, \vec{l})$, ed introducendo il quaternione t rappresentate la traslazione $(0, \vec{t})$ ed il quaternione q rappresentante la rotazione $(0, \vec{q})$.

Sfruttando

$$(0, \vec{t} \times \vec{q}) = \frac{1}{2}(q\bar{t} + tq)$$

è possibile riscrivere la trasformazione:

$$\begin{aligned}l_a &= q l_b \bar{q} \\ m_a &= q m_b \bar{q} + \frac{1}{2}(q l_b \bar{q} \bar{t} + t q l_b \bar{q})\end{aligned}$$

Si può a questo punto definire un nuovo quaternione $q' = \frac{1}{2}tq$ ed un quaternione duale $\check{q} = q + \epsilon q'$, riscrivendo l'equazione precedente

$$l_a + \epsilon m_a = (q + \epsilon q')(l_b + \epsilon m_b)(\bar{q} + \epsilon \bar{q'})$$

Scrivendo anche le rette come quaternioni duali $\check{l_a}$ e $\check{l_b}$ si ottiene

$$\check{l_a} = \check{q} \check{l_b} \bar{\check{q}}$$

la quale risulta molto simile alla formula per la rotazione di punti tramite quaternioni.

Essendo inoltre la norma

$$\begin{aligned}\bar{\check{q}}^2 &= \check{q}\bar{\check{q}} = q\bar{q} + \epsilon(q\bar{q'} + q'\bar{q}) \\ &= q\bar{q} + \epsilon/2(q\bar{q}\bar{t} + tq\bar{q}) = 1\end{aligned}$$

ne risulta che $\check{q}$ è un quaternione duale unitario. Se ne ricava esplicitamente inoltre la trasformazione da $(R, \vec{t})$ a $q+\epsilon q'$. La parte duale $q' = \frac{1}{2}tq$ e il quaternione q si ottengono dalla matrice di rotazione trovando l'asse e l'angolo di rotazione.

Se $\check{q}$ è una soluzione, anche $-\check{q}$ è una soluzione, perciò genericamente si definisce la parte scalare non duale del quaternione positiva, in modo da evitare questa ambiguità.

Di conseguenza, la traslazione t si ottiene dal quaternione duale come

$$t = 2q'\bar{q}$$

Il quaternione duale unitario $\check{q}$ può perciò essere espresso come la concatenazione di un quaternione duale unitario puro rappresentante la traslazione con un quaternione puro rappresentante la rotazione con parte duale nulla; i.e.

$$\check{q} = (1, \epsilon\frac{\vec{t}}{2})q$$

Screw Theory con quaternioni duali unitari

Secondo il teorema di Chasles[10] una trasformazione rigida può essere modellata come una rotazione attorno ad un asse non passante per l' origine, seguita da una traslazione sullo stesso asse, chiamato *screw*.

Essendo quest'asse una retta nello spazio è descritta da 4 parametri, come visto in precedenza, che insieme ad un angolo θ e ad una traslazione d costituiscono i sei gradi di libertà di una trasformazione rigida generica.

É possibile dimostrare che data una rotazione R attorno ad un asse passante per l'origine ed una traslazione $\vec{t}$ è possibile calcolare la traslazione d e l'asse screw composto dalla coppia direzione e momento $(\vec{l}, \vec{m})$.

La direzione $\vec{l}$ è parallela all'asse di rotazione, la traslazione d è la proiezione della traslazione sull'asse di rotazione, ed è perciò uguale a $\vec{t}^T\vec{l}$ mentre l'angolo θ è uguale sia per $(R, \vec{t})$ e la rappresentazione screw. Per calcolare il momento $\vec{m}$ viene introdotto il punto $\vec{c}$ sull'asse screw come proiezione dell'origine sull'asse di rotazione, come mostrato in figura

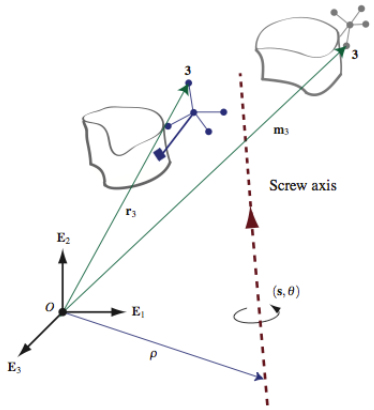


Figura 2.4: Una rappresentazione grafica della screw theory, che mostra come qualsiasi trasformazione possa essere associata ad una rotazione attorno ad un asse e ad una successiva traslazione sull'asse stesso. La trasformazione ricorda un movimento elicoidale dell'immagine attorno ad un asse di rotazione, da cui il nome della teoria.

Il sistema di riferimento viene traslato in questa posizione e poi ruotato. La traslazione risultante è quindi $d\vec{l} + (I - R)\vec{c}$.

A questo punto è possibile ricavare la traslazione d come $d = \vec{l}^T\vec{t}$. Utilizzando la formula di Rodrigues

$$R\vec{c} = \vec{c} + \sin(\theta)\vec{l} \times \vec{c} + (1 - \cos\theta)\vec{l} \times (\vec{l} \times \vec{c})$$

e sapendo che $\vec{c}^T\vec{l} = 0$ ne segue che

$$\vec{c} = \frac{1}{2}(\vec{t} - (\vec{t}^T\vec{l})\vec{l} + \cot\frac{\theta}{2}\vec{l} \times \vec{t})$$

$$\vec{m} = \vec{c} \times \vec{l} = \frac{1}{2}(\vec{t} \times \vec{l} + \vec{l} \times (\vec{t} \times \vec{l})\cot\frac{\theta}{2})$$

Con il punto $\vec{c}$ non definito per angoli θ di valore 0 o 180°, per i quali l'asse screw non è definito.

Avendo a disposizione tutti i parametri della screw $(\theta, d, \vec{l}, \vec{m})$ è possibile definire il corrispondente quaternione duale $\check{q}$.

Partendo dal quaternione derivato dalla rotazione R

$$(q_0, \vec{q}) = (\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\vec{l}) \tag{2.11}$$

e sapendo che $(\vec{l}^T\vec{t}) = d$, si può riscrivere il momento $\vec{m}$ calcolato in precedenza:

$$\sin\frac{\theta}{2}\vec{m} = \frac{1}{2}(\vec{t}\times\vec{q} + q_0\vec{t} - \cos\frac{\theta}{2}(\vec{l}^T\vec{t})\vec{l})$$
$$\sin\frac{\theta}{2}\vec{m} + \frac{d}{2}\cos\frac{\theta}{2}\vec{l} = \frac{1}{2}(\vec{t}\times\vec{q} + q_0\vec{t})$$

che è la parte vettoriale della parte duale q' del quaternione duale $\check{q}$.

Utilizzando $q' = \frac{1}{2}tq$ si ottiene la prima rappresentazione del quaternione duale $\check{q}$:

$$\begin{aligned}\check{q} &= \begin{pmatrix} q_0 \\ \vec{q} \end{pmatrix} + \epsilon \begin{pmatrix} -\frac{1}{2}\vec{q}^T\vec{t} \\ \frac{1}{2}(q_0\vec{t} + \vec{t}\times\vec{q}) \end{pmatrix} \\ &= \begin{pmatrix} \cos\frac{\theta}{2} \\ \sin\frac{\theta}{2}\vec{l} \end{pmatrix} + \epsilon \begin{pmatrix} -\frac{d}{2}\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2}\vec{m} + \frac{d}{2}\cos\frac{\theta}{2}\vec{l} \end{pmatrix}\end{aligned}$$

la quale può essere semplificata sfruttando la proprietà dei quaternioni duali per cui

$$f(a + \epsilon b) = f(a) + \epsilon b f'(a)$$

ottenendo una rappresentazione semplificata di $\check{q}$

$$\check{q} = \begin{pmatrix} \cos(\frac{\theta+\epsilon d}{2}) \\ \sin(\frac{\theta+\epsilon d}{2})(\vec{l} + \epsilon\vec{m}) \end{pmatrix} \tag{2.12}$$

Questa rappresentazione di $\check{q}$ è molto potente in quanto separa algebricamente l'angolo e la traslazione d dalle informazioni della retta che caratterizzano la posizione dell'asse screw. Inoltre, ponendo l'angolo duale $\check{\theta} = \theta + \epsilon d$ ed il vettore duale $\check{\vec{l}} = \vec{l} + \epsilon\vec{m}$ si ottiene che l'eq. (2.12) è equivalente alla rotazione pura dell'eq. (2.11).

Si può facilmente verificare che

$$\check{q} = (\cos\frac{\check{\theta}}{2}, \check{\vec{l}}\sin\frac{\check{\theta}}{2})$$

è un quaternione unitario $\check{q}\bar{\check{q}} = 1$.

Hand Eye Calibration con quaternioni duali

A questo punto, una volta assodato che una trasformazione rigida può essere completamente descritta da un quaternione duale, è possibile formulare il problema di Hand Eye Calibration utilizzando questa nuova rappresentazione delle trasformazioni, ottenendo

$$\check{a} = \check{q}\check{b}\bar{\check{q}} \tag{2.13}$$

Dove $\check{a}$ è la trasformazione fra due stazioni della camera e $\check{b}$ è la trasformazione fra due stazione del robot, entrambe rappresentate come quaternioni duali, mentre $\check{q}$ rappresenta invece la trasformazione fra camera e robot, ovvero la calibrazione risultato.

Si può inoltre dimostrare che le parti scalari dell'equazione proposta sono congruenti, infatti
$Sc(\check{a}) = \frac{1}{2}(\check{a} + \bar{\check{a}}) = \frac{1}{2}(\check{q}\check{b}\bar{\check{q}} + \check{q}\bar{\check{b}}\bar{\check{q}}) = \frac{1}{2}\check{q}(\check{b} + \bar{\check{b}})\bar{\check{q}} = \check{q}Sc(\check{b})\bar{\check{q}} = Sc(\bar{\check{b}})\check{q}\bar{\check{q}} = Sc(\check{b})$

Da cui si ricava che i parametri dell'angolo duale, ovvero la parte reale del quaternione duale, i quali rappresentano l'angolo θ di rotazione attorno all'asse e la traslazione d effettuata successivamente di robot e camera non influiscono sul calcolo del quaternione $\check{q}$ e quindi sul risultato.

L'equazione fondamentale (2.13) consiste in quattro equazioni duali, non siamo tuttavia interessati alle parti scalari poiché, come appena dimostrato, sono congruenti. Solo le parti vettoriali contribuiscono al calcolo di $\check{q}$:

$$\sin\frac{\check{\theta}_a}{2}(0, \check{\vec{a}}) = \check{q}(0, \sin\frac{\check{\theta}_b}{2}\check{\vec{b}})\bar{\check{q}} = \sin\frac{\check{\theta}_b}{2}\check{q}(0, \check{\vec{b}})\bar{\check{q}}$$

che con $\theta_{a,b}$ diversi da 0 e 360° si semplifica in

$$(0, \check{\vec{a}}) = \check{q}(0, \check{\vec{b}})\bar{\check{q}}$$

che non è altro che la trasformazione dell'asse di rotazione da una stazione all'altra.

Se ne ricava che la calibrazione si può semplicemente calcolare a partire dalla trasformazione di questi assi, la cui trasformazione, come abbiamo dimostrato in precedenza, è facilmente rappresentabile da operazioni con quaternioni duali.

Inoltre tutti gli altri metodi proposti in precedenza utilizzano, nel calcolo della calibrazione, i parametri inerenti all'angolo di rotazione e traslazione sull'asse, i quali, come abbiamo visto, non sono strettamente necessari.

Seppure soltanto la parte vettoriale sia utile per la stima di $\check{q}$ manteniamo la notazione $\check{a}$ e $\check{b}$ per $(0, \check{\vec{a}})$ e $(0, \check{\vec{b}})$.

Dall'eq. (2.13) si può separare la parte duale dalla parte non duale dell'equazione in

$$\begin{aligned} a &= qb\bar{q} \\ a' &= qb\bar{q}' + qb'\bar{q} + q'b\bar{q} \end{aligned}$$

le quali si possono moltiplicare per q e, applicando l'identità $\bar{q}q' + \bar{q}'q = 0$, si ottiene

$$\begin{aligned} aq - qb &= 0 \\ (a'q - qb') + (aq' - q'b) &= 0 \end{aligned}$$

per le quali, come dimostrato in precedenza, la parte scalare è ridondante.

Si ottengono in totale sei equazioni in otto incognite, le quali possono essere scritte in forma matriciale.

Ponendo $a = (0, \vec{a})$ e $a' = (0, \vec{a}')$ ed in maniera speculare $b = (0, \vec{b})$ e $b' = (0, \vec{b}')$ l'equazione sopra può essere riscritta in forma di matrice vettoriale

$$\begin{pmatrix} \vec{a} - \vec{b} & [\vec{a} + \vec{b}]_\times & 0_{3\times1} & 0_{3\times3} \\ \vec{a}' - \vec{b}' & [\vec{a}' + \vec{b}']_\times & \vec{a} - \vec{b} & [\vec{a} + \vec{b}]_\times \end{pmatrix} \begin{pmatrix} q \\ q' \end{pmatrix} = 0$$

dove la matrice, chiamata S, è una matrice 6×8, e il vettore delle incognite (q^T, q'^T) ha 8 dimensioni. $[\vec{a}]_\times$ è la matrice antisimmetrica del cross-product con $\vec{a}$.

Ricordiamo i due vincoli delle incognite, essendo q un quaternione duale unitario

$$q^T q = 1 \quad e \quad q^T q' = 0 \tag{2.14}$$

In teoria sei equazioni con i due vincoli aggiuntivi sarebbero sufficienti per risolvere le 8 incognite del sistema, tuttavia i versori $\vec{a}$ e $\vec{b}$ sono perpendicolari ai vettori $\vec{a}'$ e $\vec{b}'$, rendendo le due equazioni ridondanti. Per questo motivo, come è ben noto in letteratura, almeno due movimenti sono necessari per calcolare correttamente le incognite della calibrazione.

Supponiamo a questo punto di avere un numero di spostamenti $n \geq 2$, si può costruire la matrice $6n \times 8$

$$T = (\ S_1^T \quad S_2^T \quad \dots \quad S_n^T\ )^T$$

la quale, in assenza di rumore o errori, ha rango 6, ed il suo spazio nullo contiene la soluzione (q, q'). Una soluzione ortogonale ammissibile consiste anche in $(0_{4\times1}, q)$, per cui la matrice è al massimo di rango 6, 5 se tutti gli assi $\vec{b}$ sono paralleli fra loro.

Risulta banale a questo punto effettuare la decomposizione per valori singolari (SVD) della matrice $T = U\Sigma V^T$, dove Σ è una matrice diagonale di valori singolari, le colonne di U i vettori singolari sinistri e le colonne di T sono i vettori singolari destri. Se il rango della matrice è 6 ci troviamo di fronte ad un problema di deficienza di rango, di conseguenza gli ultimi due vettori singolari destri $\vec{v_7}$ e $\vec{v_8}$ ricoprono lo spazio nullo di T.

Possiamo scrivere i vettori come composizione di due vettori 4×1, ottenendo $\vec{v_7}^T = (\vec{u_1}^T, \vec{v_1}^T)$ e $\vec{v_8}^T = (\vec{u_2}^T, \vec{v_2}^T)$.

Il vettore (q^T, q'^T) che soddisfa $T(q^T, q'^T)^T = 0$ è una combinazione lineare di $\vec{v_7}$ e $\vec{v_8}$, ovvero

$$\begin{pmatrix} q \\ q' \end{pmatrix} = \lambda_1 \begin{pmatrix} \vec{u_1} \\ \vec{v_1} \end{pmatrix} + \lambda_2 \begin{pmatrix} \vec{u_2} \\ \vec{v_2} \end{pmatrix}$$

Si può ricavare λ_1 e λ_2 dai vincoli espressi nell'eq. (2.14) costruendo le due equazioni:

$$\lambda_2 \vec{u_1}^T \vec{u_1} + 2\lambda_1\lambda_2 \vec{u_1}^T \vec{u_2} + \lambda_2^2 \vec{u_2}^T \vec{u_2} = 1$$
$$\lambda_1 \vec{u_1}^T \vec{v_1} + \lambda_1\lambda_2(\vec{u_1}^T \vec{v_2} + \vec{u_2}^T \vec{v_1}) + \lambda_2^2 \vec{u_2}^T \vec{v_2} = 0$$

Si può porre senza perdita di generalità che $\vec{u_1}^T \vec{v_1} \neq 0$ non potendo essere mai λ_1 e λ_2 entrambi nulli, in modo da poter porre $\lambda_2 \neq 0$. Si può in questo modo sostituire $s = \lambda_1/\lambda_2$ nella prima equazione, ottenendo due soluzioni per s, ed inserire $\lambda_1 = s\lambda_2$ nella seconda equazione riscrivendo

$$\lambda_2^2(s^2 \vec{i_1}^T \vec{u_1} + 2s\vec{u_1}^T \vec{u_2} + \vec{u_2}^T \vec{u_2}) = 1$$

avendo come risultato due soluzioni reali di segno opposto, essendo $(s^2 \vec{i_1}^T \vec{u_1} + 2s\vec{u_1}^T \vec{u_2} + \vec{u_2}^T \vec{u_2})$ sempre positivo o tuttalpiù nullo.

In presenza di rumore, per evitare la soluzione per s che dia come risultato $(0_{4\times1}, q)$, si sceglie la soluzione per s che dia il valore di $(s^2 \vec{i_1}^T \vec{u_1} + 2s\vec{u_1}^T \vec{u_2} + \vec{u_2}^T \vec{u_2})$ maggiore, mentre la variazione di segno per λ_2 si spiega poiché sia (q^T, q'^T) che $(-q^T, -q'^T)$ soddisfano le equazioni ed i vincoli posti.

Si ottiene così λ_1 e λ_2, la cui composizione fornisce il risultato della calibrazione:

$$\check{q} = \lambda_1 \vec{v_7} + \lambda_2 \vec{v_8}$$

Miglioramento rispetto ai metodi precedenti

Daniliidis confronta il metodo appena descritto con due metodi sviluppati in precedenza.

Il primo metodo confrontato è il metodo non lineare proposto da Horaud e Dornaika, nel particolare viene confrontato con il metodo utilizzato per il calcolo simultaneo di traslazione e rotazione nella formulazione del problema $MY = M'YB$, minimizzando utilizzando il metodo di Levenberg-Marquardt.

Il secondo metodo confrontato è una implementazione del metodo descritto da Chou e Kamel[11], simile al metodo di Tsai e Lenz visto in

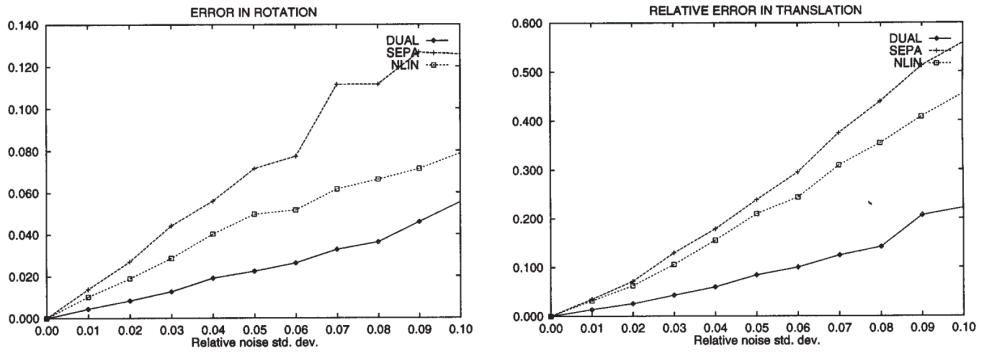


Figura 2.5: Comportamento del metodo a quaternioni duali (DUAL), del metodo non lineare proposto da Horaud e Dornaika (NLIN) e del metodo a due step separati, in cui si calcola prima rotazione poi traslazione, di Chou e Kamel (SEPA). I grafici rappresentano l'errore RMS rispetto alla deviazione standard del rumore per la rotazione, a sinistra, e per la traslazione, a destra. *Courtesy of Daniilidis.*

precedenza, il quale risolve in maniera separata rotazione e traslazione, calcolando prima la rotazione minimizzando

$$\|aq - qb\|^2 \quad \text{rispetto a} \quad q \quad \text{con} \quad \|q\|^2 = 1$$

il quale può essere ridotto ad un problema di autovettore, risolvendo poi il sistema

$$(R_A - I)\vec{t_X} = R_X\vec{t_B} - \vec{t_A}$$

per ricavare la traslazione.

I risultati ottenuti da Danilidiis mostrano un notevole miglioramento rispetto agli algoritmi proposti in precedenza, fornendo perciò un semplice nuovo algoritmo il quale ha il privilegio di evitare passi computazionali non lineari. I migliori risultati ottenuti vengono giustificati non soltanto grazie alla stima contemporanea di componente traslatoria e rotatoria, bensì all'utilizzo dei dati strettamente necessari al calcolo della calibrazione, scartando i dati che invece, come dimostrato, non influiscono nel calcolo della calibrazione, evitando perciò possibili fonti di errore aggiuntive.

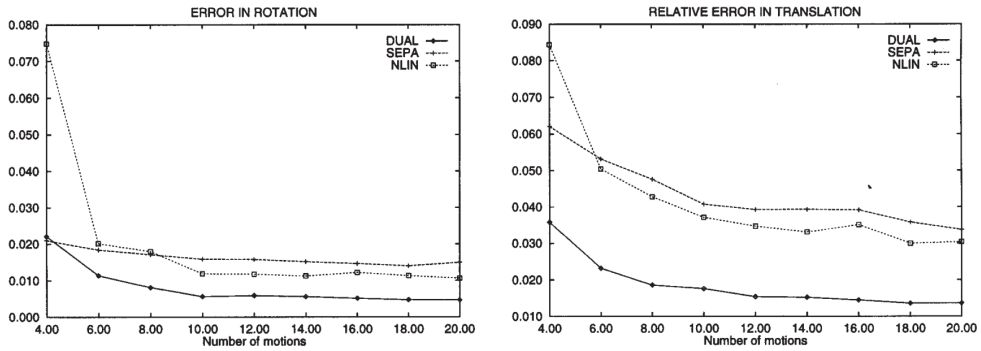


Figura 2.6: L'errore RMS rispetto al numero di movimenti del sistema per la traslazione, a sinistra, e per la rotazione, a destra. *Courtesy of Daniilidis.*

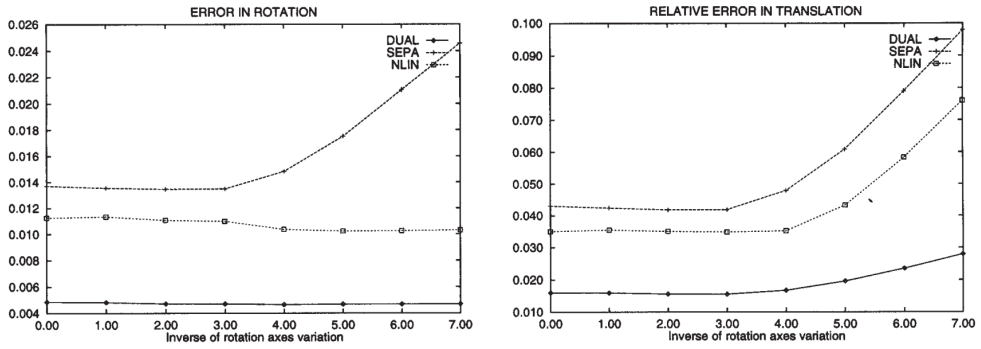


Figura 2.7: L'errore RMS come funzione dell'inversa della variazione dell'asse di rotazione. L'asse orizzontale è proporzionale all'inversa di un area di una sfera unitaria dentro la quale sono distribuiti gli assi di rotazione. *Courtesy of Daniilidis.*

2.2.4 Klaus H. Strobl and Gerd Hirzinger (2006), metric analysis

Strobl e Hirzinger[24] propongono, successivamente, un nuovo metodo di ottimizzazione per il problema di calibrazione, ponendo l'attenzione sulla minimizzazione. Dopo uno studio delle tecniche precedenti propongono una tecnica basata sulla parametrizzazione del problema come modello stocastico[4], proponendo una metrica nel gruppo delle trasformazioni rigide $SE(3)$ ed il corrispondente errore da utilizzare per l'ottimizzazione.

Viene notata l'assenza, con tutte le soluzioni precedenti, della possibilità di aggiungere dei pesi alle misure di posizione/orientazione da utilizzare per aiutare a giungere alla soluzione ottima. Inoltre la convergenza di un algoritmo alla soluzione ottima dipende fortemente sia dalla scelta delle combinazioni lineari dei parametri utilizzati (*preconditioning*) sia dalla normalizzazione delle variabili scelte rispetto alla struttura del problema (*variable scaling*).

Viene perciò proposta una funzione obiettivo Υ basata su una metrica di errore in $SE(3)$ con lo scopo di:

- Trovare la soluzione ottima del problema, ottimizzando l'errore
- Dare la possibilità di specificare dei pesi alle componenti rotazionali e traslazionali
- Aggiustare automaticamente tali pesi

Inoltre l'algoritmo che sfrutta la soluzione proposta può essere applicato sia alla formulazione classica $AX = XB$ sia alla formulazione proposta da Horaud e Dornaika $MY = M'YB$.

Formulazione del problema di Hand Eye Calibration Il metodo per stimare ottimamente la calibrazione (X o Y a seconda della formulazione utilizzata) consiste nel correggere ottimamente gli errori commessi in ogni stazione A e B (o M ed M') causati sia dall'errore geometrico del

[4]Un modello nel quale si tengono in considerazione le variazioni delle variabili di input, fornendo risultati in termini di probabilità, al contrario dei modelli deterministici, per i quali gli input sono fissi e non vi è incertezza sull'esecuzione.

modello (ovvero la misurazione effettiva) sia dalla stima effettuata dalle formule.

Il metodo di massima verosimiglianza (*Maximum Likelihood*, ML) sceglie il modello di soluzione per cui è più alta la probabilità di appartenenza dei dati osservati, o, in altre parole, per cui i dati si discostano meno dal modello. Si tratta cioè di determinare uno stimatore che massimizzi la funzione di verosimiglianza, definita in base alla probabilità di osservare una data realizzazione campionaria, condizionatamente ai valori assunti dai parametri statistici oggetto di stima.

Per una distribuzione che si ipotizza gaussiana degli errori si utilizza, solitamente, una funzione di verosimiglianza equivalente alla somma delle covarianze al quadrato, pesate, degli errori misurati.

Metrica per errore rotazionale Qualsiasi trasformazione, come già visto in precedenza, può essere modellata come una rotazione in $SO(3)$ di un angolo θ attorno ad un asse p passante per l'origine seguita da una traslazione t in $\mathbb{R}^3$.

Viene proposta la metrica

$$\Upsilon_i^{rot} = \Delta_t\theta_i = \arccos\left(\frac{trace(\Delta_t\tilde{R}_i) - 1)}{2}\right) = \Delta_b\theta_i$$

con

$$\Delta_t\tilde{R}_i = {}_t\tilde{R}_i^{\,b}\,{}_b\check{R}_i^{\,t}$$

dove $\Delta_t\tilde{R}_i$ è l'errore residuo, ${}_b\tilde{R}_i^{\,t}$ è la rotazione misurata (ad esempio dal software di controllo del robot) e ${}_b\check{R}^t$ è il valore stimato dal sistema nella trasformazione fra base del robot e tool, per il calcolo dell'errore residuo rotazionale avendo inoltre la peculiarità di essere frame invariante.

Metrica per errore traslazionale Una metrica per lo spazio Euclideo potrebbe essere la distanza Euclidea stessa, essendo una metrica naturale, tuttavia non è utilizzabile singolarmente come metrica per l'errore residuo traslazionale di una trasformazione rigida non essendo frame invariante, infatti

$${}_{ti}\tilde{t} = {}_b\check{t}^{t_i} - {}_b\tilde{t}^{t_i} \quad \text{non è uguale a} \quad {}_{bi}\tilde{t} = {}_{ti}\check{t}^{b} - {}_{ti}\tilde{t}^{b}$$

se $\{\Delta_t\theta_i = \Delta_b\theta_i > 0\}$ posto $\{\|_{ti}\tilde{t}\| > 0 \vee \|_{bi}\tilde{t}\| > 0\}$.

Viene pertanto proposta la metrica

$$\Upsilon_t^{tra} = \frac{\|_{ti}\tilde{t}\| + \|_{bi}\tilde{t}\|}{2}$$

Combinazione delle due metriche Le due metriche proposte vengono combinate in

$$\Upsilon_i = \frac{(\Upsilon_i^{rot})^2}{*\sigma_{rot}^2} + \frac{(\Upsilon_i^{tra})^2}{*\sigma_{tra}^2}$$

con $*\sigma_{rot}^2$ e $*\sigma_{tra}^2$ sono il momento del secondo ordine delle funzioni Guassiane indipendenti di densità di probabilità dell'errore di traslazione e rotazione.

Tale metodo di ottimizzazione non lineare si dimostra più stabile al rumore (errori) in quanto la funzione $\Upsilon = \sum_{i=1}^{n} \Upsilon_i$ tiene in considerazione il modello del rumore, ha inoltre in vantaggio di pesare automaticamente ed ottimamente i pesi della funzione obiettivo, convergendo alla soluzione ottima.

2.2.5 Heller, Havlena and Pajdla (2015), branch and bound

Tutte le tecniche precedentemente proposte, seppure con diversi approcci risolutivi o con diverse formulazioni, hanno tutte vari aspetti in comune, sia dal lato puramente matematico, sia dalle premesse necessarie alla risoluzione del problema.

Uno dei principali aspetti in comune consiste nell'utilizzo di un oggetto di calibrazione, solitamente una scacchiera o una griglia di punti, utilizzata per ottenere la posizione della camera rispetto all'oggetto di calibrazione. La maggior parte delle formulazioni precedenti infatti si basa sul presupposto che si abbia, all'interno della scena inquadrata, un oggetto di calibrazione geometricamente conosciuto, avendo perciò la possibilità di utilizzare tecniche di calibrazione per ottenere, con notevole precisione, la trasformazione fra oggetto e camera stessa. Questa conoscenza, unita alla conoscenza della posizione del tool al quale la camera

è fissato, permette di utilizzare qualsivoglia approccio matematico allo scopo di stimare la trasformazione mano-robot.

Tale presupposto viene messo in discussione da Heller, Havlena e Pajdla i quali si pongono il problema di come riuscire ad ottenere un risultato senza tuttavia avere a disposizione un oggetto di calibrazione noto, i.e. la scacchiera. Si pensi all'eventualità di dover calibrare un sistema braccio robotico - camera in ambienti non controllati, come ad esempio un robot dotato di braccio mobile simile a quelli utilizzati nelle missioni spaziale extraplanetarie o comunemente utilizzati in artificieria, o più semplicemente a casi controllati in cui ciononostante l'inserimento di una scacchiera potrebbe risultare problematico. Si rimuove inoltre il problema dell'accuratezza dell'oggetto, che per essere affidabile deve essere ben illuminato e realizzato in materiale non deformabile.

Per risolvere questi problemi viene proposta una tecnica radicalmente diversa da quelle mostrate finora, in grado di rimuovere completamente il presupposto della calibrazione della camera e rimuovendo inoltre la necessità di un oggetto di calibrazione conosciuto a priori, utilizzando qualsiasi punto riconoscibile nella scena da diverse angolazioni e sfruttando la tecnica del branch and bound per cercare la soluzione nello spazio delle rotazioni presentata da Hartley e Kahl[16] in combinazione con le nozioni di geometria epipolare, garantendo all'algoritmo realizzato di convergere alla soluzione ottima.

Formulazione del problema

La formulazione del problema parte dalla ormai nota formulazione classica del problema $A_iX = XB_i$, scomposta in

$$\begin{aligned} R_{A_i} &= R_X R_{B_i} R_X^T \\ t_{A_i} &= R_X((R_{B_i} - I)t'_X + t_{B_i}) \end{aligned}$$

Supponiamo di avere a disposizione m corrispondenze di punti individuati nella scena $u_{ij} \leftrightarrow v_{ij}, j = 1, \ldots, m$ dove $u_{ij}, v_{ij} \in \mathbb{R}^3$ sono versori i quali rappresentano la direzione ai punti dalla rispettiva posizione della camera. Supponiamo inoltre che u_{ij}, v_{ij} corrispondano a punti $Y_{ij} \in \mathbb{R}^3$ di fronte alla camera.

Si prenda ora una proprietà elementare della geometria della stereo visione, chiamata epipolarità, per la quale i vettori u_{ij}, v_{ij} formano una corrispondenza con una trasformazione A_i se t_{A_i} è coplanare ai due vettori: in altre parole vale la proprietà

$$t_{A_i}^T(v_{ij} \times (R_{A_i} u_{ij})) = 0$$

la quale può essere riscritta esplicitando l'angolo come

$$e_{ij} = \angle([v_{ij}]_\times R_{A_i} u_{ij}, t_{A_i}) - \frac{\pi}{2} = 0$$

con $[.]_\times$ matrice 3×3 antisimmetrica tale che $\forall u, v : [u]_\times v = u \times v$.

In presenza di rumore, ovviamente, tale vincolo non viene soddisfatto, e e_{ij} non sarà propriamente zero ed il valore di $|e_{ij}|$ rappresenterà la deviazione fra la traslazione stimata t_{A_i} ed il piano epipolare definito dalla corrispondenza $u_{ij} \leftrightarrow v_{ij}$, portando alla definizione di un vincolo ϵ-epipolare: la traslazione t_{A_i} e la corrispondenza $u_{ij} \leftrightarrow v_{ij}$ soddisfano il vincolo ϵ-epipolare se $\epsilon > 0$ e $|e_{ij}| \leq \epsilon$.

Questo vincolo può essere espresso in maniera equivalente come

$$\frac{\pi}{2} - \epsilon \leq \angle([v_{ij}]_\times R_{A_i} u_{ij}, t_{A_i}) \leq \frac{\pi}{2} + \epsilon$$

che può essere geometricamente descritto come il vincolo per cui t_{A_i} deve risieder al di fuori del doppio cono composto dall'asse $[v_{ij}]_\times R_{A_i} u_{ij}$ ed apertura $\pi - 2\epsilon$, come mostrato in figura 2.8.

Riscrivendo la disuguaglianza come

$$\frac{\pi}{2} - \epsilon \leq \angle([v_{ij}]_\times R_{A_i} u_{ij}, t_{A_i}) \Leftrightarrow \frac{\pi}{2} + \epsilon \geq \angle(-[v_{ij}]_\times R_{A_i} u_{ij}, t_{A_i})$$

si può formulare il problema di Hand Eye calibration come la minimizzazione di questo vincolo ϵ-epipolare, ovvero come minimizzazione della norma L_∞ del vettore dei residui $e = (|e_{11}|, \ldots, |e_{nm}|)$.

$$(\check{R}_X, \check{t}'_X) = \arg \min_{R_X, t'_X} \max_{i,j} |e_{i,j}| = \arg \min_{R_X, T'_X} \|e\|_\infty$$

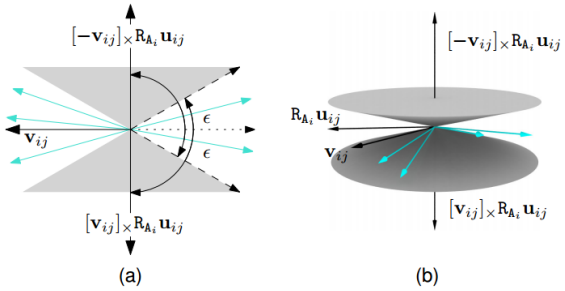


Figura 2.8: Interpretazione geometrica del vincolo ϵ-epipolare imposto dalla corrispondenza $u_{ij} \leftrightarrow v_{ij}$ sulla posizione di t_{A_i}. In azzurro le possibili soluzioni per t_{A_i}, in figura (a) in 2D, in figura (b) in 3D. *Courtesy of Heller, Havlena and Pajdla.*

Si può esprimere il residuo ottimo come $\epsilon_{\min} = \|e(\check{R}_X, \check{t}'_X)\|_\infty$. Ottenuta la soluzione per la parte rotatoria della trasformazione la traslazione si calcola semplicemente come $\check{t}_X = -\check{R}_X \check{t}'_X$ analogamente a quanto succedeva nella formulazione classica del problema.

L'idea, a questo punto, è quella di risolvere il problema di minimizzazione utilizzando la tecnica dell'ottimizzazione tramite Branch and Bound (BnB) per cercare in tutto lo spazio delle rotazioni.

Vengono rappresentate, analogamente a quanto succede con diversi altri algoritmi, le rotazioni tramite la parametrizzazione angolo-asse, sfruttando la formula di Rodrigues[5] per convertire agevolmente rotazioni fra la loro rappresentazione matriciale e quella scelta. Il vantaggio di questa rappresentazione consiste nel poter esprimere con semplicità una metrica per le rotazioni, definendo la distanza $d\angle(R_1, R_2)$ come l'angolo θ della rotazione $R_1^T R_2$ i.e. $[\alpha]_\times = \log(R_1^T R_2)$ con $0 \leq \theta = \|\alpha\| \leq \pi$.

A partire da queste nozioni è possibile definire un semplice algoritmo di BnB in grado di risolvere il problema di minimizzazione presentato come segue:

[5]Prendendo $\alpha \in \mathbb{B}_\pi = \{\beta : \beta \in \mathbb{R}^3 \wedge \|\beta\| \leq \pi\}$, α rappresenta una rotazione attorno all'asse $\alpha/\|\alpha\|$ di un angolo $\|\alpha\|$, e può essere rappresentata in forma matriciale $R \in SO(3)$ tramite la formula di Rodrigues $R = \exp[\alpha]_\times$. La trasformazione inversa per passare da forma matriciale a forma vettoriale è $[\alpha]_\times = \log R$.

1. Ottenere una prima stima di $\epsilon_{\min}$.
2. Dividere lo spazio delle rotazioni in blocchi cubici $D_\sigma^s, s = 1, \ldots, 8$ di larghezza 2σ e ripetere i passi successivi:
   - Per ogni blocco D_σ^s provare che esiste una soluzione $R \in D_\sigma^s$ per cui l'errore residuo della soluzione è minore a $\epsilon_{\min}$. Tale test viene chiamato *feasibility test*.
   - Se non esiste una soluzione tale per cui si ottenga un errore residuo minore di $\epsilon_{\min}$, scartare il blocco.
   - Altrimenti, valutare l'errore residuo ϵ ed aggiornare il valore $\epsilon_{\min} \leftarrow \epsilon$, dopodiché dividere il blocco D_σ^s in otto sottoblocchi e ripetere la procedura.

L'algoritmo termina quando la grandezza del blocco σ diventa sufficientemente piccola, superando una soglia predeterminata.

Da notare che, nonostante il problema di minimizzazione inizialmente posto ha 6 gradi di libertà, l'algoritmo BnB cerca solo nello spazio tridimensionale delle rotazioni, rendendo il *feasibility test* un problema di ottimizzazione lineare. La soluzione fornisce inoltre t'_X per il calcolo dell'errore residuo ϵ, non avendo perciò la necessità di cercare una soluzione nello spazio delle traslazioni

Feasibility test

A questo punto la difficoltà maggiore dell'algoritmo risiede nel calcolo del test di fattibilità, dove scartare o tenere un blocco D_σ.

Una prima formulazione può essere:

Problema 1

$$\begin{gathered} \textit{Dato } D_\sigma, \epsilon_{\min} \\ \textit{esiste } R_X \in D_\sigma, t'_X \\ \textit{tale che } \angle(\pm[v_{ij}]_\times R_{A_i} u_{ij}, t_{A_i}) \leq \frac{\pi}{2} + \epsilon_{\min} \\ \textit{per } i = 1, \ldots, n, j = 1, \ldots, m? \end{gathered}$$

Che è tuttavia un problema non convesso, e quindi di difficile risoluzione. Il problema viene perciò rilassato ad un problema più semplice, riducendo il numero di variabili. Per ottenere ciò viene fissata la Rotazione ponendo $\bar{R}_X$ la rotazione rappresentata dal centro del cubo D_σ e ponendo $\beta_i \in \mathbb{B}_\pi$ tale che $\exp[\beta_i]_\times = R_{B_i}$, ottenendo il problema:

Problema 2

$$\begin{aligned} &\textit{Dato } D_\sigma, \epsilon_{\min}, \bar{R}_X \\ &\textit{esiste } t'_X \\ &\textit{tale che } \angle(\pm[v_{ij}]_\times \bar{R}_{A_i} u_{ij}, \bar{t}_{A_i}) \leq \frac{\pi}{2} + \epsilon_{\min} + \gamma_{ij} \\ &\quad \textit{per } i = 1, \ldots, n, j = 1, \ldots, m \\ &\textit{tale che } \angle(\pm v_{ij}, \bar{R}_A u_{ij}) > 2\|\beta_i\| \sin(\sqrt{3}\sigma/2)? \end{aligned}$$

Il quale, seppure non convesso, possiede alcune proprietà:

- Se il problema 1 è ammissibile, lo è anche il problema 2.
- Se il problema 1 non è ammissibile, D_σ è divisibile in sufficienti sotto blocchi $D^s_{\sigma'}$ tali che il problema 2 sia inammissibile per ogni sotto blocco $D^s_{\sigma'}$.

Inoltre si può notare un vincolo aggiuntivo sull'angolo $\angle(\pm v_{ij}, \bar{R}_A u_{ij})$ ed un inserimento del bound γ_{ij}.

La dimostrazione di tali proprietà è molto lunga e tecnica, e viene pertanto deferita alla fonte della pubblicazione [19].

L'utilità di quest'ultimo problema risiede nel fatto che è possibile rilassarlo ulteriormente per ottenere un problema convesso, e quindi facilmente risolvibile con tecniche di programmazione lineare.

Nel problema 2 ogni corrispondenza $u_{ij} \leftrightarrow v_{ij}$ impongono un vincolo ϵ-epipolare su $\bar{t}_{A_i}$ determinato dal cono C_{ij} con asse $c_{ij} = [v_{ij}]_\times \bar{R}_{A_i} u_{ij}$ ed apertura $\alpha_{ij} = \pi - 2(\epsilon_{\min} + \gamma_{ij})$. Una diversa corrispondenza $u_{ik} \leftrightarrow v_{ik}$, in maniera del tutto analoga, genera un diverso vincolo ϵ-epipolare determinato dal cono C_{ik}.

Imponendo i due coni C_{ij} e C_{ik} con apertura α_{ij}, α_{ik} sufficientemente larga, allora per $c_{ij} \neq c_{ik}$ i due coni si intersecano in un vettore non nullo, come si può più facilmente vedere in figura 2.9a.

Avendo il vertice in comune, i coni si intersecano al massimo in 4 rette, l^1_{ijk}, l^2_{ijk}, l^3_{ijk} e l^4_{ijk}., le quali formano gli spigoli della piramide P_{ijk} nella quale devono essere compresi tutti i vettori che soddisfano entrambi i vincoli ϵ-epipolari. La piramide ha 4 facce, ognuna delle quali è su un piano, come si vede in figura 2.9b.

Questi piani possono essere rappresentati tramite le loro normali n^1_{ijk}, l^2_{ijk}, n^3_{ijk} e n^4_{ijk} tali che

$$n^1_{ijk} = [l^1_{ijk}]_\times l^2_{ijk}, n^2_{ijk} = [l^2_{ijk}]_\times l^3_{ijk}$$
$$n^3_{ijk} = [l^3_{ijk}]_\times l^4_{ijk}, n^4_{ijk} = [l^4_{ijk}]_\times l^1_{ijk}.$$

Ogni vettore $\bar{t}_{A_i}$ che soddisfa entrambe le condizioni ϵ-epipolari soddisfa conseguentemente anche le condizioni, lineari in t'_X,

$$\bar{t}^T_{A_i} n^1_{ijk} \geq 0, \bar{t}^T_{A_i} n^2_{ijk} \geq 0, \bar{t}^T_{A_i} n^3_{ijk} \geq 0, \bar{t}^T_{A_i} n^4_{ijk} \geq 0.$$

Tuttavia, se le aperture dei coni α_{ij} e α_{ik} sono troppo piccole, i coni C_{ij} e C_{ik} non si intersecano necessariamente in un vettore non nullo ed i vincoli ϵ-epipolari non possono essere sostituiti dai vincoli lineari.

Si può ricavare, aiutati dalla figura 2.9c, in che situazioni si verifichi tale eventualità. Prendendo i coni $C^+_{ij}, C^-_{ij}, C^+_{ik}, C^-_{ik}$ i coni determinati dagli assi $c_{ij}, -c_{ij}, c_{ik}, -c_{ik}$ rispettivamente, i coni C^+_{ij} e C^+_{ik} si intersecano in due rette l^1_{ijk} e $-l^3_{ijk}$ se

$$\angle(c_{ij}, c_{ik}) < \frac{\alpha_{ij} + \alpha_{ik}}{2}. \tag{2.15}$$

In caso di uguaglianza i coni sono tangenti e condividono una sola retta.

In maniera del tutto analoga, le rette l^2_{ijk} e $-l^4_{ijk}$ sono le intersezioni dei coni C^+_{ij} e C^-_{ik} se

$$\angle(c_{ij}, -c_{ik}) < \frac{\alpha_{ij} + \alpha_{ik}}{2}. \tag{2.16}$$

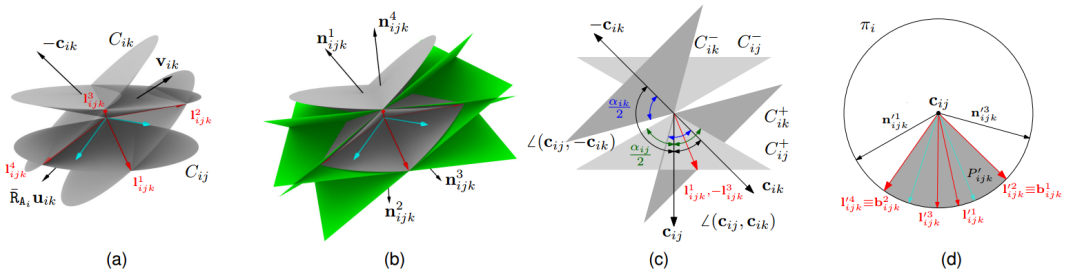


Figura 2.9: (a) Se le aperture $\alpha_i j, \alpha_i k$ sono sufficientemente grandi, i coni C_{ij} e C_{ik} si intersecano in al massimo 4 rette l^1_{ijk}, l^2_{ijk}, l^3_{ijk} e l^4_{ijk}. (b) I vincoli lineari posti dalle corrispondenze $u_{ij} \leftrightarrow v_{ij}$ e $u_{ik} \leftrightarrow v_{ik}$ sono determinati dalle normali n^1_{ijk}, l^2_{ijk}, n^3_{ijk} e n^4_{ijk}. (c) La disuguaglianza (2.15) viene rispettata, pertanto C^+_{ij} e C^+_{ik} si intersecano in l^1_{ijk} e $-l^3_{ijk}$, tuttavia la disuguaglianza (2.16) non viene rispettata, i coni C_{ij} e C_{ik} non formano pertanto la piramide P_{ijk}. (d) Una proiezione 2D della piramide P_{ijk} rappresentata in (b) nel piano π_i. *Courtesy of Heller, Havlena and Pajdla.*

Le due disuguaglianze sono pertanto condizioni necessarie e sufficienti affinché l'intersezione fra due coni C_{ij} e C_{ik} formi la piramide P_{ijk}. In realtà viene generata anche la piramide, simmetrica, P'_{ijk}, la quale tuttavia non è una soluzione ammissibile in quanto non soddisfa la condizione per cui gli oggetti risiedano di fronte alla telecamera (*chierality condition*) descritta in precedenza.

Si può, a questo punto, formulare una nuova versione del problema di ammissibilità:

Problema 3

$$\begin{aligned} &\textit{Dato } D_\sigma, \epsilon_{\min}, \bar{R}_X \\ &\textit{esiste } t'_X \\ &\textit{tale che } \vec{t}^T_{A_i} n^1_{ijk} \geq 0, \vec{t}^T_{A_i} n^2_{ijk} \geq 0, \\ &\qquad \vec{t}^T_{A_i} n^3_{ijk} \geq 0, \vec{t}^T_{A_i} n^4_{ijk} \geq 0, \\ &\qquad \textit{per } i = 1, \ldots, n, j = 1, \ldots, m \\ &\textit{tale che } \angle(\pm v_{ij}, \bar{R}_A u_{ij}) > 2\|\beta_i\| \sin(\sqrt{3}\sigma/2) \\ &\textit{e valgano le disuguaglianze (2.15) e (2.16)?} \end{aligned}$$

Il problema 3 è lineare, ha tuttavia il difetto di portare ad una soluzione troppo vasta dato un problema con un numero significativo di corrispondenze. É possibile, però, diminuire la complessità del problema selezionando accuratamente un sottoinsieme di corrispondenze, e quindi di vincoli lineari, basandosi sulla propria posizione relativa, rendendo più veloce la ricerca della soluzione.

Si considerino l'i-esimo movimento di un blocco cubico $D_\sigma \subset \mathbb{B}_\pi$. Sia C_i la lista di indici le cui corrispondenze soddisfino il vincolo

$$\angle(\pm v_{ij}, \bar{R}_A u_{ij}) > 2\|\beta_i\| \sin(\sqrt{3}\sigma/2)$$

e si consideri la corrispondenza $u_{ij} \leftrightarrow v_{ij}, j \in C_i$ tale da avere la massima ampiezza del cono α_{ij}, chiamando questa corrispondenza condizione base.

Si rimuovano ora gli indici k da C_i per cui le corrispondenze $u_{ik} \leftrightarrow v_{ik}$ non intersechino la condizione base, ovvero quegli indici per cui non valgano le disuguaglianze (2.15) e (2.16).

Si riordino, infine, gli indici in C_i secondo la distanza fra gli assi dei coni $|c^T_{ij} c_{ik}|, k \in C_i$ in ordine discendente.

L'idea è quella di ordinare le corrispondenze in C_i in modo da avere per prime le corrispondenze i cui assi sono più vicini alla perpendicolarità con la condizione base, generando conseguentemente una piramide P_{ijk} più stretta e quindi generando vincoli più restrittivi sulla soluzione ammissibile.

Inoltre è ora possibile prendere in considerazione solamente i primi valori di C_i scartando le soluzioni poco significative, ovvero quelle con un

piccolo $|c_{ij}^T c_{ik}|$, in quanto questi vincoli generano piramidi troppo larghe, e di conseguenza vincoli poco significativi sulla soluzione ammissibile.

Un secondo vantaggio dell'intersecazione fra vincolo base e tutte le restanti consiste nella possibilità di proiettare la piramide P_{ijk} sul piano π_i definito dai vettori $\bar{R}_{A_i} u_{ij}$ e v_{ij}, e di costruire di conseguenza un test semplificato per t_{A_i} basato sulla proiezione 2D.

Si prendano le proiezioni dei lati della piramide sul piano π_i, la zona ammissibile sarà compresa fra due vettori b_{ijk}^1 e b_{ijk}^2, i quali coincideranno con le proiezioni di due dei quattro lati della piramide, come si può vedere in figura 2.9. Per trovare quali dei quattro lati corrispondono ai lati del triangolo 2D proiettato occorre proiettare sul piano anche $n_{ijk}^{\prime 1}$ e $n_{ijk}^{\prime 3}$. Essendo n_{ijk}^1 definito dai lati l_{ijk}^1 e l_{ijk}^2, la proiezione dei due lati che forma l'angolo più ampio con $n_{ijk}^{\prime 1}$ sarà b_{ijk}^1. In maniera analoga si può trovare b_{ijk}^2 sfruttando la faccia della piramide n_{ijk}^3.

Non è possibile utilizzare le facce n_{ijk}^2 e n_{ijk}^4 in quanto le loro proiezioni in π_i sono ugualmente distanti dai lati che le formano per come è stata costruita la proiezione π_i.

Risulta a questo punto immediato notare che l'insieme delle proiezioni P'_{ijk} forma una serie di vincoli sulla proiezione di t'_{A_i}, da cui segue che l'intersezione di tutte le $P'_{ijk} \in C_i$ non può essere vuota per il blocco D_σ perché il problema sia ammissibile.

Purtroppo tale condizione non è sufficiente per garantire l'ammissibilità del problema (i.e. esistono casi in cui l'intersezione fra due piramidi non sia vuota ma il problema sia comunque inammissibile), rende tuttavia possibile decidere l'inammissibilità del problema in maniera molto semplice ed efficace, fornendo un veloce pre-test effettuabile in grado di dare un rapido risultato di inammissibilità in *ca.* il 30% dei casi.

Algoritmo finale proposto

Alla luce di tutto ciò appena descritto è possibile, infine, proporre un algoritmo risolutivo al problema di Hand Eye calibration.

Riguardo al test di ammissibilità, per ogni blocco cubico D_σ si risolve il problema 3. A causa dei vincoli tuttavia non tutte le corrispondenze generano vincoli lineari, rendendo difficile, se non impossibile, la decidi-

bilità di ammissibilità del problema, il quale viene definito ammissibile di default. Più piccoli i blocchi diventano, tuttavia, e più è semplice deciderne l'ammissibilità.

Inoltre, essendo un problema di ammissibilità, un risolutore LP non restituirà genericamente la soluzione ottima di minimizzazione per la funzione obiettivo non lineare $\|e\|_\infty$, bensì una qualsiasi soluzione ammissibile.

Per velocizzare la convergenza dell'algoritmo si utilizza una soluzione ammissibile al problema 3 come stima iniziale del seguente problema non lineare:

$$\begin{gathered} \textit{Dato } \bar{R}_X, \textit{ la stima iniziale } t'_X \\ \textit{minimizza } \|e\|_2^2 = \sum_{i,j} \left(\angle([v_{ij}]_\times \bar{R}_{A_i} u_{ij}, \bar{t}_{A_i}) - \frac{\pi}{2} \right)^2 \\ \textit{per } i = 1, \ldots, j, k = 1, \ldots, m. \end{gathered}$$

Un ulteriore incremento prestazionale è reso possibile dall'alta parallelizzazione dell'algoritmo proposto, in quanto la divisione a blocchi ed il calcolo dell'ammissibilità (*feasibility test*) è parallelizzabile, portando un incremento quasi lineare delle prestazioni rispetto al numero di core utilizzati, come mostrato in figura 2.10 su test effettuati su dati simulati dagli autori.

Risulta a questo punto possibile implementare l'algoritmo linearmente con quanto visto nelle sezioni precedenti.

I risultati ottenuti vengono paragonati, su due diversi sistemi reali mostrati in figura 2.11, alle formulazioni precedentemente proposte in letteratura.

I risultati, mostrati in figura 2.12, rivelano una notevole precisione dell'algoritmo proposto (H12) al pari dei migliori algoritmi presentati in precedenza. Interessentante notare dai test da loro effettuati che il migliore algoritmo provato, nel loro sistema, risulta essere la formulazione a quaternioni duali di Daniilidis (D98), nonostante diversi algoritmi siano stati proposti in seguito.

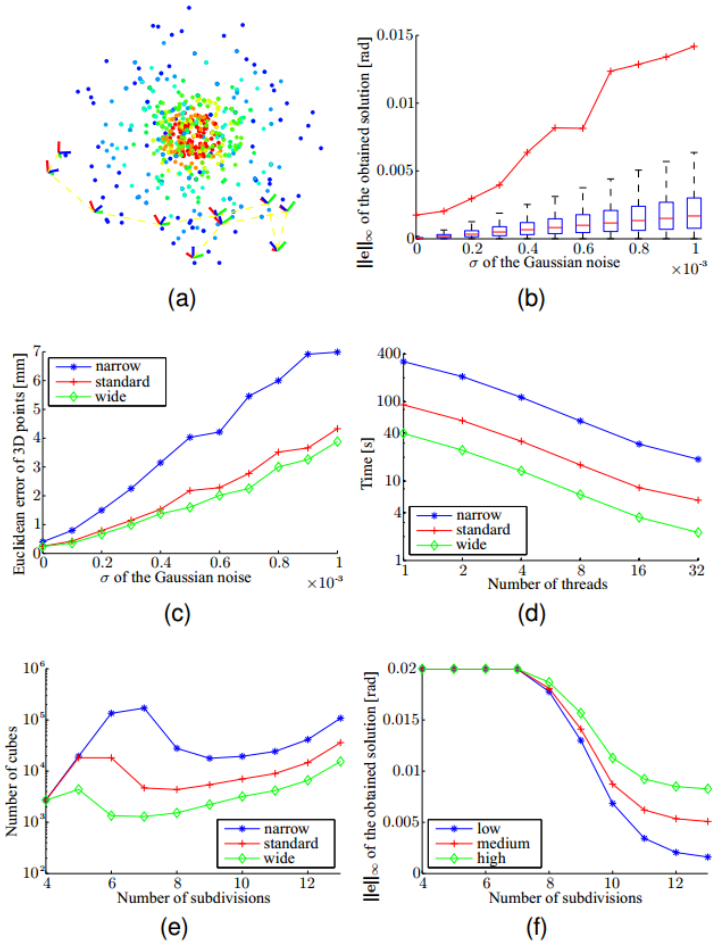


Figura 2.10: Risultati ottenuti dall'algoritmo su dati simulati. (a) mostra una ricostruzione delle diverse posizioni della camera, dove diversi colori indicano diverse FOV (Field of view) utilizzate per la creazione del dataset. (b) mostra il massimo errore residuo per diversi valori di σ (linea rossa) e la distribuzione degli errori misurati rispetto alle diverse corrispondenze (box). (c) rappresenta la distanza media Euclidea fra il punto X ed il punto X' stimato a diversi valori di rumore σ. (d) mostra il decremento, quasi lineare, del tempo computazionale richiesto a diversi livelli di parallelizzazione dell'algoritmo. (e) mostra il numero medio di cubi rimanenti da calcolare all'aumentare del numero di suddivisioni. (f) infine mostra l'errore residuo medio all'inizio di ogni fase di suddivisione. *Courtesy of Heller, Havlena and Pajdla.*

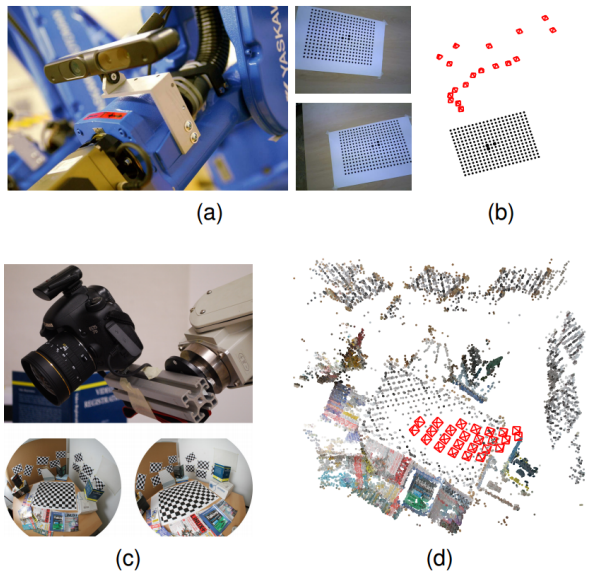


Figura 2.11: (a) e (b) mostrano un close-up del sistema di visione di un robot Motoman MA1400 e alcune immagini di esempio prese dalla sequenza, oltre ad una ricostruzione 3d delle pose utilizzate. (c) e (d) mostrano un diverso robot MELFA-RV-6S al quale è stata collegata una reflex Canon EOS 7D con lente sigma 8mm, alcune immagini di esempio dalla sequenza ed una ricostruzione delle pose in 3d. *Courtesy of Heller, Havlena and Pajdla.*

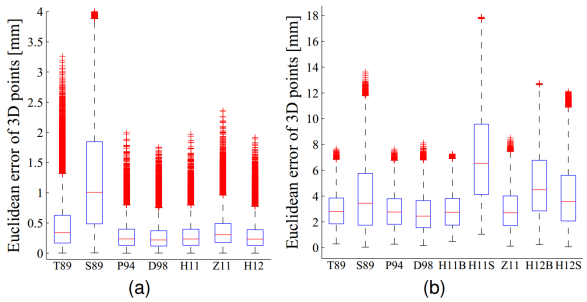


Figura 2.12: Una panoramica sui risultati ottenuti sul Motoman (a) e sul Mitsubishi (b) da diversi algoritmi testati: Tsai (T89)[25][26], Shiu e Ahmad (S89) [23], Park e Martin (P94) [22], Daniliidis (D98)[13], una formulazione precedente proposta dagli stessi autori (H11) [17], Zhao (Z11)[27], ed infine il metodo BnB proposto (H12). *Courtesy of Heller, Havlena and Pajdla.*

2.3 Soluzione adottata da Specialvideo

La calibrazione Hand Eye ha un duplice scopo: sia quello di utilizzare attivamente la telecamera allo scopo di rendere più precise operazioni del robot nel suo campo di lavoro, o addirittura in alcuni casi rendere possibili operazioni che non sarebbe effettuabili senza un sistema di visione. Essere dotati di una oppure per ottenere con precisione una trasformazione che consenta al robot di esprimere coordinate del mondo reale nel proprio sistema di riferimento.

Nella corso della pluriennale esperienza di Specialvideo diverse soluzioni si sono susseguite nel tempo, mano a mano che i sistemi di visione sono diventati sempre più comuni nel campo della robotica. Inizialmente, operando principalmente con robot non dotati di telecamere, la calibrazione doveva necessariamente avvenire con tecniche differenti da quanto visto nelle sezioni precedenti, in quanto non avendo a disposizione una camera era necessario trovare un differente approccio per poter calcolare la calibrazione del robot, la quale consisteva semplicemente nella trasformazione fra mondo reale e sistema di riferimento del robot.

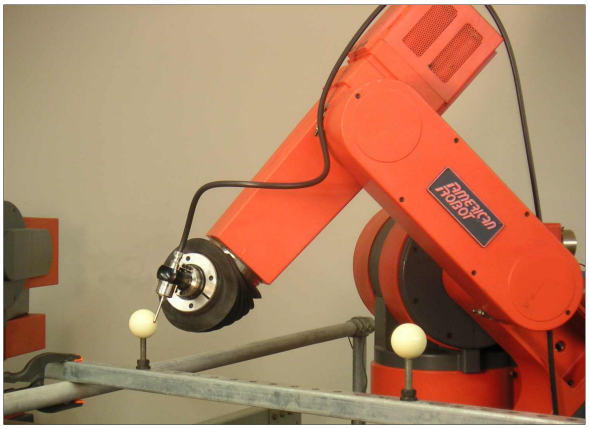


Figura 2.13: Un esempio di tool il cui compito è quello di muoversi attorno a delle sfere per imparare il sistema di riferimento del piano di lavoro.

Il metodo inizialmente utilizzato per calibrare un robot, non avendo a disposizione una camera, consiste nell'inserire nel dominio del robot alcuni mark fissi, facenti parte di un pattern conosciuto, e guidare manualmente il tool[6] del robot a toccare ciascuna di queste punte, assumendo diverse posizioni mantenendo il contatto con il mark fisso, in maniera simile a quanto raffigurato in figura 2.13.

Questa operazione permette, a partire dalle pose del robot conosciute, di stimare la posizione dei mark nel proprio sistema di riferimento, con una precisione che dipende dall'accuratezza dell'operatore incaricato di guidare manualmente il braccio robotico. A partire dalle posizioni registrate dall'operatore si ha la possibilità di creare un nuovo sistema di riferimento virtuale, conosciuto al robot, ottenendo la trasformazione robot - mondo corrispondente al piano di lavoro.

Una calibrazione di questo tipo, ovviamente, porta notevoli svantaggi:

- La calibrazione del robot richiede un grande ammontare di tempo, è infatti necessario pilotare manualmente il robot per toccare con precisione i mark di riferimento. Una soluzione del genere porta

[6]Per tool si intende l'ultimo punto del robot alla fine della catena di trasformazioni dei bracci dello stesso, avendo perciò la possibilità di conoscere, con un'accuratezza data dal costruttore del robot stesso, la sua posizione ed orientazione.

necessariamente un impiego di tempo notevole, rispetto al tempo necessario ad acquisire immagini ritraendo un oggetto sul piano di lavoro.

- Specializzazione dell'operatore incaricato di effettuare la calibrazione. Il risultato finale dipende fortemente dalla capacità dell'operatore di toccare con precisione i mark fissati al piano di lavoro per ottenere una calibrazione efficace.

- In alcuni casi, dove ad esempio il robot ha scarsa libertà di movimento o è montato in postazioni anguste, non è banale il problema del fissare le punte metalliche sulle quali effettuare la calibrazione.

- La calibrazione dipende fortemente dalla concezione che base del robot e piano di lavoro rimangano fissi nel tempo. Nel comune operare del robot possono esserci spostamenti in uno dei due piani dovuti al lavoro stesso, che possono portare alla necessità di una ricalibrazione del robot, rendendo necessario l'intervento di un operatore specializzato. Più la calibrazione è precisa, e maggiormente le perturbazioni del sistema robot-mondo dovute a contatti accidentali o all'operare del sistema stesso sono significative, arrivando nei casi più importanti a non essere tollerabili richiedendo una ricalibrazione del sistema, richiedendo quindi un blocco del sistema e l'intervento dell'operatore per ripetere la procedura.

- In alcuni casi i mark potrebbero essere d'intralcio nel piano di lavoro, rendendo problematico il fissaggio e successiva rimozione degli stessi una volta completata la calibrazione, qualora si decida di usare mark non disegnati.

- Scarsa precisione, in quanto la calibrazione dipende interamente dall'accuratezza con cui l'operatore tocca i mark con il tool del robot. Inoltre in determinati ambienti lavorativi risulta essere un problema non banale quello del vedere la precisione con cui il mark viene toccato per limiti fisici dell'operatore (si pensi a robot molto grandi nella cui zona di lavoro può essere vietato l'accesso con il robot in funzione per motivi di sicurezza).

Questa serie di problemi, insieme all'introduzione in ambiente lavorativo di robot dotati di camere a bordo, rende necessario l'utilizzo di tecniche più intelligenti di calibrazioni, in grado principalmente di

1. rimuovere la necessità di un operatore specializzato per effettuare la calibrazione del robot,

2. rendere più veloce e più precisa la procedura di calibrazione,

3. rendere la procedura contactless, rimuovendo la necessità di toccare fisicamente dei punti fissati al pianto di lavoro utilizzando al loro posto dei pattern di calibrazione i quali possono essere integrati nel piano di lavoro,

4. utilizzare efficacemente la camera di cui è dotato il robot al di fuori della calibrazione, dandogli la possibilità di effettuare operazioni altrimenti impossibili (problema del grasping e dell'ordinamento di oggetti).

Prima soluzione: camera fissa

La prima soluzione dotata di un sistema di visione venne inserita con l'inizio della diffusione di telecamere in ambito robotico, e consiste nell'utilizzo di una camera fissa in grado di inquadrare il piano di lavoro del robot.

Il primo processo di calibrazione Hand Eye si basa sulla tecnologia precedentemente utilizzata per risolvere il problema della calibrazione del robot, mantenendone di conseguenza alcune debolezze.

La calibrazione è composta principalemente da tre step consecutivi:

1. Il primo passo consiste nel conoscere, con precisione, la trasformazione del tool del robot, solitamente una punta metallica fissata alla flangia o ad un utensile utilizzato dal robot stesso. Tale calibrazione viene effettuata toccando fisicamente un oggetto fissata al piano di lavoro, solitamente un'altra punta, con diverse orientazioni. Le diverse posizioni assunte dal robot, le quali mantengono tuttavia il tool nella stessa posizione(solo con orientazioni diverse), permettono di calcolare, con una precisione dipendente dalla precisione

stessa con cui l'operatore assume le posizioni, la trasformazione del tool rispetto al robot.

2. Il passo successivo consiste nel toccare, con il tool di cui a questo punto è ben conosciuta la posizione, un pattern noto sul piano di lavoro facente parte di un tool di calibrazione, il quale può essere una griglia di punti o una scacchiera. Questo permette di conoscere, con una precisione data dalla precisione con la quale l'operatore tocca il pattern sul piano di lavoro, la posizione del tool di calibrazione definita in coordinate del sistema di riferimento del robot.

3. Infine, utilizzando tecniche standard di visione artificiale, è possibile calcolare, a partire dal tool di calibrazione, la posizione della camera, grazie a tecniche standard di calibrazione ed estrazione degli estrinseci.

Conosciute le varie trasformazioni intermedie è immediato ottenere la catena di trasformazioni

$$robot \rightarrow tool \rightarrow tooldicalibrazione \rightarrow camera$$

e di conseguenza anche le trasformazioni relative tool $\rightarrow$ camera.

Come si può notare la soluzione proposta mantiene tuttavia tutti i problemi già notati per la semplice calibrazione del robot. La precisione della calibrazione così effettuata dipende infatti dalla precisione con cui l'operatore tocca correttamente i punti del tool di calibrazione con il tool del robot, confidando nella precisione di queste operazioni per ottenere una calibrazione accurata.

Obiettivo primario per il miglioramento della qualità della calibrazione consiste quindi nella rimozione dal processo di calibrazione dell'intervento umano quanto più possibile, rendendo il processo di calibrazione il più possibile semplice ed automatizzato, migliorandone di conseguenza la precisione e rimuovendo la necessità dell'intervento di un operatore specializzato.

Seconda soluzione: mark fisso sul robot e camera fissa

La seconda soluzione prevede la rimozione della necessità, per l'operatore, di comandare manualmente il robot per toccare il pattern di calibrazione con il tool per definirne la posizione, necessaria per il calcolo della calibrazione Hand Eye.

La soluzione pensata consiste in tre step fondamentali, alcuni dei quali simili alle strategie precedentemente adottate.

1. Il primo step rimane invariato rispetto a quanto visto in precedenza, viene tuttavia modificato il tool collegato al robot. Al posto di una punta metallica viene utilizzata una placca di metallo contenente, alla sua estremità, un marker riconoscibile dalla camera. La posizione del marker viene calcolata effettuando una procedura di calibrazione simile a quanto effettuato in precedenza facendo toccare le due punte metalliche, con la differenza che in questa nuova soluzione la punta metallica fissa viene messa a contatto con il mark disegnato sulla barra. Tale operazione richiede la massima precisione da parte dell'operatore, così come accadeva in precedenza.

2. Una volta ottenuta la trasformazione fra mark e robot, viene messo il mark riconoscibile dalla telecamera in una posizione visibile alla camera, fissa, in modo che la camera sia in grado di riconoscerlo, registrando la posizione del robot.

   Viene successivamente effettuato un pattern noto di traslazioni, salvando per ogni posizione del pattern la posizione del tool del robot (il mark riconoscibile) e l'immagine presa dalla camera. Effettuando un pattern di traslazioni tridimensionale è possibile generare un cubo virtuale nello spazio del robot, di dimensioni note, ed in maniera analoga nel mondo della camera, prendendo i vari punti riconosciuti dalle immagini.

3. Partendo dai dati raccolti nello step precedente, è possibile, dal cubo virtuale composto dai mark detectati dalla camera, ricavarne la posizione relativa al cubo reale, e rispetto a robot stesso di conseguenza, tramite l'utilizzo di librerie esterne.

Quest'ultima, che al momento è la soluzione utilizzata, porta miglioramenti rispetto alla precedente soluzione adottata, ne condivide tuttavia anche alcune debolezze:

- Prima di tutto si fa affidamento, come nella prima soluzione adottata, alla capacità dell'operatore nel toccare correttamente un punto fisso con il mark del robot. L'intera calibrazione successiva si basa su questo errore, che quindi si propaga nelle misure successive. Un vantaggio, rispetto alla soluzione precedente, risiede tuttavia nel fatto che una volta che il tool è correttamente calibrato può essere riutilizzato per la calibrazione saltando il primo step, in quanto si suppone che il tool non subisca alterazioni fisiche tali da modificarne fisicamente la conformazione. Il problema dell'avere un tool esterno fisico, da attaccare al robot nel momento della calibrazione, può tuttavia comportare altri problemi: in primis lo smarrimento, cosa che purtroppo può accadere in ambiente industriale e che richiederebbe di conseguenza l'intervento dell'operatore per effettuare la calibrazione su un nuovo (e quindi necessariamente differente) tool di sviluppo, oppure un altro errore che può accadere risiede nel non corretto fissaggio del tool al robot, il quale causerebbe di conseguenza una errata calibrazione.

- La tecnica matematica dietro alla calibrazione si basa su punti generati da un pattern eseguito dal robot, al quale si chiede di effettuare con precisione una serie di traslazioni. Nel caso in cui il robot non sia preciso ma bensì commetta un errore, tale errore si propagherebbe nella lettura del mark dalla camera, generando un set sbagliato di dati e portando conseguentemente un errore dipendente dalla capacità del robot nell'essere preciso nelle operazioni di traslazione.

Soluzione alternativa con mark fisso e camera mobile

Nel caso in cui il mark sia fisso sul piano di lavoro, ad esempio nel caso sia disegnato sullo stesso, e la camera sia a bordo del robot, si può comunque utilizzare una tecnica quasi identica alla precedente.

- Il primo step consiste nella calibrazione di un tool del robot, solitamente una punta metallica, sul mark fisso disegnato sul piano di lavoro, in modo da conoscere la posizione del mark, fisso, rispetto al sistema di riferimento del robot.

- Il passo successivo consiste nell'allontanarsi dal tool e, muovendo il robot, porre ad una determinata distanza il mark fisso rispetto alla camera. A questo punto si effettua una serie di traslazioni in maniera identica a quanto accadeva con il tool sul robot e la camera fissa, generando un cubo virtuale tramite le singole immagini prese in ogni punto, facendo attenzione che la camera riprenda sempre il mark.

- Come ultimo passo, in maniera del tutto analoga a quanto accade nel caso speculare, i dati generati vengono utilizzati per calcolare la posizione della camera rispetto al mark, la cui posizione è nota al sistema, calcolando di conseguenza la calibrazione tool $\rightarrow$ camera.

Quest'ultima soluzione, speculare alla precedente, porta con se gli stessi errori già presentati nella tecnica precedente.

Capitolo 3

Design della soluzione

Viene mostrata in questo capitolo la proposta di progetto per la risoluzione del problema, alla luce sia dello studio teorico presentato nel capitolo precedente, sia delle precedenti soluzioni implementate in azienda, sia dai vincoli e dalle specifiche hardware, software e di sistema richiesti, descrivendo i ragionamenti che hanno portato alla scelta del design presentato, degli algoritmi usati e delle librerie e dell'IDE di sviluppo scelto a supporto dell'implementazione.

3.1 Specifiche di sistema e vincoli di progettazione

Specifiche del sistema

Le specifiche di sistema richieste sono schematizzabili come segue:

- Creare un sistema in grado di calcolare l'Hand Eye calibration senza la necessità di far toccare al robot, manualmente, oggetti di calibrazione come punte, incroci o oggetti simili.

- Creare una procedura di calibrazione in grado di essere completamente automatica, rimuovendo la necessità di operatori specializzati.

- Il sistema riceverà in input una serie di immagini ritraenti una scacchiera, di dimensioni note, prese da una camera matriciale; inoltre riceverà, in input, le rispettive posizioni assunte dal robot al momento di ogni scatto. Non è pertanto richiesta la creazione della fase di acquisizione dei dati. Viene tuttavia richiesta una descrizione di come tali dati dovranno essere generati, ovvero una descrizione sul percorso da far compiere al robot per ottenere un insieme di dati che garantisca precisione ed efficacia all'algoritmo.
- Creare un sistema modulare, in grado di essere agevolmente integrato con il software aziendale.
- Scrivere il codice sorgente a regola d'arte, secondo le norme di buona scrittura e formattazione, manutenibilità, portabilità e chiarezza, in modo che sia il più possibile comprensibile ed auto documentato.

La principale specifica di sistema fornita è la risoluzione del problema che già ha spinto l'azienda a passare dalla prima alla seconda soluzione da loro utilizzata, ovvero la necessità di togliere, o per lo meno limitare quanto più possibile, l'intervento umano all'interno del processo di calibrazione. Risulta infatti importante rendersi indipendenti dall'intervento umano principalmente per due motivi: sia per migliorare l'accuratezza della calibrazione, non dipendendo più quindi dall'abilità del singolo operatore nel comandare manualmente il robot per assumere determinate pose, sia per ridurre il costo stesso dell'operazione, dato sia dal tempo necessario ad effettuare la calibrazione, sia dal costo intrinseco nel mandare un operatore che conosca la funzionalità dell'algoritmo di calibrazione specializzato per eseguirla. La guida manuale del robot precedentemente utilizzata, nel tentativo di essere il più precisi possibili, porta necessariamente ad un consumo di tempo maggiore rispetto a quanto si avrebbe facendo muovere il robot in posizioni libere, con l'unico vincolo presente che la camera inquadri l'oggetto di calibrazione.

Con un diverso approccio risolutivo, come quelli presentati nel capitolo precedente, l'unico vincolo presente per l'operatore risulterebbe quello di acquisire delle fotografie le quali inquadrino l'intero tool di calibrazione in una serie di posizioni a piacere, rimuovendo inoltre il bisogno di

utilizzare un tool specifico, sia esso una punta per toccare il mark o una piastra con il mark stesso disegnato, rendendo l'operazione più semplice, veloce e precisa, in grado di essere eseguita da qualsiasi operatore senza necessità di specializzazione alcuna, diminuendo conseguentemente il costo dell'operazione stessa. Inoltre utilizzare un oggetto di calibrazione significherebbe rendere maggiormente indipendenti le stazioni fra di loro: nella tecnica utilizzata attualmente occorre generare un cubo virtuale tramite una serie di traslazioni, ed un errore sulle prime si propaga inevitabilmente alle traslazioni successive. Una tecnica che al contrario sfrutta le singole posizioni fa affidamento soltanto sulla capacità del robot di assumere correttamente una posizione data, invece che fare affidamento sulla sua capacità di compiere con precisione una serie di spostamenti.

Infine una tecnica basata sul confronto di immagini prese da diversi stazioni ha il vantaggio di ottenere per ogni immagine l'intero oggetto di calibrazione, che al momento, invece, viene creato virtualmente tramite una serie di spostamenti del robot. Si ha perciò la certezza che l'immagine vista dal sensore sia esattamente quella dell'oggetto reale, e non una immagine influenzata dalla precisione del robot (oltre che della camera) essendo stata generata da una serie di punti presi singolarmente da altrettanti spostamenti, portando implicitamente una migliore precisione dei dati di partenza.

All'algoritmo verranno forniti, da specifiche, i dati di input, dividendo perciò concettualmente il problema dell'acquisizione dei dati, diverso a seconda del tipo e modello di robot, dalla loro elaborazione, il cui output sarà la calibrazione del sistema stesso. Viene inoltre fornita la certezza di avere a disposizione un oggetto di calibrazione tradizionale, ovvero una scacchiera, di caratteristiche geometriche note.

É infine richiesto lo sviluppo di un software il più possibile modulare e portabile, secondo le buone norme di programmazione, creando un sistema che renda il più semplice ed immediato possibile l'inserimento del codice sviluppato nell'applicativo aziendale, che dovrà perciò essere performante, mantenibile e più genericamente scritto ad arte seguendo le convenzioni di buona scrittura del codice, rendendolo quanto più possibile auto documentato e comprensibile.

Vincoli di progettazione

Non si pongono particolari vincoli al sistema, né a livello hardware, né a livello software, oltre a quelli elencati:

- La stima della calibrazione dovrà essere effettuata in tempi ragionevoli.
- Non sono forniti particolari vincoli hardware, se non che la computazione possa essere effettuata su sistemi di comune potenza, in modo da poter essere eseguiti, all'occorrenza, su sistemi di potenza ridotta o embedded.
- Il progetto dovrà essere sviluppato come libreria software, sviluppata in linguaggio `C++/C#` su ambiente Windows, preferibilmente utilizzando l'ide di sviluppo Visual Studio, ed integrabile con il software aziendale.
- Potranno essere utilizzate librerie software esterne a patto che ne sia giustificato l'utilizzo e non comportino netti cali prestazionali al sistema nel suo complesso, o ne gravino la mantenibilità e sicurezza.
- Sarà consentito l'uso di tutte le librerie già utilizzate dal software aziendale.

I vincoli riguardano principalmente la modalità di deployment del prodotto, dovuti alla necessità, da parte dell'azienda, di integrare il sistema con quello già in uso. Unico vincolo posto dalla richiesta, oltre al linguaggio di programmazione da adottarsi, riguarda la velocità di computazione: la calibrazione è intrinsecamente un processo costoso, che richiede tempo per essere effettuata con precisione. Il calcolo della calibrazione non avrà vincoli computazionali stringenti, come invece ad esempio accade nel caso di sviluppo di applicazioni real-time, non dovrà tuttavia essere eccessivamente pesante, in modo da poter, all'occorrenza, essere portato anche su sistemi embedded, dotati di scarse risorse computazionali. Non vengono invece specificati vincoli particolare sul supporto hardware e software, in quanto il progetto verrà sviluppato come semplice

libreria C++, da integrare successivamente nel prodotto software dell'azienda, il quale si occuperà, tramite classica installazione, di interfacciarsi con il sistema operativo sottostante.

3.2 Scelta dell'approccio teorico

In letteratura vari algoritmi sono stati proposti per la risoluzione del problema che non comprendano movimenti precisi del robot effettuati, manualmente, da un operatore, ma bensì si basino esclusivamente sulle immagini prese dalla camera di cui è dotato. La soluzione più moderna fra quelle studiate, proposta da Heller *et al.* (2015) porta con se il notevole vantaggio di avere la completa libertà nella scelta dell'oggetto di calibrazione, potendo infatti utilizzare un qualsiasi insieme di punti all'interno della scena, con l'unico vincolo che questi siano riconoscibili ed individuabili con precisione dalle diverse posizioni assunte dalla camera. Seppur di notevole interesse rimane tuttavia non necessario, in quanto la calibrazione studiata verrà eseguita in ambiente interamente controllato, dove perciò si ipotizza di poter inserire senza alcuna difficoltà un oggetto di caratteristiche geometriche note, sia esso una scacchiera o una griglia di cerchi, dando la possibilità di utilizzare metodi di calibrazione più tradizionali, i quali portano il vantaggio di una maggiore precisione e rapidità computazionale. Uno svantaggio della soluzione citata, inoltre, risiede nella difficile generalizzazione in ambiente industriale: la richiesta di utilizzare una qualsiasi corrispondenza fra due immagini in posizioni diverse richiede che l'operatore controlli che l'oggetto individuato sia corretto; occorre inoltre decidere quali punti generino le corrispondenze, creando marker specifici per il singolo problema nella singola installazione. Non è un caso se, nella pubblicazione dell'algoritmo, viene utilizzata una semplice e classica scacchiera per testare il proprio algoritmo. L'utilizzo di un classico oggetto di calibrazione rende possibile l'utilizzo di un algoritmo con la massima generalizzazione possibile ed il minimo intervento specializzato possibile, in quanto l'unico vincolo che si pone all'operatore è che la scacchiera venga inquadrata correttamente dalla camera fra una posa e l'altra.

Rimane tuttavia una formulazione da tenere in alta considerazione, in quanto rende possibile la calibrazione per tutti i casi in cui l'inserimento di una scacchiera o oggetto similare non sia fisicamente possibile o utilizzabile, a causa ad esempio di una illuminazione non sufficiente rendendo difficoltoso il riconoscimento del pattern utilizzato come oggetto di calibrazione o nel caso di macchine in cui il campo inquadrato dalla camera sia in una posizione difficilmente accessibile.

Avendo la certezza di poter utilizzare un oggetto di calibrazione noto, ovvero una scacchiera, come da specifiche del sistema, la scelta ricade conseguentemente su formulazioni più tradizionali del problema di Hand Eye Calibration: fra queste è stato scelto l'utilizzo della formulazione di Daniliidis sfruttando la notazione dei quaternioni duali. Tale scelta è frutto principalmente della semplicità e rapidità computazionale dell'algoritmo, in quanto la formulazione prevede soltanto una semplice decomposizione per valori singolari, rendendo possibile l'esecuzione del metodo in tempi molto brevi anche su sistemi di scarsa potenza computazionale, come ad esempio sistemi embedded, sempre più comuni in ambito sia domestico che industriale. Viene inoltre semplificato lo sviluppo dal supporto notevole dato da diverse librerie open source, robuste e conosciute come ad esempio OpenCV ed Eigen, in grado di semplificare notevolmente lo sviluppo software per quanto riguarda il lato matematico comprendente quaternioni, numeri duali e decomposizione per valori singolari, oltre ai più banali problemi di calcolo algebrico fra matrici e vettori. Tali librerie, inoltre, nascono con l'idea di essere estremamente portabili e multipiattaforma, supportando il inguaggio `C++`, come richiesto.

L'algoritmo proposto da Daniliidis risulta inoltre essere molto efficace, in grado di garantire errori molto contenuti, oltre ad una notevole rapidità computazionale. La principale fonte di errori si discosta pertanto dalla qualità del metodo proposto, bensì viene influenzata in maniera molto maggiore dalla precisione delle misurazioni effettuate dal robot, per il quale occorre analizzare il data-sheet del costruttore, e dalla qualità del sistema di visione integrato, oltre che dalla possibilità fisica del robot di effettuare il maggior numero possibile di movimenti il più possibile diversi fra loro.

La soluzione proposta inoltre offre una ampia possibilità e modularità

applicativa: è infatti possibile, qualora si voglia, calcolare la calibrazione relativa rispetto a due qualsiasi oggetti rigidamente connessi, a patto che se ne descriva lo spostamento rispetto a diversi sistemi di riferimento. Tale nozione può essere utile per la calibrazione di tool diversi o per diversi tipi di camere (lineari, laser, 3d, etc).

L'algoritmo implementato viene schematizzato come segue

1. Dati n spostamenti di un robot (b, b') ed altrettanti corrispondenti movimenti di una camera (a, a') controllare che le parte scalari della trasformazione siano uguali, dopodiché estrarre la direzione ed il momento degli assi di rotazione (*screw axes*) con i quali costruire la matrice T.

2. Calcolare la decomposizione a valori singolari (SVD) di T e controllare che solamente due valori siano prossimi a zero, essendo T una matrice di rango 6. Prendere i corrispondenti vettori singolari destri $\vec{v_7}$ e $\vec{v_8}$.

3. Calcolare i coefficienti dell'equazione

$$\lambda_1 \vec{u_1}^T \vec{v_1} + \lambda_1 \lambda_2 (\vec{u_1}^T \vec{v_2} + \vec{u_2}^T \vec{v_1}) + \lambda_2^2 \vec{u_2}^T \vec{v_2} = 0$$

   trovando due soluzioni per $s = \lambda_1 / \lambda_2$.

4. Per i due valori di s calcolati, calcolare per ognuno $s^2 \vec{u_1}^T \vec{u_1} + 2s\vec{u_1}^T \vec{u_2} + \vec{u_2}^T \vec{u_2}$ e scegliere il più grande, dal quale calcolare prima λ_2 e poi λ_1.

5. La soluzione è $\lambda_1 \vec{v_7} + \lambda_2 \vec{v_8}$.

3.3 Librerie utilizzate

A seguito della scelta dell'algoritmo, delle specifiche fornite e dei vincoli presenti si decide di utilizzare alcune librerie per semplificare e rendere più agevole, leggibile, performante e mantenibile il codice sviluppato. Nascono infatti, sia dai vincoli e dalle specifiche stesse poste dall'azienda, sia dalla scelta della formulazione matematica da utilizzare per risolvere il

problema di calibrazione, problemi di natura sia implementativa, sia matematica, che richiedono l'utilizzo di librerie esterne sia per semplificare l'implementazione stessa, sia per rendere, come esplicitamente richiesto dai requisiti, più semplice, lineare e comprensibile il codice scritto.

- Il primo problema consiste nel ricavare efficacemente, a partire dalle immagini fornite ritraenti una scacchiera, la posa della camera rispetto alla scacchiera stessa.
- Un altro problema consiste nella necessità di rappresentare il concetto matematico di quaternione e quaternione duale, previsti dall'algoritmo proposto, e di come effettuare calcoli algebrici ed operazioni fra questi.
- Infine è intrinsecamente necessario, all'interno dell'algoritmo, effettuare diverse operazioni matematiche, siano esse operazioni matematiche, di calcolo algebrico fra matrici, di calcolo fra quaternioni o quaternioni duali, o decomposizioni di matrici a valori singolari: è pertanto consigliabile utilizzare delle librerie in grado di semplificare e rendere performanti questo tipo di calcoli.

3.3.1 OpenCV

La prima libreria presentata è OpenCV, in quanto essendo già presente all'interno del software aziendale, non presenta problemi nel caso sia necessario includerla nel progetto. OpenCV (Open source Computer Vision)[7] è, senza ombra di dubbio, la più famosa e supportata libreria a supporto della visione artificiale, sviluppata interamente e gratuitamente dalla comunità scientifica. Distribuita con licenza BSD[1] è pertanto libera nell'utilizzo sia personale, sia commerciale. OpenCV nasce e viene sviluppata per supportare il maggior numero possibile di piattaforme, fornendo interfacce C++, C, Python e Java e supportando Windows, Linux, Max OS, iOS e Android, potendo perciò funzionare sulla quasi totalità di dispositivi odierni. Scritta in `C/C++`, pone grande attenzione all'efficienza computazionale, rendendo possibile lo sviluppo di applicazioni real-time grazie al supporto ad architetture multicore e accelerazione hardware (grazie ad OpenCL).

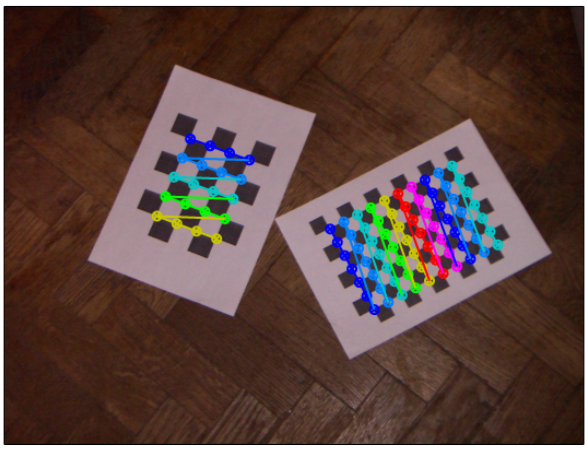


Figura 3.1: Un esempio di applicazione della funzione `findchessboardcorners()` di OpenCV, in grado di individuare automaticamente e con grande precisione una scacchiera all'interno di una immagine.

L'utilizzo di OpenCV risulta essere quasi obbligato: come da specifiche vengono infatti fornite, in input al sistema, delle semplici immagini ritraenti una scacchiera, mentre l'algoritmo risolutivo scelto necessita, in input, le trasformazioni da una posa alla successiva della camera: si pone pertanto il problema di come, a partire dalle immagini, ricavare i dati necessari al calcolo della calibrazione. Per la risoluzione di questo problema occorre necessariamente adottare tecniche di visione artificiale, sia per riconoscere l'oggetto di calibrazione utilizzato, la scacchiera, sia per, a partire da quest'ultima, calcolare la posa della camera. Tale operazione, senza l'utilizzo di una libreria specifica alla risoluzione del problema, richiederebbe l'implementazione di algoritmi avanzati di visione artificiale: sfruttare OpenCV consente di abbattere il gap tecnologico e rendere semplice, intuitiva, funzionale ed efficace la soluzione al problema. OpenCV infatti fornisce nativamente algoritmi in grado di riconoscere, con precisione, la posizione di un oggetto di calibrazione geometricamente noto, in maniera semplice ed efficace, come mostrato in figura 3.1 su una immagine reale o in figura 3.2 su immagini di test artificiali costruite tramite un simulatore. É inoltre possibile, a partire

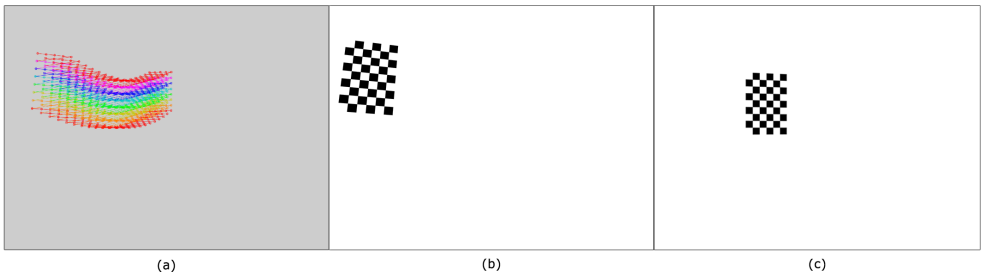


Figura 3.2: Un esempio dell'utilizzo di OpenCV per individuare spostamenti della scacchiera in diverse immagini. In (a) le scacchiere individuate da openCV in una serie di immagini, mentre (b) e (c) sono due immagini artificiali prese dalla serie, costruite tramite un simulatore sviluppato con l'ausilio della libreria Visualization Toolkit(VTK).

dalla immagini della scacchiera, utilizzare algoritmi già pronti in grado di calibrare la camera stessa, calcolarne i parametri intrinseci ed estrinseci, e ricavare successivamente le informazioni relative alla posa stessa della camera relativa alla scacchiera, necessarie per il calcolo della calibrazione Hand Eye successiva.

La scelta di utilizzare OpenCV risulta obbligata non soltanto dal fatto che è la libreria migliore per risolvere questo tipo di problemi, ma anche dal fatto che, essendo già presente ed utilizzata in Specialvideo, non comporterebbe nessun problema nella fase di integrazione della libreria nel software aziendale. Inoltre, in luce al requisito di mantenibilità e comprensibilità, l'utilizzo di una libreria ben conosciuta, documentata e già presente ed utilizzata in azienda renderebbe più semplice la comprensione del codice e la sua estensione da parte di terzi.

3.3.2 Eigen

Un'altra libreria interessante è Eigen[3], una libreria compilata in `C++98` compatibile con qualsiasi compilatore supporti tale linguaggio, come `GCC`, `MinGW`, `MSVC`, `intel C++ compiler` etc. Eigen è inoltre un software free ed open-source, distribuito tramite licenza Mozilla Public License Version 2.0 (MPL2)[6], utilizzabile liberamente per progetti proprietari, sviluppato interamente e gratuitamente dalla comunità scientifica. I suoi vantag-

gi sono la versatilità, supportando matrici di qualsiasi dimensione, dense o sparse, e di qualsiasi tipo numerico, supportando tipi di dati come `std::complex`, interi, o qualsiasi tipo custom di cui si abbia necessità. Supporta inoltre vari tipi di decomposizione di matrici ed un ecosistema di moduli in grado di fornire metodi di alto livello come ad esempio la decomposizione a valori singolari, utilizzata nella soluzione, oltre a varie proprietà geometriche, a risolutori polinomiali, funzioni matriciali, FFT, moduli di ottimizzazione non lineare etc.

Dalla scelta dell'uso dell'algoritmo di Daniilidis per risolvere il problema nasce la necessità di effettuare diversi calcoli fra matrici, quaternioni e quaternioni duali. Eigen nasce appositamente con l'intento di semplificare e rendere performante l'utilizzo di tali nozioni matematiche ed algebriche, fornendo semplici metodi di libreria per tutti i calcoli e procedure matematiche che si andranno ad affrontare, supportando ad esempio nativamente la rappresentazione di rotazioni in formato asse angolo e i quaternioni. Un altro vantaggio consiste nella sua piena compatibilità sia con il `namespace std` ma soprattutto con openCV, nel quale sono presenti metodi come `cv::eigen2cv` e `cv::cv2eigen` che rendono agevole il passaggio di una matrice dall'una all'altra libreria.

Eigen soddisfa il vincolo prestazionale posto sul sistema, in quanto è sviluppato e migliorato per essere principalmente veloce, garantendo prestazioni eccellenti con ridotto carico computazionale, è inoltre estremamente affidabile, implementando algoritmi estremamente safe (o dichiarando chiaramente, all'interno della documentazione, dove le prestazioni rendono necessarie trade-off qualitativi).

La scelta di utilizzare Eigen è frutto anche della sua semplicità di installazione: la libreria è infatti composta solamente da un insieme di interfacce, le quali non hanno bisogno di librerie esterne o di compilazione, bensì è solamente necessario includere la cartella contenente Eigen nel progetto per poter utilizzare immediatamente tutti i metodi forniti, semplificando la futura integrazione di Eigen nell'ecosistema del software prodotto da Specialvideo.

Ultimo ma non meno importante Eigen è estremamente semplice, pulito e ben scritto, con una documentazione ben fatta, in grado di essere imparato ed utilizzato da un completo neofita in tempi estremamente

brevi, avendo una curva di apprendimento estremamente piatta. Tali proprietà semplificano, come da requisiti, la manutenibilità, leggibilità e riusabilità del codice scritto, oltre a semplificare enormemente l'implementazione di algoritmi matematici complicati, liberando il programmatore da bug di derivazione matematica dando la possibilità di concentrarsi nella corretta implementazione dell'algoritmo in se, senza doversi preoccupare della corretta implementazione di parti matematiche.

3.3.3 Ceres

Ceres Solver[2] è una libreria open source `C++` per modellare e risolvere problemi di ottimizzazione. É una libreria ben scritta, utilizzata fin dal 2010 da Google e sviluppata con il supporto stesso di Google.

Principalmente Ceres nasce per risolvere due tipi di problemi:

- problemi non lineari di minimizzazione ai minimi quadrati vincolati e
- problemi di ottimizzazione non vincolati.

Tuttavia, seppur molto performante, ben scritto e documentato, open source e con il supporto diretto di Google, l'utilizzo di tale libreria risulta necessario soltanto nel caso in cui si vogliano utilizzare tecniche di minimizzazione per risolvere il problema. La risoluzione scelta, al contrario, consiste unicamente nell'utilizzo di un algoritmo di scomposizione a valori singolari, nativamente supportato sia in Eigen, la cui integrazione non è per nulla problematica, sia in OpenCV, già presente nel contesto aziendale, rendendo non necessario l'utilizzo di Ceres.

Gli svantaggi nell'utilizzo di Ceres sono molteplici, e sono dovuti principalmente al peso della libreria nel momento in cui, in futuro, la si voglia aggiungere al prodotto aziendale. La libreria Ceres infatti si appoggia, a sua volta, a molte altre librerie quali Google Flags[4], Google Logs[5], SuiteSparse e CXSparse[8], rendendo la compilazione lenta e la libreria finale da includere nei progetti molto grande. Inoltre il supporto di Ceres per Windows è relativamente recente ed in via, al momento, sperimentale, in quanto alcune librerie su cui Ceres si basa quali SuiteSparse e CXSparse non hanno supporto ufficiale su Windows, che al contrario è

il principale ambiente di esecuzione dei prodotti di SpecialVideo. Si è pertanto deciso di non utilizzare Ceres per evitarne l'integrazione, con tutti i problemi derivanti, nel progetto; in questo modo se ne snelliscono i requisiti dello stesso e ne viene facilitata l'integrazione nell'ecosistema del prodotto già sviluppato in SpecialVideo, non ponendo problemi di compatibilità, e ne viene semplificata la manutenibilità, non dovendo tenere aggiornate, in futuro, un grosso numero di librerie per mantenere aggiornato il progetto, rimane tuttavia preso in considerazione nel caso, in futuro, si decida di utilizzare tecniche alternative a supporto di quanto sviluppato, inserendo la minimizzazione come tecnica risolutiva.

Capitolo 4

Implementazione

Nel seguente capitolo vengono mostrati più nel dettaglio gli aspetti implementativi dell'algoritmo, oltre ad una breve introduzione alle librerie ed ai metodi utilizzati, insieme ad una presentazione dei metodi sviluppati non strettamente inerenti all'algoritmo in se ma che sono stati realizzati per facilitare, in seguito, il testing dell'algoritmo su dati simulati in ambiente 3D ricostruito tramite motore grafico, per giungere infine al testing in ambiente reale quando il progetto sarà integrato nel prodotto aziendale.

4.1 Implementazione dell'algoritmo

4.1.1 Header Eigen a supporto dei quaternioni duali

Dal design appare immediatamente la necessità di effettuare, all'interno del sistema, una notevole quantità di operazioni di tipo algebrico su matrici, quaternioni e quaternioni duali. Per questo motivo, prima di partire con l'implementazione vera e propria del sistema, occorre integrare ad Eigen, libreria usata in maniera pervasiva all'interno del codice, dei metodi per semplificare il calcolo basato su quaternioni duali; Eigen infatti, seppure ben scritto, semplice, documentato e facilmente utilizzabile, non ne supporta nativamente l'utilizzo. É risultato pertanto conveniente realizzare alcuni header aggiuntivi per integrare, sfruttando le funzionalità già presenti all'interno della libreria, il supporto ai quaternioni duali,

in modo da renderne più semplice il calcolo e l'utilizzo all'interno del codice il quale, affiancato dalla documentazione, dovrebbe essere facilmente comprensibile anche a chi conosce sommariamente l'algoritmo e la matematica utilizzata a supporto dello stesso, facilitando perciò la manutenibilità del sistema, anche nel caso in cui si decida di aggiornare le librerie utilizzate o modificare interamente l'algoritmo stesso.

DualQuaternion.h Viene per prima aggiunta una semplice libreria per definire la classe DualQuaternion, sfruttando il supporto nativo ai quaternioni fornito da Eigen. I prototipi dei metodi principali, in pseudocodice, sono:

- `DualQuaternion(...)`: costruttore di un oggetto quaternione duale $\check{q}$, può essere costruito vuoto, a partire da due quaternioni, da un quaternione e da un vettore traslazione, o identità.
- `conjugate()`: restituisce il coniugato di $\check{q}$.
- `dual()`: restituisce la parte duale di $\check{q}$ q'.
- `exp()`: calcola l'esponenziale di $\check{q}$
- `fromScrew(...)`: costruisce $\check{q}$ a partire dalla sua rappresentazione *Screw-Axis*.
- `identity()`: restituisce un quaternione duale identità.
- `inverse()`: restituisce $\check{q}^{-1}$.
- `log()`: calcola il logaritmo di $\check{q}$.
- `norm()`: calcola $|\check{q}|$.
- `normalize()`: normalizza $\check{q}$.
- `normalized()`: restituisce $\check{q}$ normalizzato.
- `transformPoint(point` p`)`: restituisce le coordinate del punto subita la trasformazione rappresentata da $\check{q}$.

- `transformVector(vector` v`)`: restituisce il vettore trasformato da $\check{q}$.
- `real()`: restituisce la parte reale di $\check{q}$.
- `rotation()`: restituisce la parte rotazione di $\check{q}$.
- `translation()`: restituisce la parte traslatoria di $\check{q}$ in forma vettoriale.
- `translationQuaternion()`: restituisce la parte traslatoria di $\check{q}$ come quaterione.
- `toMatrix()`: restituisce una matrice 4×4 equivalente alla trasformazione rappresentata da $\check{q}$.
- `zeros()`: annulla $\check{q}$.

L'utilizzo di tale interfaccia permette l'utilizzo semplice e trasparente, in Eigen, dei quaternioni duali, grazie al supporto nativo dato dalla libreria ai quaternioni tradizionali stessi.

EigenUtils.h Viene inoltre utilizzata una ulteriore interfaccia contenente un insieme di metodi di libreria utili per il calcolo fra matrici, anch'essa utilizzato per rendere più semplice e leggibile il codice.

I metodi principali, in pseudocodice, di questa interfaccia, sono:

- `skew(vector` t`)`: restituisce la matrice traccia 3×3 del vettore t.
- `AngleAxisToRotationMatrix(vector` r`)` e `RotationToAngleAxis(mat` R`)`: permette di trasformare una rotazione in forma di matrice di rotazione 3×3 in un vettore rotazione r in forma Angle-Axis e vice versa.
- `AngleAxisToQuaternion(vector` r`)` e `QuaternionToAngleAxis(quaternion` q`)`: permette di trasformare il quaternione q rappresentante la rotazione del vettore rotazione r in forma Angle-Axis e vice versa.

- `QuaternionToRotation(quaternion` q`)`: trasforma il quaternione q in una matrice 3×3 di rotazione R

- `AngleAxisAndTranslationToScrew(vector` r`, vector` t`, vector` l`, vector` m`, double` θ`, double` d`)`: trasforma i due vettori r e t rappresentanti una rotazione in forma Angle-Axis ed una traslazione nella rappresentazione della Screw Theory, ovvero nell'asse di rotazione l ed il suo momento m, nell'angolo di rotazione θ attorno all'asse e la traslazione d sull'asse.

- `RPY2mat()` e `mat2RPY()`: permette di convertire un vettore di rotazione composto dagli angoli di Eulero (Roll Pitch e Yaw) in una matrice di rotazione 3×3 e vice versa.[1].

Tali metodi permettono di passare agevolmente da diverse rappresentazioni equivalenti per le trasformazioni, siano essere trasformazioni omogenee composte da rotazioni e traslazioni o siano esse soltanto semplici rotazioni. Particolare interesse risiede nel metodo `AngleAxisAndTranslationToScrew()` basandosi l'algoritmo di Daniilidis sul concetto di Asse Angolo rappresentato come quaternione duale `[B]`.

4.1.2 Algoritmo completo

L'implementazione è stata, come da design, effettuata interamente in `C++` in ambiente di sviluppo Visual Studio, ricordando durante l'implementazione dei vincoli di portabilità e riusabilità del codice, in linea con le necessità aziendali.

L'esecuzione del programma viene schematizzata nel flow chart mostrato in figura 4.1, le cui diverse fasi vengono illustrate, più in dettaglio, in seguito.

[1] All'interno dell'implementazione è stato cercato di evitare il più possibile gli angoli di Eulero per il ben noto problema del gimbal lock, inoltre esistendo diverse notazioni per l'ordine delle rotazioni si è preferito utilizza.

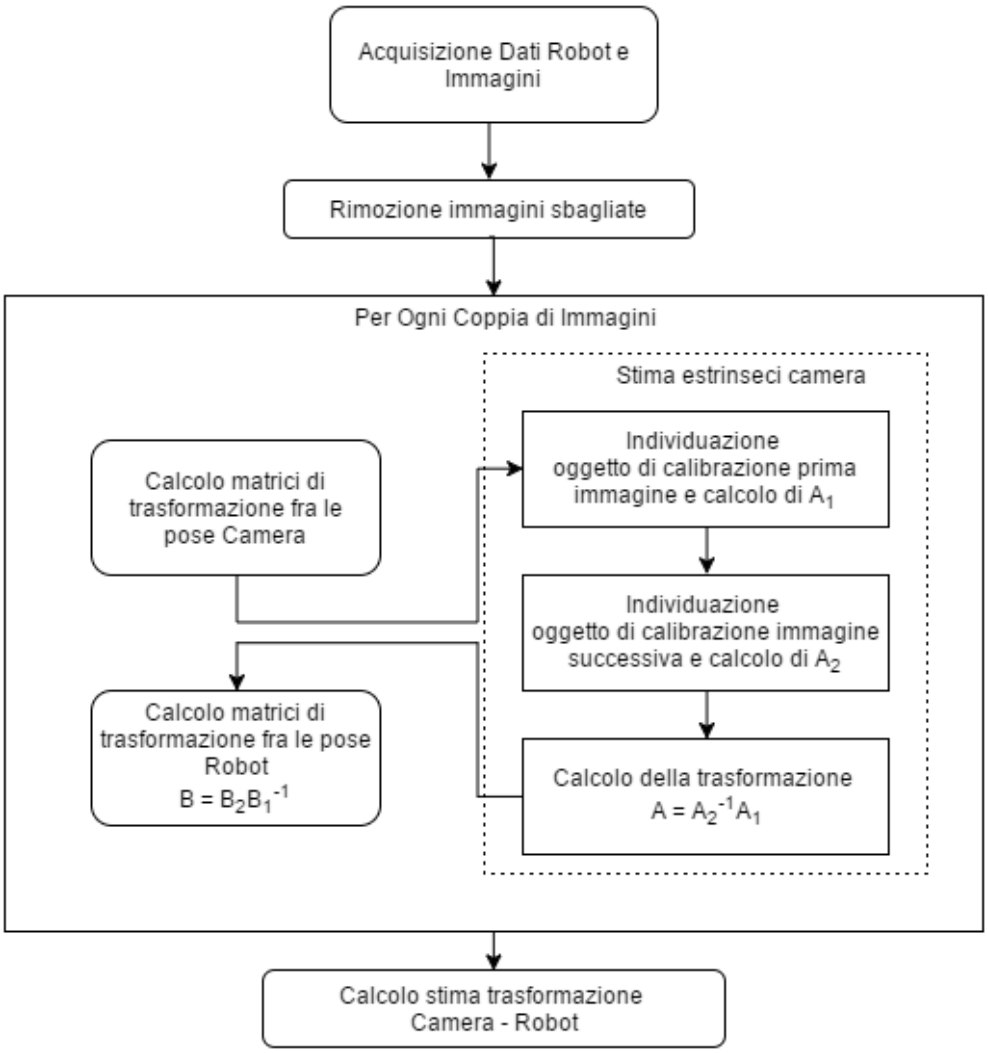


Figura 4.1: Gli step di esecuzione dell'algoritmo. A destra, all'interno del ciclo per ogni coppia di immagini, è stata divisa la parte del programma utilizzante la libreria OpenCV, Eigen invece viene utilizzato in tutto il progetto.

Acquizione dati robot e Immagini e calcolo delle trasformazioni fra le pose di camera e robot

La prima fase della calibrazione consiste nell'acquisizione delle immagini e delle posizioni corrispondenti del robot. Tale operazione può essere fatta sia automaticamente, definendo un percorso prestabilito, sia manualmente, muovendo il robot e scattando la fotografia manualmente, registrando i dati del robot per ogni posizione. Tale procedura non è definibile univocamente in quanto dipende fortemente dal modello utilizzato e dalle API fornite dal costruttore per interagire con esso. Occorre, in fase di acquisizione, ricordare le linee guida per garantire una buona stima della calibrazione, ovvero:

- Effettuare spostamenti il più ampi possibile, soprattutto riguardo alle rotazioni.
- Riprendere interamente l'oggetto di calibrazione ben illuminato, in modo da garantire un facile ed ottimo detecting.
- Non riprendere l'oggetto di calibrazione da troppo lontano.
- Utilizzare un oggetto di calibrazione ben stabile e di dimensioni ben note.
- Avere, in precedenza, calibrato il sistema di visione in maniera più accurata possibile.
- Ottenere, compatibilmente con il tempo e la potenza computazionale a disposizione, il maggior numero di pose possibile, il più diverse fra loro possibile.

Effettuata l'operazione di acquisizione viene fatto un controllo sui dati ottenuti, sfruttando il metodo

```
bool findChessboardCorners(
        InputArray image,
        Size patternSize,
        OutputArray corners,
        int flags);
```

per controllare che la scacchiera sia presente all'interno dell'immagine. Grazie a questa funzione è possibile individuare automaticamente, all'interno dell'immagine `image`, una scacchiera di dimensioni `patternSize`. In `corners` viene messa, in ordine, una lista di punti dell'immagine corrispondenti agli incroci dei quadrati nella scacchiera. L'ultimo parametro `flags` permette di impostare dei flag per modificare il comportamento dell'esecuzione dell'algoritmo. Il metodo ritorna `true` nel caso in cui la scacchiera sia individuata, `false` altrimenti. La computazione di questa funzione è estremamente rapida e permette di validare un set di dati acquisito prima della computazione vera e propria, cosa che può rivelarsi utile ad esempio nel caso in cui l'acquisizione sia stata effettuata automaticamente per poterla ripetere.

É a questo punto possibile ricavare i dati che verranno passati in ingresso all'algoritmo di Hand Eye Calibration vero e proprio il quale, come si vedrà in seguito, richiede in ingresso quattro liste di vettori $\mathbb{R}^3$ `rvecs1`, `tvecs1`, `rvecs2` e `tvecs2`, che, a coppie, descrivono le trasformazioni subite dalla flangia del robot (`rvec2` e `tvec2`) in ogni posizione e la rispettiva trasformazione della camera (`rvec1`, `tvec1`) ad essa. I vettori di traslazione sono rappresentati nel classico formato $\texttt{tvec} = \vec{t} = [x, y, z]$ mentre i vettori `rvec` di rotazione sono rappresentati in formato asse angolo (Angle Axis) definito come

$$\begin{aligned} \vec{r} &= \vec{v_x} * \theta_{angle} \\ con & \\ \|\vec{v_x}\| &= 1 \\ \|\vec{r}\| &= \theta_{angle} \end{aligned}$$

dove $\vec{v_x}$ è un versore $[x_{axis}, y_{axis}, z_{azis}]$ rappresentante l'asse di rotazione e θ_{angle} è la rotazione sull'asse.

Viene utilizzata questa notazione per evitare problemi derivanti dall'ambiguità nell'uso di notazioni alternative, quali ad esempio gli angoli di Eulero, per cui ad esempio risulta fondamentale specificare in che ordine vengono eseguite le rotazioni. Tale soluzione, oltre ad evitare ambiguità,

risulta essere facilmente implementabile grazie alle librerie mostrate in precedenza.

Trasformazione fra le pose del robot

Per ogni posizione assunta $i = 1, \ldots, n$ dal robot si ha, secondo una certa convenzione[2], la posizione ed orientazione rispetto al proprio sistema di riferimento. Per spostamento del robot si ha una coppia i, j di queste posizioni, equivalenti ad uno spostamento del robot, dalla quale è possibile calcolare, a partire dai dati del robot salvati in fase di acquisizione, la trasformazione chiamata B in figura 4.2, definita $B = B_j^{-1} B_i$, dove B, B_i e B_j sono matrici di trasformazioni omogenee dalla base del robot al tool.

Per decomporre la matrice B in `tvec2` e `rvec2` è sufficiente scrivere B come

$$\begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}$$

da cui `rvec2` si calcola semplicemente estraendo la matrice di rotazione R e passandola al metodo precedentemente descritto `RotationToAngle-Axis(...)` mentre l'ultima colonna t equivale a `tvec2`.

Trasformazione fra le pose della camera

Per ogni posizione assunta $i = 1, \ldots, n$ per cui si calcola la trasformazione fra le pose del robot si calcola, contemporaneamente, la corrispondente coppia di vettori `rvec1` e `tvec1`, i quali descrivono la trasformazione della camera A di figura 4.2. Le trasformazioni della camera devono

[2] É necessario porre particolare attenzione alla convenzione con la quale vengono forniti i dati del robot, soprattutto riguardo alla rappresentazione della rotazione. Nel caso in cui, ad esempio, vengano forniti gli angoli di Eulero, occorre sapere con che notazione (ovvero in che ordine) sono effettuate le rotazioni. Un metodo pratico e sicuro è passare il prima possibile a notazioni non ambigue, come ad esempio i quaternioni o la matrice di rotazione, in modo da lasciare il codice successivo indifferente alle diverse notazioni utilizzate.

essere calcolate a partire dalle immagini acquisite sfruttando i metodi messi a disposizione dalla libreria OpenCV.

Supponendo di aver precedentemente calibrato la camera, avendo quindi a disposizione la matrice degli intrinseci, è possibile estrarre con ragionevole accuratezza la posizione della camera rispetto alla scacchiera (o qualsiasi altro oggetto di calibrazione si decida di utilizzare) a patto di conoscerne le proprietà geometriche con precisione e di poterle riconoscere all'interno dell'immagine. Per semplicità viene utilizzata una scacchiera, sia per la facilità con cui questa può essere costruita e posta in una posizione inquadrabile, sia per il supporto che OpenCV fornisce nativamente per l'utilizzo di questa come oggetto di calibrazione.

Chiamando il metodo `findChessboardCorners()` descritto in precedenza è possibile ottenere una lista di punti corrispondenti agli angoli della scacchiera. É buona norma chiamare successivamente il metodo

```
cornerSubPix(
        InputArray image,
        InputOutputArray corners,
        Size winSize,
        Size zeroZone,
        TermCriteria criteria)
```

per rifinire i punti risultato, rendendo questi ultimi più precisi e più resistenti al rumore.

Individuati i punti della scacchiera nell'immagine è possibile ottenere la posa della camera rispetto alla scacchiera con la chiamata

```
bool solvePnP(
        InputArray objectPoints,
        InputArray imagePoints,
        InputArray cameraMatrix,
        InputArray distCoeffs,
        OutputArray rvec,
        OutputArray tvec,
        bool useExtrinsicGuess,
        int flags);
```

il quale a partire di una descrizione geometrica dell'oggetto di calibrazione `objectPoints`, dai punti trovati dello stesso oggetto nell'immagine `imagePoints`, dai parametri intrinseci della camera `cameraMatrix` e dai coefficienti di distorsione di quest'ultima `distCoeffs` calcola, con precisione dipendente da vari fattori elencati in seguito, la posa della camera già decomposta in `rvec` e `tvec`. I due restanti parametri permettono di modificare l'algoritmo: `useExtrinsicGuess`, se settato a true, utilizza dei valori di `rvec` e `tvec` passati in ingresso come prima stima sulla posa della camera (accelerando di conseguenza l'algoritmo), `flags` invece permette di impostare flag particolari per modificare l'esecuzione dell'algoritmo.

É importante che `objectPoints` e `imagePoints` siano nello stesso ordine, in quanto semplici vettori, per garantire la corretta esecuzione dell'algoritmo.

SolvePnP restituisce gli angoli della posa con la notazione di Rodrigues: è pertanto necessario, tramite la chiamata al metodo `void Rodrigues(InputArray src, OutputArray dst)` convertire gli stessi in forma di matrice di rotazione 3×3 per poter costruire la matrice di trasformazione dalla camera alla scacchiera.

A questo punto si procede analogamente con quanto fatto in precedenza per le pose del robot, a partire dalle due matrici di trasformazione A_i, A_j delle due posizioni i, j è possibile calcolare la matrice di trasformazione da una posizione all'altra A, decomponendola successivamente nei due vettori `tvec1` e `rvec1`.

Stima della trasformazione Camera-Robot

L'interfaccia mette a disposizione un solo metodo per il calcolo della calibrazione

```
static void estimateHandEye(
 const std::vector<Vector3d,
  Eigen::aligned_allocator<Vector3d>>& rvecs1,
 const std::vector<Vector3d,
  Eigen::aligned_allocator<Vector3d>>& tvecs1,
 const std::vector<Vector3d,
```

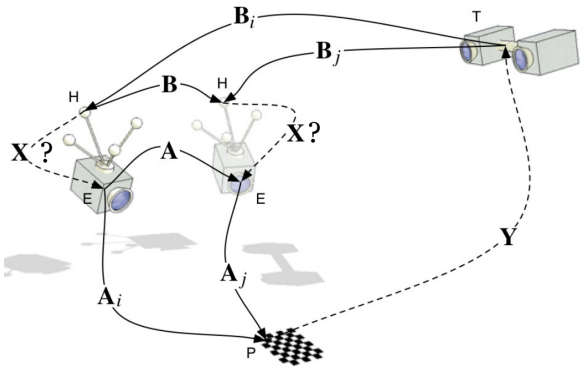


Figura 4.2: Una rappresentazione del problema di Hand Eye Calibrazione.

```
  Eigen::aligned_allocator<Vector3d>>& rvecs2,
const std::vector<Vector3d,
  Eigen::aligned_allocator<Vector3d>>& tvecs2,
Eigen::Matrix4d& H\12);
```

Il metodo `estimateHandEye` prende in ingresso 5 parametri, 4 di input ed un riferimento ad una matrice da utilizzare come output, fornito sotto forma di una matrice di trasformazione 4×4 H dal sistema di riferimento 1, quello della camera, al sistema di riferimento 2, del robot. Le liste textttrvecs1, `tvecs1`, `rvecs2` e `tvecs2` sono liste della stessa lunghezza n dove n è il numero di spostamenti effettuati, contenenti i vettori calcolati nei passi precedenti nelle notazioni sopra descritte.

Per ogni i-esimo spostamento effettuato dal robot si hanno quindi 4 vettori $\texttt{rvec1}_i$, $\texttt{tvec1}_i$, $\texttt{rvec2}_i$ e $\texttt{tvec2}_i$ dai quali è possibile costruire i due quaternioni duali $\breve{a}_i$ e $\breve{b}_i$ tramite il metodo `AngleAxisAndTranslationToScrew()`, dei quali si controlla che le parti scalari siano uguali, confermando, quindi, che si tratta della stessa trasformazione applicata a diversi sistemi di riferimento.

Viene creata, per ogni coppia $(\breve{a}_i, \breve{b}_i)$ il blocco S_i^T tramite la chiamata del metodo `ScrewToStransposeBlockofT()`

$$S_i^T = \begin{pmatrix} \vec{a_i} - \vec{b_i} & [\vec{a_i} + \vec{b_i}]_\times & 0_{3\times1} & 0_{3\times3} \\ \vec{a_i}' - \vec{b_i}' & [\vec{a_i}' + \vec{b_i}']_\times & \vec{a_i} - \vec{b_i} & [\vec{a_i} + \vec{b_i}]_\times \end{pmatrix}$$

le quali formano, insieme, la matrice $T = (\ S_1^T \quad S_2^T \quad \dots \quad S_n^T\ )^T$.

Viene, generata la matrice T, calcolata la sua decomposizione a valori singolari sfruttando Eigen con la chiamata

```
Eigen::JacobiSVD<Eigen::MatrixXd>
  svd(T,
      Eigen::ComputeFullU | Eigen::ComputeFullV);
```

la quale implementa la decomposizione SVD garantendo stabilità ed affidabilità. I parmetri `Eigen::ComputeFullU` e `Eigen::ComputeFullV` sono necessari in quanto, di default, la classe `JacobiSVD` calcola soltanto i valori singolari Σ della decomposizione $T = U\Sigma V^T$.

Si controlla, effettuato il calcolo della decomposizione, che gli ultimi due valori singolari siano tendenti a zero, accedendo ai valori singolari calcolati tramite la chiamata al metodo `svd.singularValues()`.

Confermato che la matrice T è di rango 6 e che quindi gli ultimi due vettori singolari destri ricoprono lo spazio nullo di T, si ricava dalla matrice V $\vec{v_7}$ e $\vec{v_8}$.

Posti $v_7^T = (u_1^T, u_2^T)$ e $v_8^T = (v_1^T, v_2^T)$ risulta banale a questo punto calcolare due valori di s per cui

$$\lambda_1 \vec{u_1}^T \vec{v_1} + \lambda_1\lambda_2(\vec{u_1}^T \vec{v_2} + \vec{u_2}^T \vec{v_1}) + \lambda_2^2 \vec{u_2}^T \vec{v_2} = 0$$

con $s = \lambda_2/\lambda_1$ sfruttando la funzione `solveQuadraticEquation(...)`, funzione di libreria implementata per alleggerire il codice da calcoli elementari di risoluzione per una funzione quadratica.

Per ognuno dei due valori di s così ottenuti si calcola il valore della funzione $s^2 \vec{u_1}^T \vec{u_1} + 2s\vec{u_1}^T \vec{u_2} + \vec{u_2}^T \vec{u_2}$, scegliendo il valore di s per cui la funzione restituisce il risultato maggiore s_+. Viene calcolato λ_2 a partire da questo valore di s

$$\lambda_2 = \sqrt{\frac{1}{s_+}}$$

e poi banalmente $\lambda_1 = s_+\lambda_2$ dove s_+ sta ad indicare il valore di s per cui si era ottenuto il risultato maggiore.

La soluzione, a questo punto, è data dal quaternione duale $\check{q}$ composto da (q, q') con $q = \lambda_1 v_7$ e $q' = \lambda_2 v_8$, restituito in forma matriciale 4×4 dal metodo `H_12 = dq.toMatrix()` con `dq` $= \check{q}$.

Capitolo 5

Analisi dei risultati

Viene fornita, in questo capitolo, la metrica con la quale è stata misurata l'efficacia dell'algoritmo. A seguire si presenta una breve analisi dei risultati ottenuti dall'algoritmo su diversi set di dati simulati in presenza di diversi tipi di errori, con diversi movimenti del sistema camera robot e diverse quantità di movimenti, confrontando i risultati ottenuti.

5.1 Metrica per la misurazione dell'errore

Nel test dell'algoritmo si è scelto di utilizzare la metrica proposta da Strobl e Hirzinger, già vista in precedenza, per valutare la precisione della trasformazione ottenuta, in modo da poter confrontare i risultati ottenuti nei diversi casi di test ottenendo un risultato numerico facilmente confrontabile.

Per ogni test effettuato si conosce in partenza la trasformata reale fra camera e robot $\hat{X}$ da confrontare con la soluzione stimata $\tilde{X}$. Le matrici vengono decomposte in parte rotatoria e traslatoria. L'errore nella stima della parte rotatoria di ottiene calcolando prima il residuo

$$_{\Delta}\tilde{R}_X = \tilde{R}_X^{-1}\hat{R}_X$$

da cui si ottiene l'errore di rotazione

$$\Upsilon_{rot} = \arccos\left(\frac{trace(_{\Delta}\tilde{R}_X) - 1}{2}\right)$$

L'errore di traslazione invece è la semplice distanza euclidea fra le due traslazioni

$$\Upsilon_{tra} = \frac{\|\tilde{t}_X\| + \|\tilde{t}_X\|}{2}$$

5.2 Test effettuati su dati simulati

Per testare l'algoritmo si è realizzato un software di test il quale, a partire da una trasformazione arbitraria fra flangia del robot e camera ad essa collegata, simula una serie di spostamenti del robot, calcolando a partire dalla trasformazione conosciuta le pose assunte dalla camera. A questa serie di coppie di posizioni Camera Robot è possibile aggiungere diversi tipi di errori e rumori, simulando ciò che avviene nella realtà aziendale: è possibile ad esempio simulare errori maggiori nelle diverse componenti, come nelle traslazioni nel caso di robot imprecisi, o diversi tipi di errore, siano essi gaussiani, sistematici etc. Da questo set di dati, generato artificialmente, viene facilmente calcolato l'insieme delle trasformazioni da una posa alla successiva, sia per il Robot, che equivalgono a quelle utilizzate in partenza per spostare il robot nelle varie posizioni, sia quelle della camera, che dipendono dalla posizione della camera relativa al robot, ovvero ciò a cui siamo interessati. Tali trasformazioni vengono successivamente passate all'algoritmo di calibrazione implementato, ottenendo la stima della trasformazione utilizzata in partenza, potendola confrontare con quest'ultima.

In questa fase di test si vuole misurare l'efficacia del metodo di calibrazione proposto indipendentemente dalla fase di estrazione degli estrinseci realizzata tramite OpenCV, la quale può tuttavia largamente influenzare la stima della posa. Il calcolo della posa del robot viene infatti influenzato solamente dalla precisione del robot stesso, dipendente dalle specifiche tecniche del robot in se; per quanto riguarda l'estrazione degli estrinseci della camera, invece, si hanno errori dipendenti da molteplici cause.

Il calcolo della posa della camera viene prima di tutto influenzata dalla qualità ottica del sistema di visione e dalla calibrazione dello stesso, infatti una calibrazione migliore permette, tramite l'uso della chiamata `SolvePnP(..)` di OpenCV, di ottenere una stima della posa della camera più precisa, tanto più è accurata la calibrazione dei parametri intrinseci. Influisce sul risultato finale anche la scelta dell'oggetto di calibrazione: è stato ad esempio dimostrato che l'utilizzo di una griglia di punti restituisce risultati più precisi rispetto alla scacchiera, generando parametri estrinseci più precisi. La stima della posa della camera viene influenzata anche dalle proprietà fisiche dell'oggetto di calibrazione, che deve essere il più preciso possibile: una impercettibile curvatura di una scacchiera, ad esempio, può influenzare la stima dell'angolo di questa rispetto all'asse ottico, generando un errore che di conseguenza si propagherebbe alla stima della calibrazione Hand Eye. Ancora la distanza del sistema di visione dall'oggetto di calibrazione, così come l'illuminazione e la risoluzione del sistema di visione, influiscono sulla corretta stima della posa, in quanto un'immagine piccola a causa di un sensore minuto, oppure a causa dell'eccessiva distanza della camera dalla scacchiera, portano necessariamente ad una stima più imprecisa.

Per tutti questi motivi si è voluto testare l'algoritmo isolandolo da queste problematiche, non inerenti al caso specifico ma ben attuali e di primaria importanza in qualsiasi applicazione reale che necessiti la calibrazione, utilizzando nei test dati fabbricati e simulati. Un altro vantaggio nell'utilizzo di dati simulati consiste nel poter valutare il sistema a diversi gradi di errore e con diversi tipi di rumore, valutandone la resistenza ed efficacia in diverse simulazioni, rappresentanti casi reali, misurandone la precisione ed accuratezza.

Tutti i test vengono effettuati su una configurazione del problema simile ad un caso reale:

- La trasformazione camera-robot ha rotazione completamente casuale e traslazione rispetto alla flangia di $\pm$ 50 mm in tutte le direzioni.
- Il punto di partenza del robot è a 1000mm dall'oggetto di calibrazione, con la camera che punta all'oggetto.

- Il robot è in grado di muoversi in un cubo di 1 metro per lato, potendo raggiungere rotazioni di $\pm$ 45° sugli assi paralleli al piano della telecamera, libero sull'asse Z, ipotizzando che la camera abbia un angolo di visuale tale da ritrarre sempre l'oggetto di calibrazione.

- Per ogni spostamento viene salvata la posizione del robot e della camera.

- Il robot reale ha una precisione di 0.1-0.2 mm, con test di ripetibilità ottenuti di 0.08mm (ma valore assoluto più ampio dipendente dalla posa del braccio), con errori che possono essere anche a media non nulla (errori sistematici).

- Un risultato è considerato accettabile se avente accuratezza inferiore ai 0.5cm di traslazione e 1° per componente di rotazione.

Diverse combinazioni di parametri influiscono sul risultato finale della calibrazione, sui quali si può agire per controllare l'efficacia del motodo proposto e simulare diversi casi d'uso del sistema:

- Il sistema è soggetto a diversi gradi di imprecisione, il errore assoluto di $\pm$ n mm/° viene simulato con distribuzione gaussiana con $\sigma = n$ e media nulla, ipotizzando che l'errore assoluto sia prima deviazione standard della misurazione.

- Il sistema viene testato con imprecisione nelle misurazioni del robot, della camera od entrambe, sporcando i dati generati con l'errore sopra descritto con diverse quantità *sigma*.

- Il sistema viene testato con un diverso numero di spostamenti del robot.

Per ogni combinazione valutata, in diversi casi specifici, vengono effettuate 200 simulazioni con altrettanti dataset casualmente generati, graficandone l'andamento.

Test in assenza di errore

Viene per prima cosa valutata l'efficacia del metodo utilizzato in assenza di errore, valutandone le prestazioni del sistema in relazione all'aumentare degli spostamenti effettuati dal robot. Sappiamo dalla teoria che due soli spostamenti sono sufficienti ad ottenere una perfetta calibrazione, a patto che gli assi di rotazione delle screw degli spostamenti non siano parallele. Dai risultati ottenuti tuttavia se ne ricava che con due soli spostamenti la precisione del metodo è molto scarsa, con oltre la metà dei risultati (mediana) superiori a 31.3mm e 2.45°, alcuni dei quali anche di ordini grandezza superiori (il risultato peggiore dista 35cm nella parte traslatoria e 180° nella parte rotatoria). Tale fenomeno può essere giustificato dagli errori commessi quando, nel dataset generato casualmente, i due spostamenti sono troppo simili, portando in alcuni casi valori decisamente errati. Si è però avuto conferma anche del fenomeno opposto, ovvero che due spostamenti sono sufficienti a calcolare la calibrazione a patto che questi siano sufficientemente ampi, come dimostrato dal fatto che, per una parte significativa delle simulazioni effettuate, il risultato ottenuto è stato ottimale, come dimostrato dal decimo percentile per cui si ha un errore di traslazione inferiore a 3.98 mm e di rotazione inferiore a 0.379°.

Tuttavia anche in presenza di un numero maggiore di spostamenti i risultati possono essere molto imprecisi, rendendo necessaria la rimozione degli outlier dai dati, compatibilmente con quanto succede in ambiente reale: un risultato sbagliato di uno, o in alcuni casi addirittura due o tre ordini di grandezza superiore a quanto previsto sarebbe infatti facilmente individuabile e porterebbe ad una immediata ripetizione della procedura di calibrazione. La pulizia dei dati è stata effettuata generando un istogramma contenente tutti i risultati, di cui un esempio è proposto nell'istogramma in figura 5.2a, effettuando una media mobile per effettuare un'operazione di smoothing sullo stesso e poi individuando un valore di soglia utilizzando il metodo del triangolo[1], scartando i valori maggiori

[1]Il metodo del triangolo consiste nel tracciare nell'istogramma una linea virtuale dal massimo valore raggiunto (peak) al valore più lontano da questo (nel nostro caso solitamente il massimo valore raggiunto), calcolando, per ogni punto compreso fra questi, la distanza dell'istogramma dalla linea. Il punto che ottiene valore massimo

alla soglia calcolata. La distribuzione dei dati rimanenti si avvicina in maniera decisamente più marcata ad una distribuzione normale, risultando in una distribuzione right-skewed, come testimoniato dall'istogramma in figura 5.2b, rendendo possibile l'analisi statistica dei dati. Si è notato inoltre che il numero di outlier si riduce esponenzialmente con l'aumentare degli spostamenti del robot: se con due soli spostamenti la maggior parte dei risultati viene marcata outlier e rimossa, nel caso d'esempio con 3 spostamenti soltanto 18 risultati sono stati marcati outliers, arrivando a zero con 6 spostamenti, a testimoniare che, con un numero sufficiente di spostamenti, l'algoritmo ha una buona precisione, arrivando ad ottenere un errore dell'ordine del decimo di millimetro e centesimo di grado con varianza tendente a zero.

Test in presenza di errore su robot o camera

Il test seguente è stato effettuato ipotizzando diversi gradi di errore su uno dei dispositivi, ipotizzando che l'altro, al contrario, sia perfetto. Tale test è significativo in quanto, come si è visto in precedenza, la fase di stima della posa del robot e della camera vengono effettuate l'una indipendentemente dall'altra, è pertanto interessante vedere come uno solo di questi influisca sulla calibrazione. L'accuratezza con cui viene stimata la posa del robot rispetto al proprio sistema di riferimento dipende interamente dalle specifiche tecniche del robot stesso, dalla sua posa, dalla precisione tecnica con cui è stato costruito, dal tipo di movimento e da altri fattori dipendenti dalla qualità del robot in se, il cui errore assoluto nelle misurazioni, così come le specifiche tecniche, sono solitamente descritte nel data sheet fornito dal costruttore. L'accuratezza della camera invece dipende da tutt'altri fattori, come la qualità del sistema di visione (ottica e sensore), dalla qualità della calibrazione degli intrinseci effettuata in precedenza, dalla qualità dell'illuminazione della scena, dal tipo di dispositivo di calibrazione utilizzato. Si effettuano diverse prove, in maniera simile a quanto visto in assenza di errore, mettendo in relazione la precisione ed accuratezza della calibrazione in relazione al numero di spostamenti effettuati dal robot e all'errore assoluto della misura. Come

viene scelto come soglia.

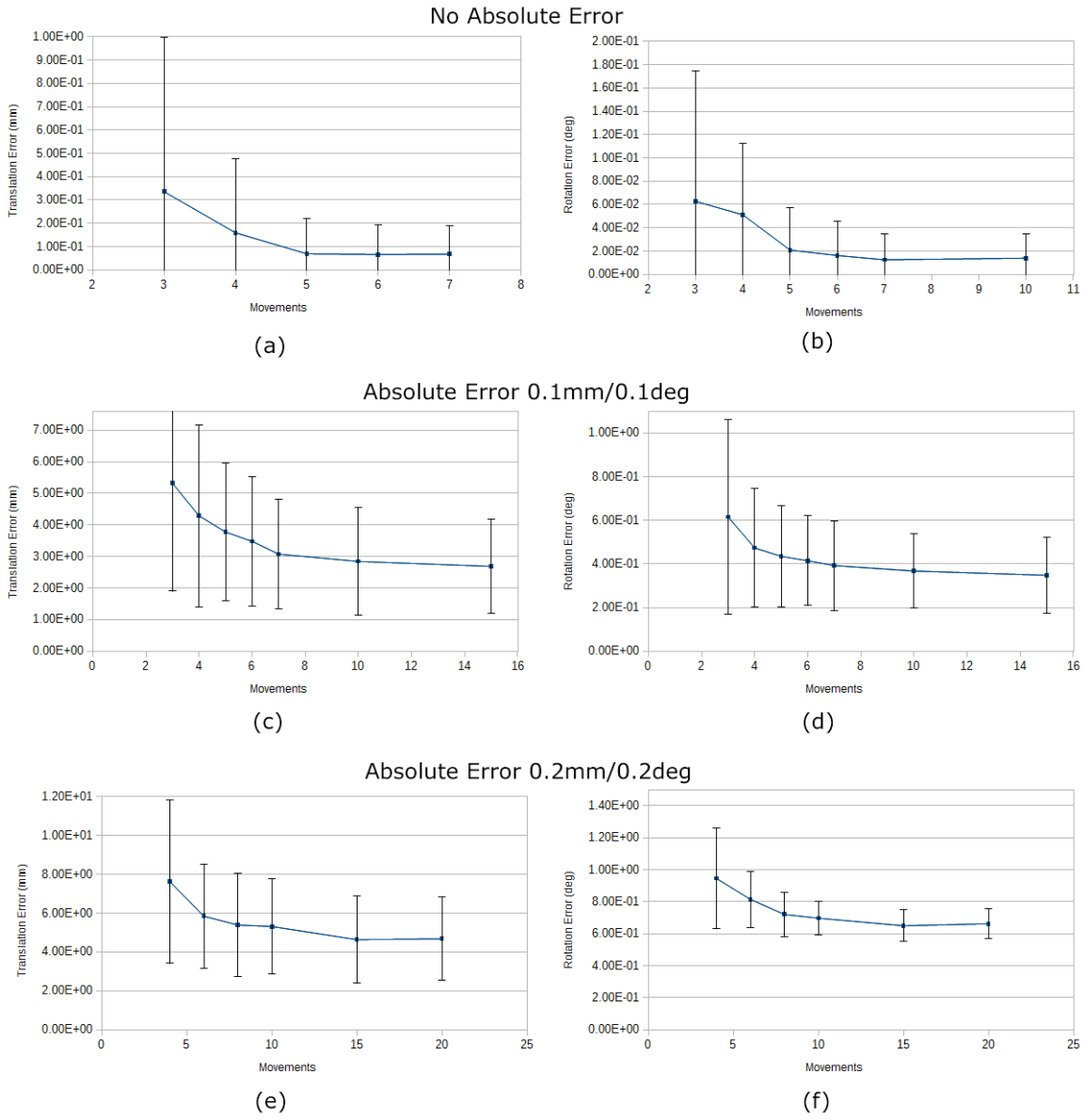


Figura 5.1: I grafici mostrano la precisione ed accuratezza delle misure ottenute, in relazione al numero di spostamenti effettuati dal robot. Sull'asse Y è presente la media dei risultati, mostrando inoltre la deviazione standard dei risultati, dopo essere stati puliti dagli outliers presenti ove necessario. Si nota in (a) e (b) come, in assenza di errore, si ottenga una calibrazione molto precisa con pochi spostamenti, mentre ne occorrono un numero maggiore per ottenere una calibrazione precisa in presenza di errore.

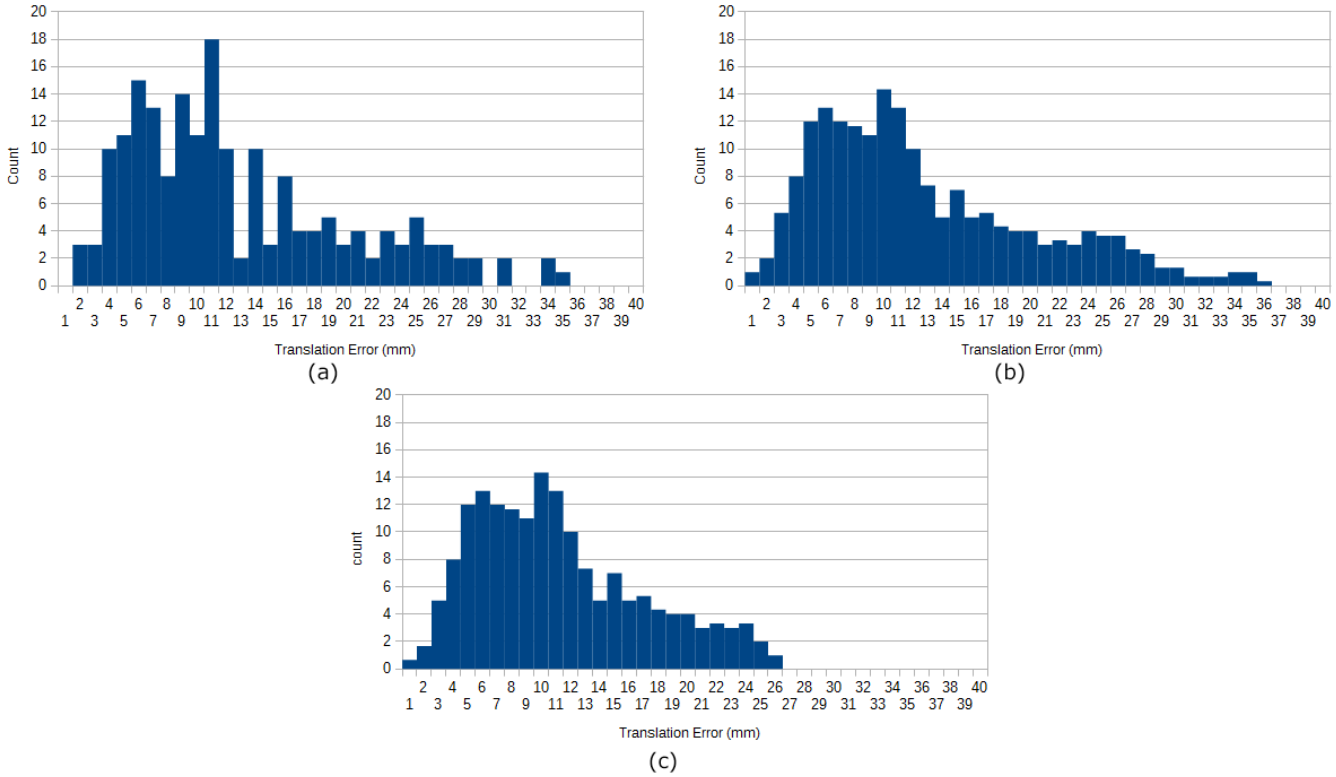


Figura 5.2: Un esempio di rimozione degli outliers dai risultati. Il grafico mostra la distribuzione dei risultati dell'errore di traslazione nel caso di 3 spostamenti effettuati con un errore di 0.3mm/0.3° sul sistema robot (a), sul quale viene effettuato uno smoothing tramite media mobile (b) ed infine rimuovendo gli outliers con il metodo dei triangoli (c). Si può notare con in (b) i dati siano distribuiti lungo tutto l'asse x, nel quale inoltre non sono presenti 7 valori superiori al valore massimo raggiunto dall'asse in figura poiché troppo distanti, arrivando fino ad un massimo di 1011.75mm di errore. Dopo la rimozione di 18 risultati considerati outliers i dati sono più concentrati, arrivando ad una distribuzione right skewed.

spiegato nel capitolo precedente, i dati ottenuti sono stati filtrati rimuovendo gli outlier per i casi con un basso numero di spostamenti, come visto in precedenza.

Si può notare dai grafici mostrati nelle figure 5.1 e 5.3 che, come per il caso senza errori, l'algoritmo tende a convergere ed avere risultati sempre più precisi, con deviazione standard minore, aumentando il numero di spostamenti. Tuttavia è interessante notare che, superata una certa soglia di spostamenti, il risultato non migliora significativamente. Tale comportamento mostra l'esistenza di una relazione diretta fra errore compiuto nelle acquisizioni e massima precisione raggiungibile dall'algoritmo, per cui, superata una certa soglia di spostamenti, non si riesce ad ottenere una calibrazione più precisa, in quanto l'errore introdotto nella misurazione non migliora la stima della calibrazione finale. Tale informazione è confermata dal grafico mostrato in figura 5.4, nel quale si mette in relazione il valore assoluto dell'errore commesso in misurazione con l'accuratezza dell'algoritmo nel caso di 15 spostamenti effettuati, avendo visto, dai grafici precedenti, che tale numeri di spostamenti è sufficiente ad ottenere un'elevata precisione dell'algoritmo. Si può notare che, anche con un errore elevato (1mm e 1° su ogni componente), molti risultati siano comunque relativamente buoni (decimo percentile 12.9mm/1.84°), tuttavia un'errore così ampio porta l'algoritmo a non essere affidabile rispetto alle specifiche fornite nel caso di errori troppo elevati.

Tale dato è interessante in quanto pone un vincolo importante al sistema, che deve perciò soddisfare requisiti di precisione e accuratezza nelle misure acquisite per garantire una stima affidabile della calibrazione, compatibile con i requisiti richiesti.

Si è inoltre notato dagli esperimenti che, ipotizzando l'errore su un solo sistema, non ci sono differenze nel risultato stimato nel caso in cui si applichi l'errore su un sistema o l'altro, ipotizzando che l'errore sia il medesimo. Ovvero, se viene posto lo stesso errore assoluto sulla camera, ipotizzando un perfetto comportamento del robot, si ottengo risultati del tutto simili, con identica precisione, accuratezza ed andamento dei dati. Si è pertanto deciso di non riportare i test per l'errore applicato alla sola camera, essendo perfettamente speculari a quanto già visto nel test precedente.

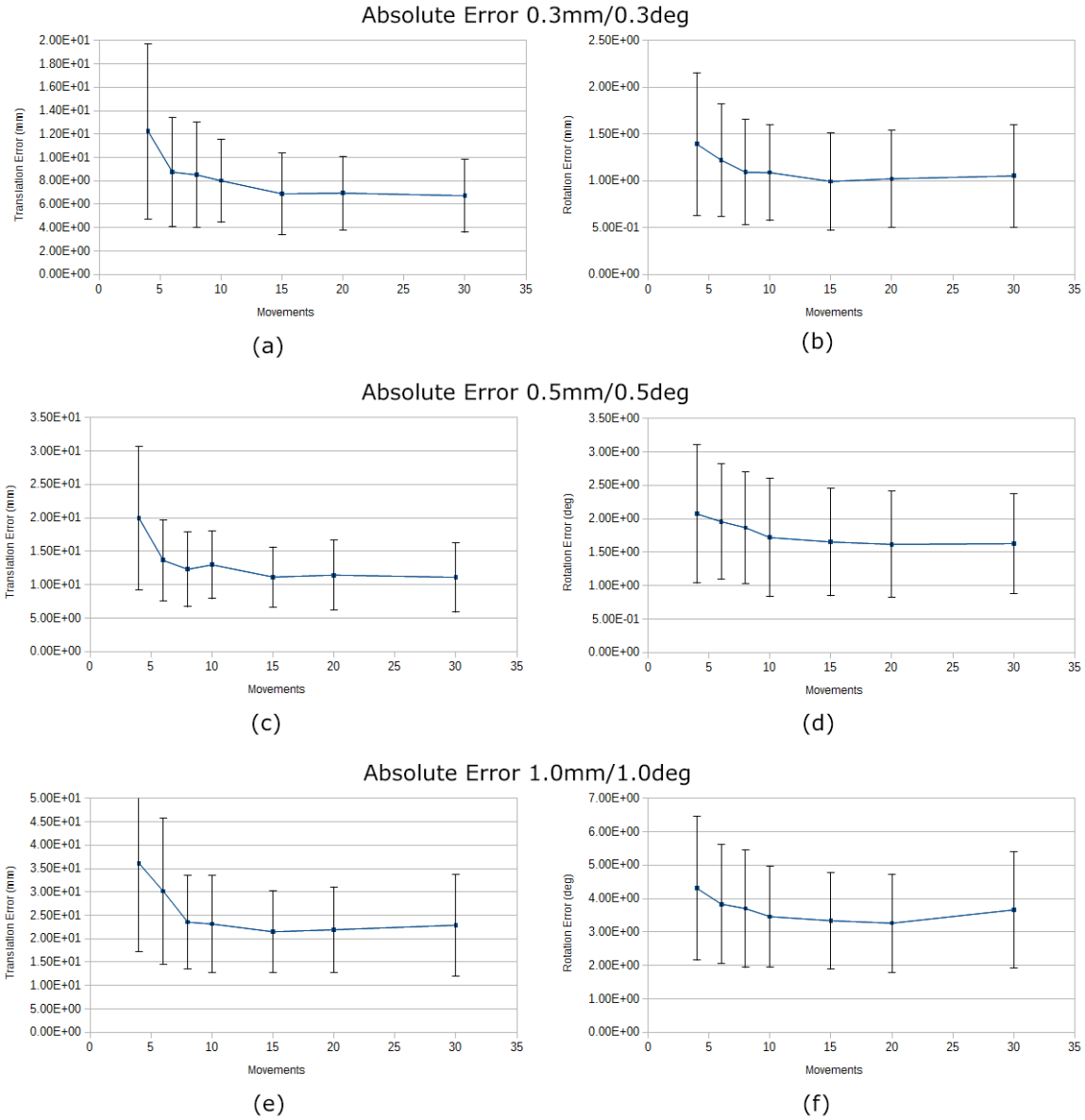
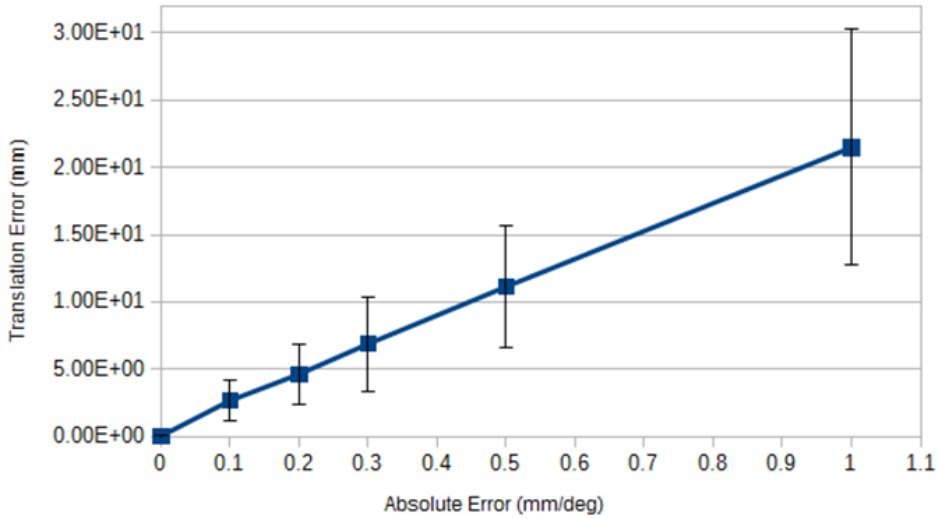
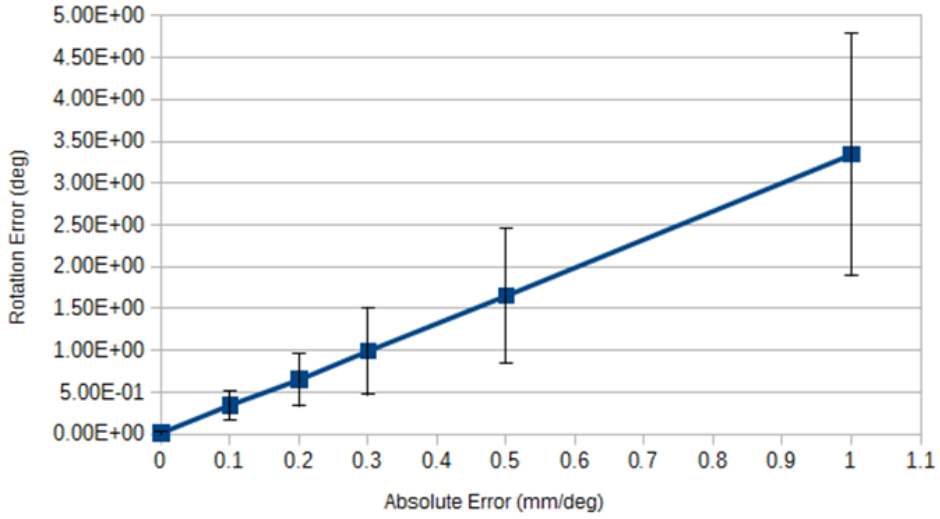


Figura 5.3: I grafici mostrano la precisione ed accuratezza delle misure ottenute in relazione al numero di spostamenti effettuati dal robot, in presenza di un errore maggiore rispetto a quanto visto in figura 5.1. Sull'asse Y è presente la media dei risultati, mostrando inoltre la deviazione standard dei risultati.



(a)



(b)

Figura 5.4: I grafici mostrano la precisione ed accuratezza delle misure ottenute in relazione all'errore assoluto delle misurazioni dei dati di partenza. Si nota chiaramente un andamento lineare, con un graduale peggioramento di precisione ed accuratezza all'aumentare dell'errore.

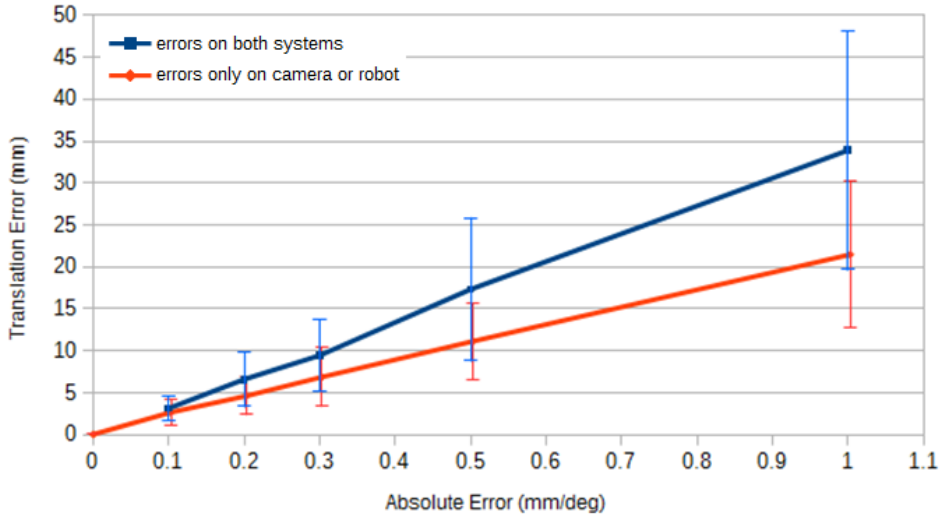
Test in presenza di errore su entrambi i sistemi

Nonostante il risultato sia uguale nel caso si applichi errore su un sistema o l'altro, è tuttavia interessante analizzare il comportamento del sistema in una simulazione più vicina a ciò che accade realmente, ovvero con diversi errori sui diversi sistemi, ed analizzare come diversi gradi di errore influiscano sul risultato finale.

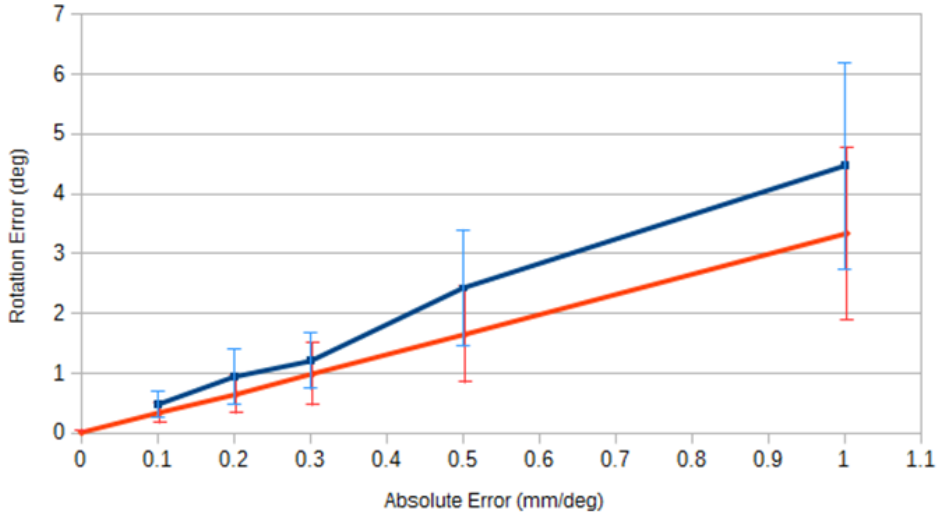
Per fare ciò si è utilizzato lo stesso criterio utilizzato nelle simulazioni precedenti, simulando un errore assoluto di n mm/n° come una funzione gaussiana di media nulla con $\sigma = n$ prima deviazione standard, ipotizzando però un errore diverso fra robot e camera (o viceversa) e quantificare quanto sia robusto il sistema a errori in entrambi.

In figura 5.5 sono presenti dei grafici che sintetizzano il risultato dell'algoritmo nel caso in cui l'errore sia presente sia sulla camera, sia sul robot, ed aumenti in entrambi con andamento lineare. Confrontandone l'andamento rispetto al caso in cui sia presente errore in un solo sistema. Si può notare come l'andamento sia del tutto simile a quanto accadeva nel test precedente, a parte un errore maggiore portato ovviamente dalla maggiore imprecisione dei dati.

Risulta interessante analizzare un caso più simile a quanto avviene nel mondo reale, nel quale il robot e la camera spesso non hanno lo stesso errore assoluto, bensì un errore diverso e dipendente alla qualità dei singoli sistemi indipendenti. Sono state pertanto effettuate una serie di simulazioni per mettere in relazione, nel caso in cui siano presenti differenti errori in uno dei due sistemi, il graduale aumento di imprecisione in un sistema con il degrado della soluzione finale. Sono state effettuate tre serie di simulazioni ponendo ponendo due diversi errori assoluti su di un sistema, ad esempio il robot, effettuando la stessa simulazioni per valori sempre crescenti di errore nel secondo sistema, ad esempio la camera, utilizzando 15 spostamenti. Come ci si aspettava, e come si può dedurre dai grafici in figura 5.6 se ne ricava che minore è l'errore commesso e di conseguenza migliore è la soluzione trovata, tuttavia si nota che con un errore molto grande in un sistema le due soluzioni influiscono meno nella precisione della soluzione finale, ottenendo risultati simili. Tale conclusione non stupisce, in quanto non sorprende il fatto che, con un errore su



(a)



(b)

Figura 5.5: Il grafico confronta l'aumento di errore in traslazione (a) e in rotazione (b) nel caso in cui sia presente l'errore su un solo sistema, sia esso camera o robot, od entrambi. Si può notare come entrambi abbiano un andamento lineare, come già visto in figura 5.5, avendo tuttavia una maggiore imprecisione nel caso siano presenti errori di misurazione in entrambi i sistemi robot e camera.

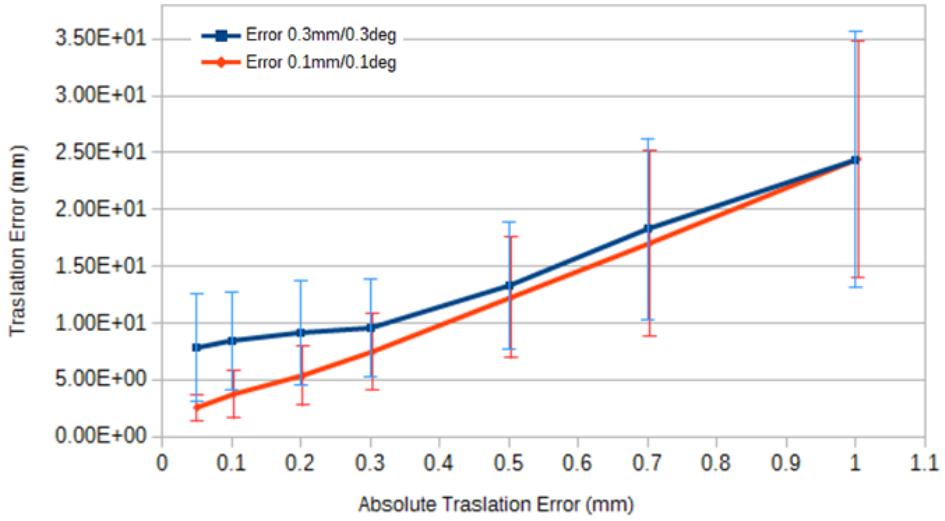
un sistema molto grande rispetto all'errore commesso sul secondo sistema, l'errore maggiore influisce in maniera predominante sulla precisione del sistema.

Se ne conclude che l'errore dato nella stima della calibrazione dipende principalmente dal massimo errore assoluto compiuto nell'acquisizione dei dati dei due sistemi, per cui avere ad esempio misurazioni quasi perfette sul robot risulta inutile nel caso in cui poi si abbia una scarsa precisione ed accuratezza nella misurazione della camera, o viceversa. Occorre pertanto porre grande attenzione nella precisione dell'acquisizione in entrambi i sistemi, cercando, per entrambi, di ridurre al minimo le incertezze, requisito fondamentale per ottenere una calibrazione di buona qualità.

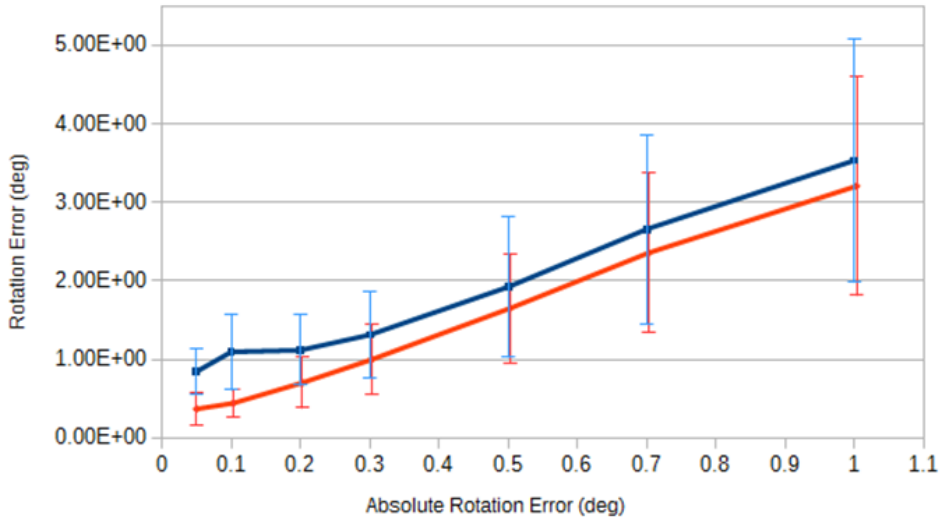
Test con diversi errore in traslazione e rotazione

Nei test fino ad ora effettuati si è stimato un incremento proporzionale fra errore di rotazione e di traslazione. Ci si pone ora il problema di quale trasformazione influisca maggiormente sulla qualità del risultato finale. Per fare ciò vengono effettuate delle simulazioni con diverse combinazioni di errori.

Si nota immediatamente dalla figura 5.7 come l'errore finale sia dato principalmente dalla componente rotatoria rispetto alla componente traslatoria. Occorre un errore di quasi 1 cm su ogni componente perché le serie inizino a convergere e a mostrare comportamenti simili, mentre con errori più simili a quanto misurato in ambiente reale, con errori nell'ordine di decimo di millimetro, la componente rotatoria è principale causa dell'inaccuratezza ed imprecisione del risultato, peggiorando in maniera significativa la soluzione con piccole variazioni di errore. Occorre perciò prestare particolare attenzione, in fase di acquisizione di dati reali, al variare il più possibile l'angolo fra uno spostamento ed il suo successivo in modo da minimizzare l'errore compiuto nella misurazione, in quanto errori molto piccoli si propagano in maniera significativa nell'errore finale. Si nota infatti come per errori nella parte traslatoria nell'ordine dei millimetri, fino a quasi 1 cm su ogni componente tridimensionale, si ottiene comunque una buona calibrazione, compatibile con le esigenze mostrate



(a)



(b)

Figura 5.6: Il grafico mette in relazione l'aumento di errore compiuto nella misurazione dei dati in un sistema, sia esso camera o robot, con l'aumento di imprecisione nella soluzione nella sua parte traslatoria (a) e rotatoria (b), ponendo un errore assoluto fisso sul secondo sistema. Si nota come il maggiore errore influisce maggiormente nell'errore compiuto nella soluzione.

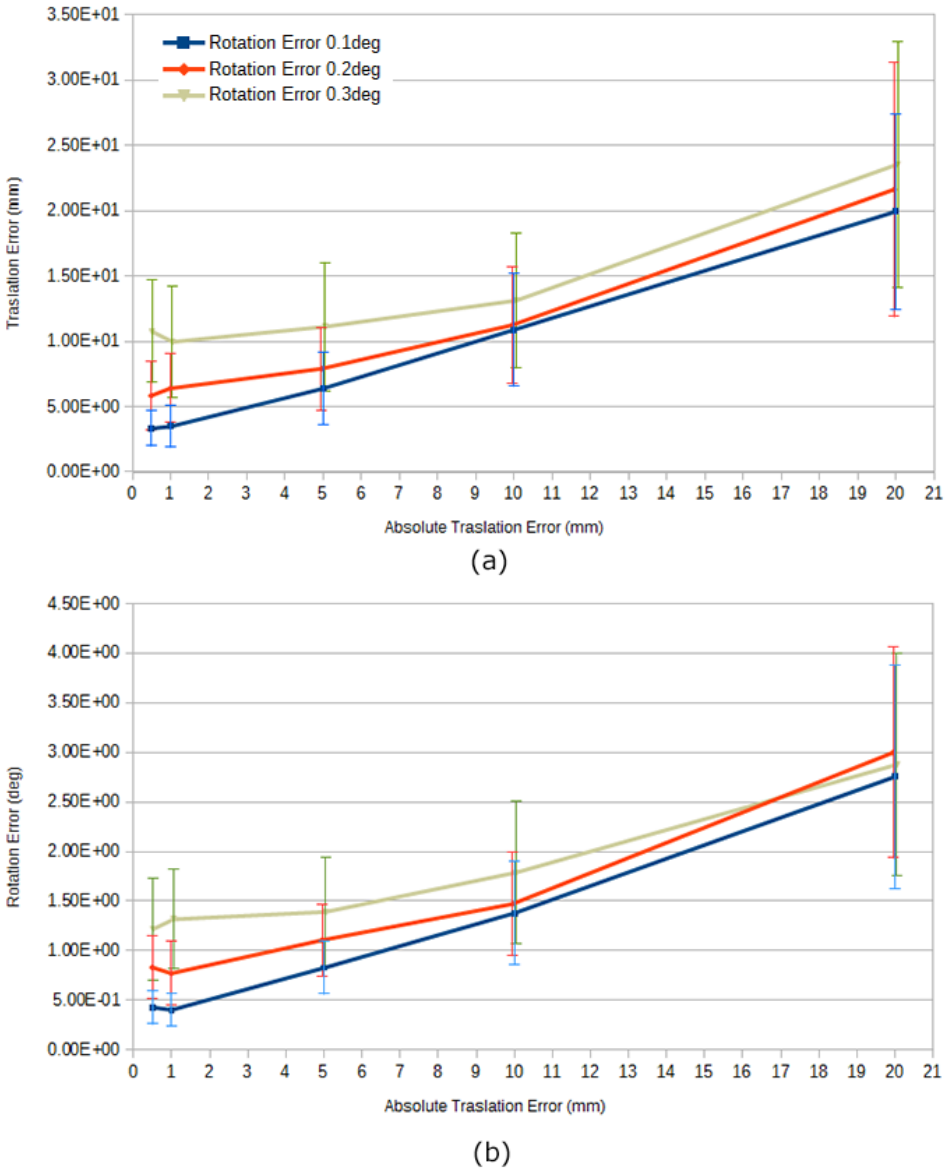


Figura 5.7: Il grafico mette in relazione il risultato della calibrazione, misurandone la differenza rispetto alla trasformazione reale nella sua componente traslatoria (a) e rotatoria (b), con l'errore assoluto commesso nella parte traslatoria. Sono graficate 3 diverse simulazioni soggette a diversi gradi di errore nella misurazione della parte rotatoria.

nei requisiti. Si dimostra inoltre che il sistema è robusto ad errori sistematici commessi dal robot nella componente traslatoria, in quanto errori, seppure sistematici, non influiscono in maniera significativa sul risultato, poiché nella parte traslatoria il sistema è resistente a dati con un errore di un ordine di grandezza superiore a quanto richiesta. Tuttavia sarà necessario sottolineare e prestare particolare attenzione alla componente rotatoria, la quale dovrà essere più precisa possibile, per garantire buoni risultati.

Capitolo 6

Conclusioni

É stato presentato un percorso che analizza nel complesso il problema di Hand Eye Calibration, ponendo attenzione, nello studio, alle necessità della realtà aziendale SpecialVideo. Il lavoro ha portato allo sviluppo di un pacchetto software fondato su quanto studiato dalle pubblicazioni scientifiche, realizzando, come richiesto dai requisiti forniti dall'azienda, una libreria facilmente integrabile al software aziendale in grado di calcolare con buona precisione l'Hand Eye Calibration di un braccio robotico, senza la necessità di avere un contatto fisico fra il robot ed un punto di riferimento nel piano di lavoro o di avere un operatore specializzato con il dovere di guidare, manualmente, il robot in determinate posizioni prestabilite. É stata inoltre effettuata una serie di test su dati simulati per provare la bontà del software e misurarne la precisione ed accuratezza, simulando il comportamento di un robot reale con diversi livelli di errore, ottenendo risultati soddisfacenti rispetto ai requisiti richiesti. La libreria è stata realizzata utilizzando il maggior numero di librerie esterne possibile, rendendo il codice il più semplice, modulare, efficiente e mantenibile possibile seguendo e rispettando comunque le specifiche e vincoli progettuali durante tutto il processo di sviluppo. Tale processo di sviluppo dovrà essere tuttavia un processo continuo nel tempo, in cui eventuali nuovi approcci risolutivi al problema, proposti sia dal lato accademico che da quello commerciale, dovranno essere tenuti in considerazione e studiati per reagire con efficacia ad eventuali nuovi bisogni aziendali, siano essi un cambiamento dei requisiti, del problema o

nuovi vincoli richiesti da diverse realtà operative, a causa del fatto che il problema è un problema attuale, giovane, e con nuove proposte che annualmente vengono presentate nei maggiori convegni e nelle maggiori pubblicazioni di robotica, allo scopo di migliorare e mantenere attuale nel tempo l'efficacia ed efficienza del software realizzato.

6.1 Lavori futuri

Il proseguo naturale del lavoro consiste nell'integrare l'algoritmo svolto all'interno del software prodotto dall'azienda SpecialVideo.

Per fare ciò occorre inizialmente effettuare una fase di testing su dati reali, anziché simulare tali informazioni, in modo da garantire una buona efficacia. Sarà possibile simulare, prima dell'acquisizione dei dati reali, le stesse pose sul simulatore sviluppato, in modo da garantire la funzionalità del software sulle posizioni scelte evitando errori intrinseci nell'applicazione al mondo reale quali imperfezioni del robot nel suo spostamento, difetti ottici della camera e dell'ottica utilizzata, problemi relativi all'illuminazione della scena o difetti strutturali dell'oggetto di calibrazione.

Testati gli algoritmi occorrerà rendere il più generale e semplice possibile l'utilizzo della libreria generata, creando una routine di calibrazione in grado di effettuare automaticamente non soltanto la calibrazione, ma anche l'acquisizione delle immagini e gli spostamenti del robot, limitando il più possibile la necessità di interventi attivi da parte dell'operatore.

Si potrà inoltre pensare di aggiungere un passaggio all'algoritmo di calibrazione per migliorare ulteriormente i risultati, utilizzando la soluzione ricavata con l'algoritmo implementato per generare un problema di minimizzazione e, sfruttando le librerie viste, come esempio Ceres, e gli studi effettuati sulla metrica del problema, come la pubblicazione Strobl e Hirzinger[24], cercare di migliorare la soluzione ricavata.

Per quanto riguarda una estensione del metodo sviluppato, è possibile pensare di generalizzare il metodo creato in modo da poter estendere il dominio applicativo dell'algoritmo non più solamente a telecamere matriciali, ma a telecamere di più vasto tipo, come ad esempio camere lineari o

matriciali, o a robot con diversi vincoli, ad esempio creando una calibrazione in grado di essere precisa utilizzando solamente movimenti planari e non più in tre dimensioni.

Tali problemi sono di grande interesse in campo robotico odierno in quanto sempre più si fa affidamento a robot per una moltitudine di task aziendali, siano essi controlli

6.2 Ringraziamenti

Alla fine di un lavoro di tesi sono doverosi ringraziamenti a tutti coloro che, in modo o nell'altro, mi hanno consentito di raggiungere questo traguardo, spingendomi, aiutandomi e sostenendomi in questo percorso che è finalmente giunto ad una conclusione.

Prima di tutto un ringraziamento doveroso ai docenti della seconda facoltà di Ingegneria dell'Università di Bologna, per i preziosi insegnamenti, la disponibilità, la puntualità e la voglia di rendere me ed i miei colleghi persone preparate per il mondo del lavoro, in particolare al prof. Bevilacqua, per il costante supporto ed aiuto durante lo svolgimento del lavoro di tesi.

A SpecialVideo srl, in particolare a Luca Tersi e Giuseppe Casadio per aver reso possibile il lavoro di tesi, mettendo a disposizione tempo e risorse per aiutarmi.

Un ringraziamento speciale ai miei familiari, che mi hanno sostenuto nell'arco di questo intero percorso, credendo in me e nei miei risultati ed appoggiando le mie scelte.

Grazie a Rachele, che più di tutti ha dovuto sopportare i momenti difficili dei quali, immancabilmente, è pieno qualsiasi percorso, universitario e non.

Grazie anche ai miei compagni di università ed ai soci dell'associazione S.P.R.I.Te di Cesena, che hanno accompagnato il cammino universitario fra mangiate, risate ed aiuti fra una lezione e l'altra ed agli amici, che fra giri in moto, partite a calcetto, pizzate e uscite al pub hanno aiutato a farmi staccare la spina e distrarmi dallo studio.

Grazie infine a tutti coloro che non ho nominato ma che, anche con un semplice in bocca al lupo prima di un esame, hanno reso possibile il compiersi di questo percorso.

Grazie.

Appendices

Appendice A

Quaternioni

Viene proposta in questo capitolo una breve introduzione ai quaternioni, largamente usati in computer vision, ed alla loro relazione con le rotazioni.

I vantaggi nell'utilizzo di un quaternione per rappresentare una rotazione tridimensionale sono molteplici, il che ne spiega il vasto utilizzo in vari campi, principalmente in campo informatico come la computer grafica tridimensionale[1] o la computer vision, anche se spesso vengono utilizzati in ambito puramente scientifico come in cristallografia[2].

Uno dei vantaggi più immediati dei quaternioni è la loro compattezza ed immediatezza dell'informazione, essendo infatti composto da solamente 4 numeri, al contrario di una matrice ortogonale, la quale è composta da 9 numeri; inoltre, dato un asse e un angolo, risulta immediato scrivere il quaternione corrispondente, operazione che al contrario risulta essere molto complicata se effettuata tramite matrici o angoli di Eulero.

Un altro vantaggio risiede nella composizione di varie rotazioni, per cui, nel calcolo automatico, vengono necessariamente effettuati degli arrotondamenti. Un quaternione leggermente sbagliato, a causa di arrotondamenti, risulta comunque essere una rotazione a seguito di una normalizzazione. Una matrice che, invece, perdere la proprietà di ortogonalità

[1] Soprattutto nel campo dei video games i quaternioni sono utilizzati nella quasi totalità delle volte, soprattutto in motori grafici moderni come OpenGL, Unity o VTK

[2] Scienza che si occupa dello studio dei cristalli, in particolare la loro formazione, crescita, struttura microscopica, aspetto macroscopico e proprietà fisiche.

a seguito di calcoli è molto più complicata da trattare affinché ritorni ad essere una matrice ortogonale compatibile con le rotazioni effettuate su di essa.

Un ultimo vantaggio dei quaternioni sugli angoli di Eulero, infine, consiste nell'inesistenza del problema del Gimbal Lock, problema che deve essere considerato invece in sistemi che si basano sulle informazioni di Pitch/Yaw/Roll, molto comuni soprattutto in ambito aereospaziale.

A.1 Algebra dei quaternioni

Un quaternione è definito come $q = w + xi + yj + zk$ con w, x, y e z numeri reali.

La operazioni di somma e sottrazione fra due quaternioni q_1 e q_2 sono definite come

$$q_0 \pm q_1 = (w_0 \pm w_1) + (x_0 \pm x_1)i + (y_0 \pm y_1)j + (z_0 \pm z_1)k.$$

La moltiplicazione degli elementi primitivi i, j e k è definita come $i^2 = j^2 = k^2 = -1, ij = -ji = k, jk = -kj = i$ e $ki - ik = j$.

La moltiplicazione fra due quaternioni è data conseguentemente da

$$\begin{aligned} q_0 q_1 =& (w_0 + x_0 i + y_0 j + z_0 k)(w_1 + x_1 i + y_1 j + z_1 k) \\ =& (w_0 w_1 - x_0 x_1 - y_0 y_1 - z_0 z_1) + \\ & (w_0 x_1 + x_0 w_1 + y_0 z_1 - z_0 y_1) i + \\ & (w_0 y_1 - x_0 z_1 + y_0 w_1 + z_0 x_1) j + \\ & (w_0 z_1 + x_0 y_1 - y_0 x_1 + z_0 w_1) k. \end{aligned}$$

e non è commutativa, da cui $q_0 q_1$ e $q_1 q_0$ non sempre è necessariamente equivalente.

Il coniugato di q è definito come

$$q^* = (w + xi + yj + zk)^* = w - xi - yj - zk.$$

per cui valgono le proprietà $(p^*)^* = p$ e $(pq)^* = q^*p^*$.

La norma di un quaternione è definita come

$$Norm(q) = Norm(w + xi + yj + zk) = w^2 + x^2 + y^2 + z^2.$$

La norma è un numero reale e la norma di un prodotto di quaternioni soddifa le proprietà

$$Norm(q^*) = Norm(q) \quad \text{e} \quad Norm(pq) = Norm(p)Norm(q).$$

L'operazione inversa di un quaternione q si denota con q^{-1} e gode della proprietà $qq^{-1} = q^{-1}q = 1$. Si costruisce come

$$q^{-1} = \frac{q^*}{Norm(q)}$$

dove l'operazione di divisione di un quaternione per un numero reale corrisponde alla divisione di ogni suo componente per tale valore.

L'operazione inversa gode delle proprietà $(p^{-1})^{-1} = p$ e $(pq)^{-1} = q^{-1}p^{-1}$.

Un quaternione può essere visto come la combinazione di un numero scalare w ed un vettore $\mathbb{R}^3$ $\vec{v}$, dove $\vec{v} = xi + yj + zk$.

Si può così esprimere la moltiplicazione fra quaternioni come prodotto scalare e vettoriale ottenendo

$$(w_0 + \vec{v_0})(w_1 + \vec{v_1}) = (w_0w_1 - \vec{v_0} \cdot \vec{v_1}) + w_0\vec{v_1} + w_1\vec{v_0} + \vec{v_0} \times \vec{v_1}.$$

da cui si ricava la proprietà per cui $q_0q_1 = q_1q_0$ se e solo se $\vec{v_0} \times \vec{v_1} = 0$, ovvero se i due vettori $\vec{v_0}$ e $\vec{v_1}$ sono paralleli.

Un quaternione può anche essere visto come un vettore $\mathbb{R}^4$ (w, x, y, z) per cui il prodotto scalare vale

$$q_0 \cdot q_1 = w_0w_1 + x_0x_1 + y_0y_1 + z_0z_1 = Real(q_0q_1).$$

dove $Real(q) = w$ ovvero la parte reale di un quaternione.

La moltiplicazione fra due quaternioni (denotata *) può essere definita come segue:

$$r * q = (r_0 + ir_x + jr_y + kr_z)(q_0 + iq_x + jq_y + kq_z)$$

può essere riscritta utilizzando una notazione matriciale:

$$r * q = Q(r)q = W(q)r \tag{A.1}$$

con

$$Q(r) = \begin{pmatrix} r_0 & -r_x & -r_y & -r_z \\ r_x & r_0 & -r_z & r_y \\ r_y & r_z & r_0 & -r_x \\ r_z & -r_y & r_x & r_0 \end{pmatrix}$$

e

$$W(r) = \begin{pmatrix} r_0 & -r_x & -r_y & -r_z \\ r_x & r_0 & r_z & -r_y \\ r_y & -r_z & r_0 & r_x \\ r_z & r_y & -r_x & r_0 \end{pmatrix}$$

Un quaternione unitario è un quaternione per cui $Norm(q) = 1$.

L'inversa di un quaternione unitario ed il prodotto di quaternioni unitari sono anch'essi quaternioni unitari. Un quaternione unitario può essere rappresentato come

$$q = \cos\theta + \vec{u}\sin\theta$$

con $\vec{u}$ vettore unitario. Si osserva che il prodotto $\vec{u}\vec{u} = -1$. Vale, per questa rappresentazione, una generalizzazione dell'identità di Eulero, ottenendo

$$exp(\vec{u}\theta) = \cos\theta + \vec{u}\sin\theta$$

da cui si ricava la formula per calcolare la potenza di un quaternione unitario

$$q^t = (\cos\theta + \vec{u}\sin\theta)^t = \exp(\vec{u}t\theta) = \cos(t\theta) + \vec{u}\sin(t\theta)$$

ed il logaritmo di un quaternione unitario

$$\log(q) = \log(\cos\theta + \vec{u}\sin\theta) = \log(\exp(\vec{u}\theta)) = \vec{u}\theta.$$

Tuttavia è importante ricordarsi che le operazioni fra quaternioni non sono commutative, per cui non valgono le proprietà $\exp(p)\exp(q) = \exp(p+q)$ e $\log(pq) = \log(p)\log(q)$.

A.2 Relazione fra rotazioni e quaternioni

Come si è appena visto un quaternione unitario può essere espresso come $q = \cos\theta + \vec{u}\sin\theta$, dove q rappresenta una rotazione di angolo 2θ attorno ad un asse $\vec{u}$.

Il vettore $\vec{v}'$ ruotato, risultato dell'operazione, si ottiene facilmente come

$$\vec{v}' = q\vec{v}q^*$$

Inoltre è possibile comporre due rotazioni rappresentate da due quaternioni q_1 e q_2 in un quaternione q' con la relazione

$$q' = q_2 q_1$$

nella quale q' rappresenta alla rotazione q_1 seguita dalla rotazioneq_2.

Tale proprietà viene dimostrata mostrando che $\vec{v}'$ è un vettore 3D, una funzione che non varia la lunghezza di $\vec{v}$, che è una trasformazione lineare e che non è una riflessione.

Per mostrare $\vec{v}' \in \mathbb{R}^e$

$$\begin{aligned} Real(\vec{v}') &= Real(q\vec{v}q^*) \\ &= \frac{(q\vec{v}q^*) + (q\vec{v}^*q^*)^*}{2} \\ &= q\frac{v + v^*}{2}q^* \\ &= qReal(\vec{v})q^* \\ &= Real(\vec{v}) \\ &= 0. \end{aligned}$$

Per dimostrare che la lunhezza di $\vec{v}$ rimane invariata

$$\begin{aligned} Norm(\vec{v}') &= Norm(q\vec{v}q^*) \\ &= Norm(q)Norm(\vec{v})Norm(q^*) \\ &= Norm(\vec{v}). \end{aligned}$$

Per dimostrare che $\vec{v}'$ è frutto di una trasformazione lineare

$$\begin{aligned} f(\alpha\vec{v} + \vec{w} &= q(\alpha\vec{v} + \vec{w})q^* \\ &= q\alpha\vec{v}q^* + q\vec{w}q^* \\ &= \alpha q\vec{v}q^* + q\vec{w}q^* \\ &= \alpha Real(\vec{v}) + Real(\vec{w}) \end{aligned}$$

da cui si dimostra che la trasformazione della combinazione lineare di vettori è la combinazione delle loro trasformazioni.

Da queste proprietà se ne ricava che $\vec{v}'$ è ortonormale, può essere pertanto una riflessione o una rotazione.

Tuttavia, pensando di mantenere fisso il vettore $\vec{v}$ si può dimostrare che la funzione $f(\vec{v}) = q\vec{v}q^*$ è una funzione continua di q, e che per ogni trasformazione lineare q la funzione determinante $D(q)$ è continua. Se ne deduce che $\lim_{q\to 1} f(q) = f(1) = I$ e $\lim_{q\to 1} D(q) = D(1) = 1$. Da cui si ottiene che la funzione rappresenta una rotazione e non una riflessione.

Per dimostrare che effettivamente il quaternione q rappresenta una rotazione θ su un asse $\vec{u}$ occorre dimostrare che per qualsiasi valore di θ $\vec{u}$ rimane invariato. Ricordando che $\vec{u}^2 = \vec{u}\vec{u} = -1$ si ricava che $\vec{u}^3 = -\vec{u}$

$$\begin{aligned} \vec{u}' &= q\vec{u}q^* \\ &= (\cos\theta + \vec{u}\sin\theta)\vec{u}(\cos\theta - \vec{u}\sin\theta) \\ &= (\cos\theta)^2\vec{u} - (\sin\theta)^2\vec{u}^3 \\ &= (\cos\theta)^2\vec{u} - (\sin\theta)^2 - \vec{u} \\ &= \vec{u}. \end{aligned}$$

Rimane infine da dimostrare che la rotazione attorno all'asse è 2θ. Si prenda la base ortonormale composta dai versori $\vec{u}$, $\vec{v}$ e $\vec{w}$, e si pensi di ruotare il vettore $\vec{v}$ di un angolo ϕ sul vettore $q\vec{v}q^*$, per cui $\vec{v} \cdot (q\vec{v}q^*) = \cos\phi$. Rappresentando il quaternione come vettore $\mathbb{R}^4$ e sfruttando la proprietà per cui $q_0 \cdot q_1 = Real(q_0 q_1)$

$$\begin{aligned}
\cos\phi &= \vec{v} \cdot (q\vec{v}q^*) \\
&= Real(\vec{v}^* q\vec{v}q^*) \\
&= Real(-\vec{v}(\cos\theta + \vec{u}\sin\theta)\vec{v}(\cos\theta - \vec{u}\sin\theta)) \\
&= Real((\cos\theta)^2 - (\sin\theta)^2 - \vec{u}(2\sin\theta\cos\theta)) \\
&= (\cos\theta)^2 - (\sin\theta)^2 \\
&= \cos(2\theta)
\end{aligned}$$

per cui l'angolo di rotazione risulta $\phi = 2\theta$ come previsto.

Appendice B

Numeri e Quaternioni duali

Numeri duali

I numeri duali sono introdotti da Clifford (1873): in maniera simile ai numeri complessi che consistono in una parte reale ed una parte complessa, i numeri duali sono composti da due parti, definite come:

$$\check{z} = r + \epsilon d \quad con \quad \epsilon^2 = 0$$

dove ϵ si chiama operatore duale ed è un operatore nilpotente, r è la parte reale e d la parte duale. In maniera simile ai numeri complessi, dove i identifica la parte immainaria, nei numeri duali ϵ identifica la parte duale. Il concetto di numero duale può essere esteso, definendo ad esempio vettori duali, ma soprattutto può essere esteso ai quaternioni per rappresentare trasformazioni spaziali, come rotazioni, traslazioni o una trasformazione generica.

Le operazioni fondamentali fra numeri duali, in breve, sono così risolte:

- Somma : $(r_a + d_a\epsilon) + (r_b + d_b\epsilon) = (r_a + r_b) + (d_a + d_b)\epsilon$.
- Moltiplicazione: $(r_a + d_a\epsilon)(r_b + d_b\epsilon) = r_a r_b + r_a d_b\epsilon + r_b d_a\epsilon + d_a d_b\epsilon^2 = r_a r_b + (r_a d_b + r_b d_a)\epsilon$.

- Divisione:

$$\frac{r_a + d_a\epsilon}{r_b + d_b\epsilon} = \frac{r_a + d_a\epsilon}{r_b + d_b\epsilon}\frac{r_b - d_b\epsilon}{r_b - d_b\epsilon}$$
$$= \frac{r_a r_b + (r_b d_a - r_a d_b)\epsilon}{r_b^2}$$
$$= \frac{r_a}{r_b} + \frac{r_b d_a - r_a d_b}{r_b^2}\epsilon.$$

I numeri duali possono essere rappresentati matricialmente come:

$$r + d\epsilon = \begin{pmatrix} r & d \\ 0 & r \end{pmatrix} \quad con \quad \epsilon = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

Quaternioni Duali

Il vantaggio dell'introduzione della teoria dei numeri duali consiste nella possibilità di integrare questi ultimi con i quaternioni, ottenendo i quaternioni duali, introdotti anch'essi da Clifford nel 1882[12]. Mentre i quaternioni, come visto in precedenza, possono rappresentare una rotazione, i quaternioni duali unitari possono rappresentare in maniera equivalente rotazioni e traslazioni.

I vantaggi in questa notazione sono molteplici:

- Non ambigui, ogni quaternione duale rappresenta esattamente una traformazione.
- Più efficienti e compatti, rispetto al calcolo fra matrici, essendo descritti da 8 parametri invece che 12.
- Rappresentazione unica per rotazione e traslazione.
- Rappresenta in maniera chiara ed immediata il concetto di Screw, con la quale rappresentare una trasformazione rigida.
- Semplicità ed immediatezza nel comporre trasformazioni.

Un quaternione duale consiste in due quaternioni, uno parte reale ed uno parte duale.

$$q = q_r + q_d\epsilon$$

Combinando l'algebra dei quaternioni con quella dei numeri duali si ottengono le operazioni aritmetiche per i quaternioni duali, brevemente elencate in seguito:

- Moltiplicazione scalare: $sq = sq_r + sq_d\epsilon$.
- Addizione: $q_1q_2 = q_{r1} + q_{r2} + (q_{d1} + q_{d2})\epsilon$.
- Moltiplicazione: $q_1q_2 = q_{r1}q_{r2} + (q_{r1}q_{d2} + q_{r2}qd_1)\epsilon$.
- Coniugato: $q^* = q_r^* + q_d^*\epsilon$.
- Valore assoluto: $\|q\| = qq^*$.
- Condizione unitaria: $\|q\| = 1$ e $q_r^*q_d + q_d^*q_r = 0$.

Un quaternione duale può rappresentare una qualsiasi trasformazione rigida, componendo il quaternione come

$$q_r = r$$
$$q_d = \frac{1}{2}tr$$

dove r è un quaternione unitario rappresentante la rotazione e t un quaternione che descrive la rotazione rappresentata dal vettore $t = (0, \vec{t})$.

Un quaternione duale può rappresentare una rotazione pura, come un quaternione, semplicemente potendo a zero la parte duale.

$$q_r = [\cos\frac{\theta}{2}, n_x \sin\frac{\theta}{2}, n_y \sin\frac{\theta}{2}, n_z \sin\frac{\theta}{2}][0,0,0,0]$$

Un quaternione duale può anche rappresentare una traslazione pura, nel caso in cui la parte reale sia un'identità, in cui la parte duale rappresenta la traslazione

$$q_t = [1,0,0,0][0, \frac{t_x}{2}, \frac{t_y}{2}, \frac{t_z}{2}]$$

Combinando la parte rotazionale e traslazionale in un singolo quaternione unitario $q = q_t \times q_r$ si ottiene il metodo per trasformare un punto p, espresso sotto forma di quaternione, secondo la trasformazione q.

$$p' = qpq^*$$

Relazione fra quaternione duale e matrice di trasformazione

Un quaternione duale unitario inoltre rappresenta il concetto di Screw Axis, in cui una trasformazione viene rappresentata come una rotazione attorno ad un asse ed una successiva traslazione sull'asse stesso. Un quaternione duale può anche essere scritto come

$$q = \cos\frac{\theta}{2} + s\sin\frac{\theta}{2}$$

dove s è l'angolo duale $\theta = \theta_r + \epsilon\theta_d$ e s è il vettore duale $s = s_r + \epsilon r_d$.

In questa rappresentazione ogni valore ha un diverso significato geometrico all'interno della trasformazione:

- s_r rappresenta la direzione dell'asse di rotazione.
- s_d rappresenta in maniera non ambigua la posizione spaziale dell'asse di rotazione nello spazio, data dal momento dell'asse $s_d = s_r \times p$ con p vettore dall'origine ad un punto qualsiasi dell'asse.
- θ_r rappresenta la rotazione attorno all'asse di rotazione.
- θ_d rappresenta la traslazione sull'asse di rotazione.

Questa rappresentazione risulta molto efficace nel momento in cui sia necessario confrontare trasformazioni fra loro, come ad esempio nel caso del metodo di Hand Eye Calibration proposto da Daniliidis[13], il quale sfrutta proprio questa rappresentazione per costruire una formulazione semplice ed immediata per il calcolo della calibrazione.

Bibliografia

[1] Bsd license. http://www.linfo.org/bsdlicense.html.

[2] Ceres solver library. http://ceres-solver.org/.

[3] Eigen c++ library v. 3.2.7. http://eigen.tuxfamily.org/.

[4] Google commandline flags. http://gflags.github.io/gflags/.

[5] Google logging module. https://github.com/google/glog.

[6] Mozilla public license version 2.0. https://www.mozilla.org/en-US/MPL/2.0/.

[7] Opencv library. http://opencv.org/.

[8] Suitesparse : A suite of sparse matrix software. http://faculty.cse.tamu.edu/davis/suitesparse.html.

[9] Gary Bradsky and Adrian Kaehler. *Learning OpenCV - Computer Visin with the OpenCV Library*. O'Reilly, first edition, 2008.

[10] H. Chen. A screw-motion approach to uniqueness analysis of head-eye geometry. *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR '91., IEEE Computer Society Conference*, pages 145–151, 1991.

[11] J. C. K. Chou and M. Kamel. Finding the position and orientation of a sensor on a robot manipulator using quaternions. *IJRR*, 10(3):240–254, 1991.

[12] W. Clifford. Mathematical papers. *London: Macmillan*, 1882.

[13] Konstantinos Daniilidis. Hand-eye calibration using dual quaternions. *IJRR*, 18:3497–3503, 1998.

[14] David Eberly. Quaternion algebra and calculus. *Geometric Tools, LLC*.

[15] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in computer vision*. Cambridge University Press, second edition, 2004.

[16] Richard I. Hartley and Fredrik Kahl. Global optimization through rotation space search. *IJCV*, 82(1):64–79, 2009.

[17] J. Heller, M. Havlena, A. Sugimoto, , and T. Pajdla. Structure-frommotion based hand-eye calibration using l_∞ minimization. *CVPR*, pages 3497–3503, 2011.

[18] Jan Heller, Michal Havlena, and Tomas Pajdla. A branch-and-bound algorithm for globally optimal hand-eye calibration. *CVPR*, pages 1608–1615, 2012.

[19] Jan Heller, Michal Havlena, and Tomas Pajdla. Globally optimal hand-eye calibration using branch-and-bound. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP:1, 2015.

[20] Radu Horaud and Fadi Dornaika. Hand-eye calibration. *IJRR*, 14(3):195–210, 1995.

[21] Ben Kenwright. A beginners guide to dual-quaternions: What they are, how they work, and how to use them for 3d. *Character Hierarchies, The 20th International Conference on Computer Graphics, Visualization and Computer Vision, WSCG 2012 Communication Proceedings*, pages 1–13.

[22] F. C. Park and B. J. Martin. Robot sensor calibration: solving ax=xb on the euclidean group. *EEE Transactions of Robotics and Automation*, pages 717–721, 1994.

[23] Y. Shiu and S. Ahmad. Calibration of wrist-mounted robotic sensors by solving homogeneous transform equations of the form ax=xb. *IEEE Transactions on Robotics and Automation*, 5(1):16–29, 1989.

[24] Klaus H. Strobl and Gerd Hirzinger. Optimal hand-eye calibration. *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference*, pages 4647–4653, 2006.

[25] Roger Y. Tsai and Reimar K. Lenz. Real time versatile robotics hand/eye calibration using 3d machine vision. *In International Conference on Robotics and Automation*, 1:54–561, 1988.

[26] Roger Y. Tsai and Reimar K. Lenz. A new technique for fully autonomous and efficient 3d robotics hand/eye calibration. *IEEE Transactions on Robotics and Automation*, 5(3):345–358, 1989.

[27] Z. Zhao. Hand/eye calibration using convex optimization. *ICRA*, pages 2947–2952, 2011.