Università degli Studi di Padova

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in
Ingegneria Informatica

Calibrazione di una rete di telecamere all'infrarosso usando marker retroriflettenti

Relatore:
Prof. Emanuele Menegatti

Correlatore:
Dott. Mattia Guidolin

Laureando:
Thomas Porro
1237030

Anno Accademico 2021/2022
14 Aprile 2022

Abstract

I sistemi di telecamere sono una tecnologia che viene utilizzata sempre più spesso in svariati settori come ad esempio quello sportivo, medico e dell'intrattenimento; per questo motivo la loro calibrazione è un tema estremamente importante da affrontare.

Per questa procedura la letteratura attuale utilizza un numero elevato di strumenti spesso costosi e dispendiosi in termini di costruzione e spazio, come ad esempio la realizzazione di pattern a scacchiera che raggiungono dimensioni ragguardevoli (1-2 m). L'obiettivo della sperimentazione presentata in questo elaborato è quello di calibrare efficacemente e in tempo reale una rete di telecamere all'infrarosso, semplificando la procedura rispetto le attuali proposte, sfruttando una bacchetta, con una struttura a T, sulla quale sono stati posti dei marker sferici retroriflettenti. Tale strategia di calibrazione prende ispirazione dai sistemi optoelettronici, tecnologia che permette un preciso rilevamento tridimensionale di specifici marker tramite la riflessione della luce. Difatti, le maggiori aziende del settore attuano una calibrazione tramite delle bacchette con dei marker.

Per sviluppare la nuova tipologia di calibrazione, il lavoro di questa tesi è stato suddiviso in due fasi: la prima consiste in uno studio del problema e delle attuali sue soluzioni più efficaci, la seconda nell'implementazione della nuova metodologia di calibrazione. Il risultato ottenuto da questo studio è lontano da essere paragonabile all'attuale stato dell'arte ma pone importanti basi per svincolarsi dai classici metodi di calibrazione; per questo motivo ne vengono analizzate anche le cause e proposte valide soluzioni sulle quali è possibile svolgere ulteriori studi in un prossimo futuro.

Ringraziamenti

Vorrei dedicare questo spazio a tutte quelle persone che nel corso di questi anni mi hanno sostenuto in questo percorso di studi.

In primis, ringrazio il Prof. Menegatti Emanuele che mi ha permesso di intraprendere il percorso di questa tesi nell'ambito della Computer Vision.

Un sentito ringraziamento anche al mio correlatore il Dott. Guidolin Mattia per il supporto costante fornitomi durante l'intera tesi.

Ringrazio i miei genitori, Olga e Christian, che in tutti questi anni hanno sempre sostenuto le mie scelte anche nei momenti più difficili.

Ringrazio mia nonna Isabella che, sebbene le numerose liti, non mi ha mai fatto mancare nulla.

Ringrazio Sara e Michele per il supporto morale nelle numerose occasioni di sconforto.

Dedico questo traguardo anche ai miei amici e ai miei compagni di corso, le persone con cui ho condiviso questo lungo percorso e che mi sono state vicine nonostante tutto.

Un grazie speciale a Matilde, la persona che mi ha sopportato di più. Mi ha consolato quando ero triste e mi ha dato la forza quando non ne avevo.

Padova, Aprile 2022 Thomas Porro

Indice

Capitolo 1

Introduzione

I sistemi di telecamere sono una tecnologia che viene sfruttata in ambiti svariati: militare e di sorveglianza [51], dell'intrattenimento [16], sportivo [32], medico [26] e naturalmente tecnologico, informatico, digitale e robotico [20]. L'ambito tuttavia dove questi sistemi hanno la loro implementazione massima è quello della computer grafica: questo perché grazie a una rete di telecamere calibrate è possibile ottenere delle rappresentazioni fedeli della figura umana che possono essere così trasportate fedelmente nella loro corrispettiva versione digitale. Ciò permette di registrare il movimento non solo di persone ma anche di oggetti e animali in uno spazio tridimensionale garantendo un'analisi precisa della scena che si sta riprendendo. Ad esempio Roy *et al.* è riuscito a tracciare il veloce movimento delle vibrisse dei topi con un'elevata precisione (< 0.5 mm) [40].

Come anticipato, la Motion Capture[1] viene utilizzata anche in ambiti non strettamente scientifici, come ad esempio nel settore videoludico e in quello della realtà virtuale. Questa espansione ha portato a richiedere metodologie di calibrazione sempre più veloci, economiche e precise.

Per creare un modello 3D altamente fedele, è necessario che la cattura di queste immagini avvenga tramite un sistema di telecamere perfettamente calibrato, ovvero che la posa relativa di ciascuna videocamera sia perfettamente conosciuta dal programma in uso.

Il corrente stato dell'arte per la calibrazione di questi tipi di sistemi, sebbene sia estremamente preciso e affidabile, richiede l'utilizzo di scacchiere che devono essere posizionate all'interno del campo visivo della rete di telecamere [57]. La realizzazione di queste scacchiere può rappresentare un problema a causa delle loro dimensioni: per poter essere rilevate in modo corretto a media distanza (5-6 m), infatti, devono essere di grandezze ragguardevoli (sull'ordine dei 1-2 m). Ciò le rende ingombranti e di difficile costruzione.

[1] Il processo di registrare il movimento di oggetti e/o persone.

Per ovviare a questo problema in questo elaborato la calibrazione di riferimento (ovvero la quale con cui si è svolto il confronto con il metodo sviluppato nel corso della tesi) è stata eseguita con degli AprilTag[2], poiché quest'ultimi sono di dimensioni ridotte ma hanno una precisione comparabile a quella delle scacchiere.

Si è preso spunto dall'articolo di Jackson *et al.* [19], il quale per ovviare alla difficoltà di utilizzo dei sistemi di telecamere da parte di molti scienziati a causa dell'elevato costo della strumentazione (scacchiere e telecamere), dimostra, grazie ad uno studio di tracciamento del comportamento animale, come sia possibile utilizzare a tale scopo delle telecamere a basso costo ottenendo comunque dei risultati attendibili.

Sulla scia di Jackson *et al.*, l'obiettivo di questa tesi è realizzare un nuovo metodo di calibrazione che abbia un costo relativamente basso e un facile utilizzo. Tuttavia questo elaborato sfrutta un differente approccio alla calibrazione, ovvero un metodo basato sull'uso di un bacchetta, con struttura a T, sulla quale sono applicati dei marker retroriflettenti permesso dall'uso di telecamera all'infrarosso, emulando così il processo di calibrazione che avviene all'interno dei sistemi optoelettronici, considerati come punto di riferimento per la Motion Capture all'interno della letteratura [11]. L'utente, tenendo in mano la bacchetta e muovendola liberamente all'interno della scena, consente all'algoritmo sviluppato di registrare i movimenti dei marker (e, di conseguenza, la posa della bacchetta) permettendo la calibrazione del sistema in tempo reale. L'utilizzo della sola frequenza dell'infrarosso permette l'applicazione del procedimento su una vasta gamma di telecamere.

1.1 Sistemi optoelettronici

I sistemi optoelettronici rappresentano l'attuale stato dell'arte nell'ambito della Motion Capture. Questa particolare tipologia di sistemi è composta da telecamere che contengono sensori in grado di percepire la luce infrarossa, proprietà che viene sfruttata come base di partenza per ottenere la posizione tridimensionale di un marker attraverso la triangolazione della sua posizione.

L'accuratezza di questi sistemi di sensori dipende da molti fattori, compresi fattori esterni al sistema stesso come per esempio la posa delle telecamere e l'ambiente stesso [46].

La posa delle telecamere dell'intero sistema deve essere perfettamente conosciuta, e tale obiettivo si può raggiungere attraverso una procedura di calibrazione molto precisa. Tale processo necessita di una grande quantità di dati: questi vengono acquisiti durante un periodo di tempo variabile (60-120 s) in cui delle apposite bacchette, sulle quali

[2]Approfondimento nei capitoli 2 e 5 assieme al confronto con l'algoritmo creato.

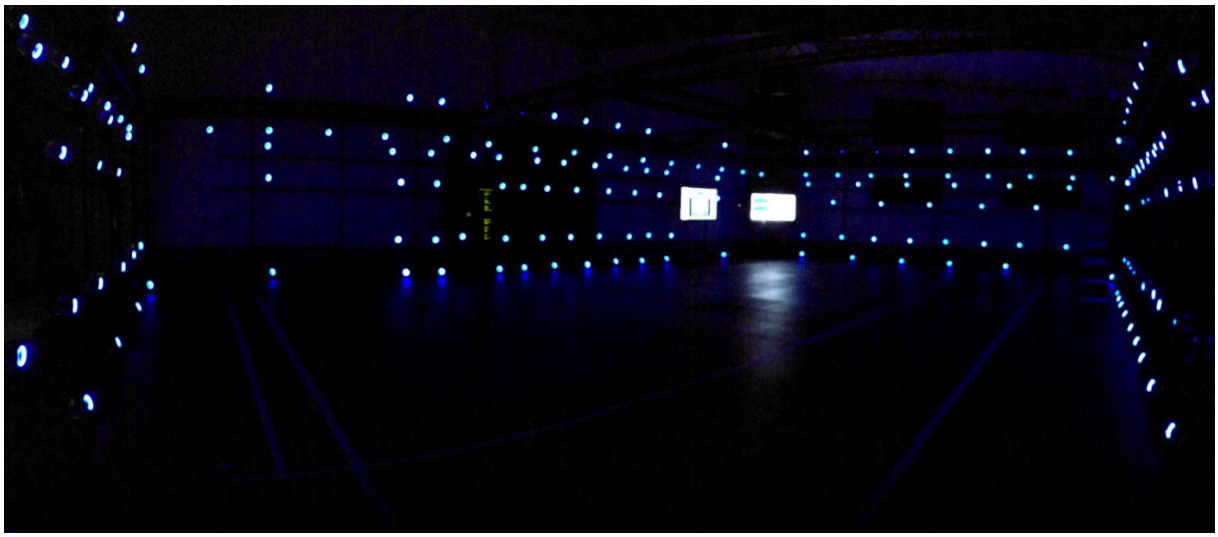


Figura 1.1: L'immagine mostra uno dei palchi di Activision Blizzard, azienda produttrice di videogiochi, per la cattura delle performance dei suoi attori. Quello in foto è il più grande del suo genere ed è in grado di rilevare 100 marker delle dimensioni di 3 mm alla distanza di circa 20 m [36]. Ogni cerchio luminoso visibile è una telecamera a infrarossi utilizzata per l'identificazione dei marker.

vengono posti dei marker passivi (in questo caso retroriflettenti) oppure attivi, vengono mosse ripetutamente all'interno della scena ripresa dal sistema. Nel caso in cui una telecamera venga spostata accidentalmente tale procedura deve essere ripetuta dall'inizio, permettendo così al sistema di essere nuovamente calibrato.

Un altro fattore che condiziona la stabilità dei sistemi optoelettronici è sicuramente l'ambiente di utilizzo. Le telecamere utilizzate sono infatti estremamente sensibili alla luce e dunque devono essere posizionate in laboratori dedicati per evitare che la presenza di riflessi causi una diminuzione eccessiva della precisione dei rilevamenti.

Nella figura 1.1 è possibile vedere un dei palchi di Activision Blizzard[3] [1] in cui vengono registrate le performance degli attori. Tale palco utilizza la tecnologia optoelettronica, ogni luce circolare presente in tale figura è una telecamera. L'utilizzo di un numero così elevato di dispositivi permette il rilevamento di circa 100 marker di 3 mm di diametro da una distanza di circa 20 m.

Nei sistemi optoelettronici i marker non servono solamente durante la calibrazione ma anche per il rilevamento dei movimenti, processo possibile grazie al fatto che sono attaccati al corpo da analizzare. I marker sono principalmente di due diverse tipologie: attivi e passivi. I marker passivi sono retroriflettenti e possono essere utilizzati solo quando le telecamere del sistema presentano un emettitore di raggi infrarossi. Sfruttando gli impulsi generati da esso è possibile triangolare, tramite la riflessione della luce, la posizione dei marker all'interno di un'area. I marker attivi invece non necessitano della presenza di un

[3] Azienda di videogiochi.

emettitore poiché gli impulsi infrarossi sono diffusi dai marker stessi. Queste tipologie di marker differiscono fra loro per due aspetti principali: precisione e praticità. L'utilizzo di marker attivi permette di avere una maggiore precisione, ma al tempo stesso il loro utilizzo limita la libertà di movimento poiché richiedono l'uso di cavi o di batterie per l'emissione dei raggi infrarossi.

L'applicazione dei sistemi optoelettronici trova grande utilizzo sia nel settore dell'intrattenimento, come videogiochi e film, che in ambito medico [8][9]. Nella medicina sta trovando grande spazio nell'individuazione di patologie che influenzano il movimento del corpo umano, come ad esempio il Parkinson, la sindrome di Down e la paralisi cerebrale [3].

La nuova metodologia di calibrazione introdotta all'interno di questa tesi prende ispirazione dalla procedura utilizzata nei sistemi optoelettronici, ma viene applicata in un contesto diverso grazie al fatto che è stato sviluppato un nuovo algoritmo che permette la calibrazione di un qualsiasi sistema telecamere all'infrarosso tramite l'uso di una bacchetta sulla quale sono posti dei marker retroriflettenti.

1.2 Sistemi markerless

Una delle più grandi sfide all'interno della Motion Capture è rappresentata dal tracciamento di persone e di oggetti senza l'utilizzo di strumentazione aggiuntiva (come possono essere dei marker retroriflettenti), ovvero *markerless*. La grande attenzione riservata a questo settore, molto rilevante in vari campi, deriva dalle numerose difficoltà nell'utilizzo dei sistemi optoelettronici dovute soprattutto alla necessità di agire in un ambiente controllato e all'uso di numerosi marker da parte del soggetto ripreso. In aggiunta a ciò, l'applicazione di marker su un corpo è un processo delicato e può richiedere un tempo eccessivo. Difatti anche piccoli errori nel loro posizionamento comporterebbe ad una errata ricostruzione dei movimenti del soggetto [10].

Per questo motivo nella letteratura sono innumerevoli gli studi che tentano di sviluppare nuove tecniche basate sia sull'elaborazione di immagini, con nuovi algoritmi, sia con l'utilizzo dell'intelligenza artificiale.

Nell'ambito medico si possono citare [10] e [43], lavori che presentano soluzioni diverse per il rilevamento del movimento del corpo umano. Il primo si basa sulla ricostruzione del movimento tramite un sistema di telecamere e un algoritmo probabilistico sfruttando un modello del corpo usato precedentemente conosciuto. Il secondo, invece, propone un metodo per l'identificazione e il tracciamento di un modello del corpo umano stimandone la struttura (riuscendo a ipotizzarne anche l'altezza ed il peso).

In ambito sportivo, Nakano *et al.* [31] per la Motion Capture markerless ha suggerito invece un metodo che utilizza un'intelligenza artificiale in grado di rilevare alcuni movimenti predefiniti, come il lancio di una palla, la camminata e un salto. Il livello di accuratezza raggiunto da questo algoritmo è notevole, ponendo l'80% dei rilevamenti sotto i 30 mm di errore.

1.3 Organizzazione della tesi

Il contenuto di questo elaborato è suddiviso nei seguenti capitoli:

Il secondo capitolo introduce il concetto di calibrazione intrinseca, presentando l'attuale standard utilizzato e le sue problematiche. Si pone poi attenzione all'articolo da cui si è preso ispirazione per la metodologia di calibrazione, ovvero una calibrazione a coppie seguita da un algoritmo di ottimizzazione. Il capitolo termina infine con la descrizione degli strumenti utilizzati per la realizzazione della calibrazione di riferimento per questa tesi e gli algoritmi sviluppati.

Il terzo capitolo descrive gli strumenti utilizzati nel corso della tesi, ovvero *OpenCV* [7] e *ROS* [38]. Entrambi i software sono introdotti da alcuni cenni storici che permettono di contestualizzare il loro sviluppo e la loro diffusione nei rispettivi ambiti. Vengono poi illustrate le numerose applicazione e funzionalità della libreria *OpenCV*. Viene infine approfondita la struttura interna di *ROS* per comprenderne al meglio il funzionamento.

Il quarto capitolo approfondisce il nuovo processo di calibrazione sviluppato in questo lavoro. Viene dapprima spiegata la strumentazione utilizzata all'interno della sperimentazione, ovvero le telecamere usate e la bacchetta usata come strumento di calibrazione (di cui è spiegato il processo di progettazione), e successivamente vengono introdotti gli algoritmi sviluppati per il riconoscimento dei marker e il calcolo della loro posa. Il nucleo del primo è incentrato sull'elaborazione delle immagini al fine di rilevare correttamente i marker, mentre il secondo permette il calcolo delle pose relative delle telecamere usate. Il capitolo termina infine con una spiegazione approfondita della struttura software del sistema usato durante la sperimentazione.

Il quinto capitolo presenta i risultati della calibrazione di riferimento, ovvero quelli realizzati tramite l'uso degli AprilTag, e successivamente è testata la nuova metodologia introdotta in questa tesi. I risultati delle due tipologie di calibrazione sono messi a confronto e sono riportate le principali differenze. I risultati ottenuti mostrano

come il metodo sviluppato in questa tesi non possa essere ancora utilizzato in un contesto reale, tuttavia tale procedura pone importanti basi per lo sviluppo di future ricerche.

Nel sesto capitolo vengono tratte le conclusione dell'elaborato. I risultati ottenuti dal confronto fra le due tipologie di calibrazione sono poi analizzati, al fine di estrarre le possibili cause di tali esiti. Fra di esse si possono notare delle limitazioni sia dal punto di vista hardware, come la risoluzione delle telecamere usate, che software, come l'accuratezza dell'algoritmo di stima dei centri dei marker. Questa analisi permette di porre le basi per future ricerche nell'ambito di questa tipologia di calibrazione.

Capitolo 2

Calibrazione ottima e lavori precedenti

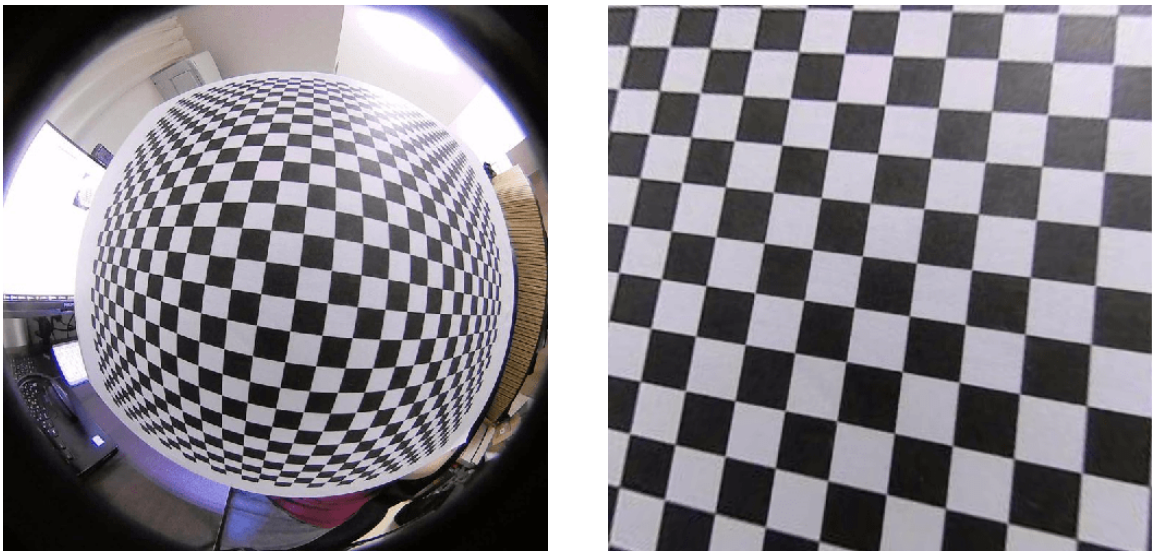
La calibrazione di un sistema di telecamere è un problema già affrontato precedentemente all'interno della letteratura scientifica, il che permette a questa tesi di avere una visione ampia sui vari aspetti da tenere in considerazione per una misurazione efficace. Per tale motivo, prima di introdurre la nuova metodologia proposta per la calibrazione, sarà analizzata la ricerca che ha costituito la base di partenza di questo elaborato.

Per avere una visione completa delle operazioni che si svolgeranno all'interno di questo elaborato è bene introdurre il concetto di calibrazione.

Con il termine calibrazione si intende effettuare la stima di tutti i parametri necessari per descrivere con accuratezza la relazione fra un punto 3D nel mondo reale e il suo corrispondente punto 2D in digitale. I parametri sono di due tipi: intrinseci ed estrinseci. I primi sono una serie di dati interni che riguardano la telecamera o le lenti di essa, di cui due esempi sono la lunghezza focale e il coefficiente di distorsione della lente. I parametri estrinseci invece sono parametri che esprimono la posa della camera (in particolare rotazione e traslazione) rispetto ad un certo sistema di riferimento (solitamente questo viene chiamato world o mondo).

In particolare la calibrazione intrinseca mira ad ottenere dei valori che permettono di correggere la distorsione provocata dalla lente della fotocamera, in questo modo si possono produrre delle immagini prive di errori sistematici[1]. La determinazione di tali valori è necessaria in quanto la costruzione delle telecamere a volte presenta degli errori causati dalla bassa qualità dell'assemblaggio oppure da delle imperfezioni presente nelle lenti stesse; tutte queste possibili imprecisioni provocano delle distorsioni nell'immagine ripresa; un esempio di questo può essere visto nella figura 2.1 in cui a sinistra mostra una foto in cui è fortemente presente la distorsione provocata dalla lente.

[1]Errori dovuti a imperfezioni dello strumento.



(a) Immagine originale

(b) Immagine priva di distorsioni

Figura 2.1: Nell'immagine a sinistra si può osservare un'immagine con una forte distorsione dovuta alla lente usata. A destra invece si può osservare l'immagine priva di distorsioni, ottenuta tramite il processi di calibrazione intrinseca.[25]

Come si può notare della figura 2.1b, parte della fotografia è stata ritagliata. Questo avviene a causa della calibrazione, infatti l'immagine presa in considerazione rappresenta la massima porzione della foto originale che può essere rettificata, ovvero riprodotta senza distorsioni. Questo esempio è un caso limite in quando l'immagine 2.1a è altamente distorta.

In questa tesi si assume che la calibrazione intrinseca delle telecamere utilizzate sia stata precedentemente eseguita e si presta quindi attenzione unicamente alla calibrazione estrinseca.

Per effettuare il calcolo dei parametri intrinseci ed estrinseci è necessario utilizzare degli oggetti o pattern (schemi) di riferimento di dimensioni conosciute e porli all'interno del campo visivo della telecamera. Il pattern più utilizzato in letteratura è quello della scacchiera: l'uso di questa presenta molti vantaggi. Infatti non solo gli angoli di ogni casella sono agevoli da rilevare (grazie alla forte differenza di colore fra una e l'altra) ma questi ultimi rappresentano anche i punti di intersezione fra le linee della scacchiera. L'insieme di questi elementi permette una localizzazione robusta degli incroci delle singole caselle.

Un esempio del risultato ottenuto dal rilevamento degli angoli di una scacchiera può essere visto nell'immagine 2.2 dove gli angoli sono stati correttamente identificati. La figura presenta inoltre un dettaglio interessante poiché suddivide visivamente le righe individuate.

La realizzazione di queste scacchiere, come anticipato nel capitolo 1, rappresenta

Figura 2.2: L'immagine mostra il processo di calibrazione effettuato con una scacchiera, di cui si sottolineano le notevoli dimensioni. Si può vedere chiaramente che gli angoli delle caselle sono stati correttamente identificati. [47]

molto spesso una difficoltà poiché devono essere realizzate con grande precisione. Difatti la qualità della calibrazione dipende da molti fattori come, ad esempio, il numero di caselle create. Un numero elevato di esse permette al software di calibrazione di ottenere maggiori informazioni, ottenendo così una migliore stima dei parametri intrinseci, ma al contempo un numero eccessivo porta ad una diminuzione della precisione; il pattern infatti, in presenza di un numero esagerato di dettagli, diventa confusionario e il rilevamento degli angoli risulta più arduo.

Altri fattori che potrebbero condizionare la calibrazione sono di natura fisica. Il processo richiede infatti che le linee della scacchiera siano perfettamente rette. Dunque, se la stampa della scacchiera presenta sbavature o se il materiale di costruzione ha subito una deformazione, la valutazione dei parametri intrinseci ed estrinseci potrebbe essere errata.

Oltre al pattern a scacchiera è possibile utilizzarne anche di altre tipologie. Le griglie di cerchi rappresentano una valida alternativa a quella proposta in questa sezione. Esse riprendono lo stesso concetto delle scacchiere poiché hanno delle caratteristiche ben identificabili che permettono la calibrazione, ovvero cerchi suddivisi in linee orizzontali e verticali.

2.1 Articolo di riferimento

In questa sezione si andrà ad evidenziare i procedimenti illustrati nell'articolo di Basso *et al.* [6] che affronta questo problema con concetti e terminologie che saranno poi ripresi e

approfonditi nell'elaborato in questione (capitolo 4).

Nell'articolo Basso *et al.* [6] si pone l'obiettivo di calcolare in tempo reale i parametri estrinseci, quali rotazione e traslazione, di una rete di telecamere mediante l'utilizzo di una scacchiera. Nel fare ciò riprende l'approccio più utilizzato nell'ambito scientifico (come riportato da Le *et al.* [22]) ovvero la suddivisione dei sensori a coppie in modo così che ogni paio possa poi essere calibrato indipendentemente. Questo, oltre che a facilitare la calibrazione, permette inoltre per ogni coppia l'utilizzo di diverse metodologie di calibrazione. In aggiunta alla metodologia citata, nel lavoro di Basso si fa inoltre uso di ROS[2], middleware che verrà ripreso anche all'interno di questo progetto (per approfondimenti vedi capitolo 3).

È necessario specificare che l'algoritmo sviluppato dall'autore può essere usato sia sulle comuni telecamere RGB[3] sia sui sensori di profondità. Con questa metodologia, i passaggi per la realizzazione della calibrazione sono:

- calibrazione grezza del sistema;
- creazione del grafico con le trasformate calcolate;
- ottimizzazione.

Basso per la calibrazione impiega le classiche scacchiere. L'uso di queste rappresenta un problema per tutti i sensori che utilizzano solo la profondità, in quanto gli angoli di esse sono visibili solo nello spettro RGB e diventa dunque impossibile rilevare tutti gli elementi necessari per la calibrazione. Per superare questo problema è stato utilizzato l'approccio descritto in Unnikrishnan *et al.* [45] che permette di introdurre ulteriori vincoli per l'identificazione della scacchiera. La differenza con il metodo descritto precedentemente risiede nella necessità di avere almeno tre istanze (ovvero immagini con al loro interno la scacchiera) di quest'ultima per procedere con la calibrazione.

La posa delle telecamere viene quindi calcolata partendo dal procedimento più comune, ovvero una calibrazione a coppie. Ottenute queste trasformazioni (matrici contenenti la pose relative delle telecamere) vengono poi utilizzate diverse funzioni[4] per permettere al sistema di ottimizzare la calibrazione generale finale.

I risultati ottenuti con l'approccio descritto nell'articolo di Basso presentano una buona precisione, nell'ordine dei millimetri, perciò è stata dunque ritenuta adeguata per fungere da base per la ricerca presente.

[2]Robot Operating System.
[3]Modello di colore basato sulla percezione umana.
[4]Ulteriori dettagli possono essere trovati in [6].

2.2 Calibrazione tramite AprilTag

La calibrazione mediante uso di AprilTag [50] sarà confrontata con la metodologia sviluppata nel corso di questo progetto.

Gli AprilTag, sviluppati dall'APRIL Robotics Laboratory dell'Università del Michigan, sono un sistema fiduciale visivo composto da oggetti posti nel campo visivo del sistema in uso come punti di riferimento o di misura. Questi dispositivi vengono già utilizzati nella calibrazione di telecamere e vantano un'ampia varietà di applicazioni come realtà aumentata e nella robotica [21]. Il vantaggio principale è che questi possono essere realizzati con una normale stampante.

Ovviamente anche l'utilizzo di questi presenta alcuni svantaggi. Per ottenere una calibrazione con un'alta precisione vi è la necessità di utilizzarne numerosi e inoltre devono essere tutti quanti all'interno del campo visivo delle telecamere. Inoltre, potendoli realizzare con una stampante da casa, vi è l'elevata possibilità di deformare i fogli e rendere impreciso l'intero processo di calibrazione.

Il software utilizzato per la calibrazione del sistema di telecamere tramite AprilTag permette di calcolare posizione, orientamento e identità dei singoli Tag nello spazio tridimensionale con un'alta precisione. Questo programma è implementato in C++, non presenta dipendenze esterne, ed è progettato per essere utilizzato in modo semplice su qualsiasi applicazione, anche su dispositivi mobile [21].

Si passa ora ad analizzare il metodo con cui la calibrazione tramite AprilTag è stata effettuata.

La procedura si svolge tramite telecamere che utilizzano sensori RGB. Tali modelli presentano un target leggermente diverso rispetto a quello delle telecamere utilizzate in questa analisi, ovvero ad infrarossi. Per questo motivo nel capitolo 5 si utilizzeranno Kinect [5], realizzate da Microsoft, per riuscire ad avere un confronto fra le diverse tipologie di telecamere[5].

Come nel lavoro Basso, il calcolo delle posizioni relative tra l'intero sistema di telecamere parte dal calcolo della calibrazione a coppie [6].

Come avviene con i processi classici "a scacchiera", anche in questo caso vi è il bisogno che gli AprilTag siano nel campo visivo di tutte le telecamere. Per rendere più efficace la calibrazione vengono usati numerosi Tag, tutti visibili contemporaneamente.

Il rilevamento di questi e il calcolo delle loro pose rispetto ai sistemi di riferimento di ogni telecamera sono effettuati tramite il pacchetto di ROS *apriltag_ros*[6], un wrap-

[5] Per approfondimenti vedi 4.1.1.

[6] `https://github.com/AprilRobotics/apriltag_ros`

per[7] del software sviluppato dalla APRIL che permette un accesso completo a tutte le personalizzazione dell'algoritmo, incluso appunto il rilevamento di un gruppo di tag [23].

Il processo di calibrazione con AprilTag avviene in due fasi:

1. l'acquisizione;
2. la calibrazione effettiva.

Durante la prima fase sono registrati e salvati una serie di dati utili al processo, ovvero le varie posizioni dei tag. Una volta posizionati all'interno della scena i tag, le loro pose sono rilevate dal pacchetto *apriltag_ros*; si avranno quindi la lista di tutte le etichette (tag) e le relative pose espresse rispetto al sistema di riferimento di ciascuna telecamera.

La seconda fase inizia con il calcolo della media fra tutte le pose acquisite di ogni AprilTag dalla fase precedente. Questo consente di ottenere una posa robusta per ogni etichetta in quanto l'eventuale rumore di fondo è attutito nella misura del valore intermedio delle posizioni rispetto a ciascuna telecamera.

Una volta ottenuti questi dati di riferimento ha inizio la effettiva fase di calibrazione. Le telecamere, suddivise due a due, analizzano le posizioni di ogni singolo tag. Successivamente viene calcolata, sempre per ogni singola etichetta, la matrice di trasformazione che corrisponde alla posa relativa tra le due telecamere. I valori ottenuti, iterando sul numero totale dei sistemi fiduciali visivi usati, dovrebbero risultare simili fra loro poiché le camere sono statiche e le loro posizioni non variano nel tempo.

Ottenute le matrici di trasformazione per tutti i tag e per tutte le coppie di telecamere, l'ultimo passo da eseguire è una media fra tutte le posizioni relative trovate: i risultati ottenuti rappresenteranno i valori del sistema calibrato.

2.3 Conclusioni

In questo capitolo è stata analizzata la differenza fra calibrazione intrinseca ed estrinseca. Mentre la prima determina parametri interni alla fotocamera, la seconda individua la sua posa rispetto ad un sistema di riferimento. In relazione a questo concetto sono state presentate le scacchiere e le loro caratteristiche, oggetti comunemente utilizzati durante il processo di calibrazione, ponendo attenzione ai dettagli rilevanti per la determinazione dei parametri intrinseci ed estrinseci.

L'articolo di Basso *et al.* è stato poi preso come riferimento per la strategia da utilizzare nello sviluppo della nuova metodologia di calibrazione, ovvero il processo di

[7]Particolare funzione che permette di utilizzarne una seconda senza particolare sforzo.

calibrazione a coppie e successiva ottimizzazione dei risultati. Oltre a questo articolo è stata specificata la procedura e gli algoritmi della calibrazione di riferimento di questa tesi, in particolare è stato fatto uso degli AprilTag, sistemi fiduciali visivi estremamente precisi, che hanno permesso di evitare alcune difficoltà provocate dalla costruzione di una scacchiera.

Capitolo 3

Strumenti utilizzati

Il processo di calibrazione realizzato in questa tesi si è basato sull'utilizzo di alcune delle librerie più utilizzate nel campo della robotica e della computer vision[1]. Prima di introdurle è necessario specificare che il linguaggio con cui è stato sviluppato il codice sorgente è C++. Questa scelta è basata su due fattori principali: in primo luogo tale linguaggio permette di creare un codice estremamente robusto con ottimi risultati per quanto riguardo affidabilità e velocità, in secondo luogo si è optata per questa scelta poiché è un linguaggio adottato da entrambe le librerie esterne utilizzate.

Le librerie utilizzate nel progetto sono due:

1. OpenCV, che rappresenta il nucleo per il processo di rilevamento dei marker all'interno di una scena (sezione 3.1);

2. ROS, che ha reso possibile la gestione dell'intero sistema di telecamere consentendo la comunicazione e controllo dei dati acquisiti durante la tesi (sezione 3.2).

I loghi delle due librerie sono visibili nella figura 3.1.

3.1 OpenCV

La Open Source Computer Vision Library (OpenCV) rappresenta il nucleo centrale della sperimentazione poiché attraverso il suo utilizzo è stato possibile identificare la posizione dei marker all'interno di un'immagine e successivamente calcolarne la posa.

OpenCV fu sviluppata in un progetto del 1999 su iniziativa di Intel[2] con l'obiettivo di progredire nell'ambito di applicazioni CPU[3] intensive, ovvero quelle applicazione che

[1]Disciplina scientifica che mira a capire e automatizzare le abilità del sistema visivo umano.

[2]Intel Corporation. Azienda americana leader nel settore dello sviluppo di processori per Personal Computer.

[3]Central Processing Unit. Conosciuto anche come processore.

permettono di sfruttare al massimo le potenza di calcolo dei processori dei computer. Lo sviluppo della libreria fu supportato da *Willow Garage*[4] prima e da *Itseez Inc.*[5] poi (azienda che il 25 maggio 2016 fu acquisita da Intel [33]). L'intero progetto è gestito dall'agosto 2012 dall'associazione non profit *OpenCV.org*.

La versione del software utilizzata in questa tesi è la 4.5.3. Con essa è stato realizzato l'intero algoritmo di rilevamento dei marker (capitolo 4), il quale sfrutta in particolare i moduli di OpenCV relativi alla elaborazione delle immagini e alla calibrazione 3D.

3.1.1 Funzionalità e utilizzi

La libreria in questione è un prodotto con licenza BSD, che permette quindi alle aziende di usufruire e modificare il codice in base alle loro esigenze. Ciò è possibile grazie al fatto che OpenCV presenta più di 2500 algoritmi disponibili, utilizzabili per identificazione di oggetti, classificazione di azioni umane nei video, tracciamento del movimento di oggetti, inseguimento del movimento delle pupille e molto altro. La community di coloro che contribuiscono a questa libreria comprende più di 47 mila persone e 18 milioni di download in totale [34].

Fra le aziende che fanno uso di OpenCV si possono citare *Google* [15], *Microsoft* [28], *Intel* [18], *IBM* [17], *Sony* [41] e *Honda* [2]. OpenCV è una risorsa però anche per gruppi di ricerca e governi.

Da parte dei governi, OpenCV serve sopratutto per monitorare gli eventi in luoghi pubblici: in Israele, per esempio, è utilizzata nei video di sorveglianza per rilevare eventuali intrusioni, in Cina invece supporta nella verifica della sicurezza delle attrezzature minerarie, in Europa ad abbassare il rischio di annegamento nelle piscine, e in Giappone per un'identificazione dei volti [34].

3.2 ROS – Robot Operating System

Il secondo strumento software utilizzato durante la calibrazione del sistema di telecamere è ROS.

Nonostante il nome sembri indicare un sistema operativo, in realtà: *"ROS è un insieme di librerie software e strumenti che ti aiutano a costruire applicazioni per robot"* [39]. Questa breve frase racchiude l'essenza di ROS, ovvero un insieme di framework per lo sviluppo di software dedicato ad applicazioni robotiche distribuite. L'intero progetto è

[4]Laboratorio di robotica e di ricerca orientato verso lo sviluppo di software open source.
[5]Azienda che sviluppa software per la computer vision personalizzati e realizza servizi di consulenza.

(a) OpenCV [34] **(b)** ROS [39]

Figura 3.1: Loghi delle due librerie utilizzate nel corso della tesi.

stato sviluppato seguendo la filosofia open source e dunque ne permette il libero utilizzo sia nell'ambito della ricerca che commerciale.

Questa middleware[6] nasce dall'idea di due dottorandi dell'Università di Stanford, Eric Berger e Keenan Wyrobek, che misero in luce alcune lacune nelle conoscenze tecniche da parte di sviluppatori software che non permettevano di utilizzare tutte le potenzialità degli hardware robotici.

Nel 2007 i due ricercatori si unirono alla neo-nata società *Willow Garage*, fondata da Scott Hassan, che nel novembre dello stesso anno caricò su SourceForge una primissima versione di ROS. In soli tre anni la società riuscì a realizzare un prototipo di robot in grado di spostarsi all'interno di un ufficio e aprire le porte. A seguito della pubblicazione della documentazione del lavoro compiuto, l'intero ecosistema creato cominciò a diffondersi rapidamente come standard all'interno della comunità robotica.

Nel 2012 la stessa Willow Garage fondò la *Open Source Robotic Foundation* (OSRF), che prese in gestione l'intero progetto ROS. Nel 2017 la OSFR cambiò nome in *Open-Robotics*, e nel contempo colossi della tecnologia come *Amazon* e *Microsoft* iniziarono a mostrare interesse verso la – ormai rimarchevole – raccolta di librerie: nel settembre 2018 l'azienda di Redmond inserì il supporto per ROS all'interno di Windows mentre nel novembre 2018 Amazon lo aggiunse al suo Web Services con il rilascio di Robomaker [4], che permette di estendere ROS con servizi cloud.

All'interno di questa tesi è stata utilizzata la versione ROS denominata Noetic. Questo software ha permesso il facile utilizzo e connessione delle telecamere durante gli esperimenti svolti. In particolare, uno dei i pacchetti dell'ecosistema utilizzati è stato *cv_bridge*[7], il quale ha consentito una perfetta integrazione della libreria OpenCV all'interno del codice sorgente (sezione 4.2.1).

[6] Con middleware si intende un software che fornisce alle applicazioni funzionalità e servizi comuni aggiuntivi.

[7] `https://github.com/ros-perception/vision_opencv`

3.2.1 Concetti fondamentali di ROS

Nella sezione precedente è stato specificato che, sebbene ROS sia l'anagramma di Robot Operating System, non si tratta propriamente di un sistema operativo ma più di un insieme di framework. L'intero sistema ROS è formato dunque da elementi più piccoli che cooperano fra di loro, tra i quali il più piccolo è il pacchetto (*package* in inglese), ma ricopre tuttavia un ruolo fondamentale perché permette di definire la modalità con cui il codice viene pubblicato. Solitamente a ogni pacchetto è assegnata una specifica operazione da svolgere. In questo modo il compito di ogni package è ben definito e il suo utilizzo e debugging[8] risulta più agevole.

Ognuno dei pacchetti, per funzionare al meglio, utilizza diverse componenti messe a disposizioni da ROS. Al fine di capire interamente dove le operazioni vengano compiute e come funzioni la comunicazione all'interno dei framework. Gli elementi di ROS che verranno spiegati sono: i nodi, i messaggi, i topic, le callback e le bag.

Nodi I nodi sono dei processi che eseguono delle azioni e sono gli elementi in cui vengono eseguite tutte le operazioni che sono svolte all'interno del programma. Essi sono combinati insieme in grafi[9] e comunicano fra di loro tramite i topic (vedi sezione 3.2.1) , i servizi[10] e parameter server (un dizionario condiviso fra i nodi tramite API[11]). L'obiettivo dei singoli nodi è quello di gestire l'intero sistema fino ai minimi dettagli. Per esempio nella gestione di un piccolo robot un nodo si occuperebbe del controllo delle ruote, un'altro calcolerebbe il percorso più adatto da seguire, e così via. L'utilizzo dei nodi in ROS porta molti vantaggi all'intero sistema a cui sono applicati:

- forniscono una maggiore tolleranza ai guasti: essendo i nodi indipendenti uno dall'altro un eventuale guasto andrebbe ad intaccare una singola unità e non il sistema intero;
- riducono la complessità del codice rispetto ad una struttura monolitica in cui la modularità è quasi assente;
- i dettagli dell'implementazione sono privati in quanto i nodi comunicano con il resto del grafo tramite una API estremamente limitata.

Solitamente per ogni pacchetto realizzato viene lanciato un solo nodo, in questo modo ogni nodo del sistema ha il compito di eseguire il codice di un singolo package [54].

[8] Indica l'attività di ricerca e correzione di errori all'interno del codice.
[9] Insieme di nodi.
[10] Modo di comunicare basato sul modello richiesta/risposta.
[11] Application Programming Interface. Permette di comunicare con altri computer o altre sezioni di codice.

Messaggi I messaggi sono gli strumenti tramite i quali i nodi si scambiano informazioni. Possono essere descritti semplicemente come una struttura dati. Sono supportati i tipi di dati primitivi (ovvero: *integer*, *floating point*, *boolean*, ...) e array di essi. Inoltre i messaggi possono contenere strutture nidificate [53].

Topic Il topic è il mezzo tramite il quale i nodi riescono a comunicare tra di loro: sono dunque dei bus[12] denominati attraverso i quali i nodi scambiano messaggi. Ogni topic può trasportare una sola tipologia di messaggi, ovvero possono trasportare solo delle predeterminate informazioni. I topic possiedono una politica di anonimità per i publisher/subscriber, termini che si possono tradurre con mittenti e destinatari. Per approfondire: il nodo 1 pubblica le informazioni in un determinato topic, se il nodo 2 vuole utilizzarle si deve iscrivere (ovvero ascoltare i messaggi che sono trasportati da un topic) al topic utilizzato dal nodo 1. L'intero scambio avviene senza condividere informazione su chi ha inviato i dati e chi li ha ricevuti, garantendo dunque l'anonimato. Una caratteristica da sottolineare è quella che per ognuno dei topic possono esserci svariati publisher e svariati subscriber.[55]

Callback Quando un nodo si iscrive ad un topic, quindi si mette ad ascoltare tutti i messaggi che arrivano su quel determinato bus, riceve dei dati che sono stati inviati da altri nodi del sistema. Quando uno di questi viene ricevuto, ROS permette di eseguire delle funzioni personalizzate (callback) che consentono di elaborare le informazioni ricevute e effettuare le operazioni necessarie come, ad esempio, una volta ricevuta un'immagine cercare all'interno di essa dei marker.

Bag Le bag sono uno strumento attraverso il quale ROS consente di salvare i dati ricevuti su uno o più topic; questo permette di replicare esattamente lo stato di un intero sistema in un qualsiasi momento. Esse sono usate da molti ricercatori per poter registrare, analizzare, visualizzare e salvare tutti i dati necessari per un uso futuro, anche a lungo termine. Ciò permette anche di lavorare in condizione "offline" ovvero in condizioni in cui non si ha diretto accesso alla strumentazione fondamentale per lo studio [52].

[12]Il bus è un sistema di comunicazione attraverso il quale vengono trasferiti dei dati all'interno del computer oppure tra computer diversi. In questo caso lo scambio di dati avviene fra nodi del sistema.

3.3 Conclusioni

In questo capitolo sono stati introdotti i due strumenti software che sono stati ampiamente utilizzati nel corso di questa tesi: OpenCV e ROS. Di OpenCV sono state presentate le varie funzionalità e la varietà dei suoi utilizzi, che spazia dalle aziende più famose ai governi. Di ROS sono stati spiegati i concetti fondamentali attraverso cui sono stati sviluppati gli algoritmi presentati in questa tesi, quali:

- nodi: gli elementi in cui vengono eseguite tutte le operazioni;
- messaggi: strumenti tramite i quali i nodi si scambiano informazioni;
- topic: mezzi tramite i quali i nodi si scambiano messaggi;
- callback: funzioni personalizzate che vengono eseguite quando un nodo riceve un messaggio;
- bag: strumento attraverso il quale ROS consente di salvare i dati ricevuti su uno o più topic.

Capitolo 4

Algoritmo di calibrazione proposto

Questo capitolo descrive in dettaglio il metodo sviluppato nel corso della tesi per calibrare reti di telecamere IR[1]. Si andranno dunque ad approfondire le strumentazioni utilizzate, con particolare attenzione alla progettazione e realizzazione della bacchetta impiegata. È inoltre presente una sezione che descrive i pacchetti creati e gli algoritmi utilizzati per il funzionamento del progetto (vedi sezione 4.2). A conclusione del capitolo sarà riassunto e specificato l'intero funzionamento della struttura del sistema.

4.1 Strumentazione

Una parte fondamentale della tesi è rivolta alla verifica dell'efficacia degli algoritmi sviluppati nel corso del progetto e a garantire la replicabilità dei risultati, pertanto si è ritenuto di fondamentale importanza approfondire in modo dettagliato la strumentazione utilizzata. In particolare in questa sottosezione verranno presentate minuziosamente le telecamere utilizzate e la bacchetta impiegata durante la calibrazione.

4.1.1 Telecamere utilizzate

Con l'obiettivo di poter creare un nuovo algoritmo valido per una vasta gamma di telecamere all'infrarosso si è deciso di utilizzare il modello più diffuso nell'ambito scientifico: le Kinect. Questa tipologia di telecamere è stata sviluppata da *Microsoft* per il suo dipartimento di videogiochi: la prima versione venne infatti rilasciata il 4 novembre 2010 per la console Xbox 360[2]. La caratteristica principale di questa nuova telecamera risiede nel permettere di utilizzare il corpo umano come controller[3] per la console di casa Redmond.

[1] Infrared. Ovvero lo spettro della luce infrarossa.
[2] Console sviluppata da Microsoft la cui prima versione è stata rilasciata nel Novembre 2007.
[3] Periferica attraverso la quale è possibile trasmettere comandi.

Per fare ciò, diverse furono le tecnologie introdotte da queste telecamere: oltre all'affidabile sensore RGB, in grado di ricevere immagini nello spettro dei colori visibili, sono stati aggiunti anche un sensore di profondità e un emettitore di raggi infrarossi. Negli anni successivi sono state messe in commercio versioni aggiornate delle videocamere, fino alla loro progressiva scomparsa dal settore videoludico a causa del loro scarso utilizzo da parte del pubblico. A differenza del settore dei videogiochi, l'ambito scientifico ha avuto modo di apprezzarne i sensori e il costo contenuto di questa tipologia di telecamere rendendone la diffusione sempre maggiore. Lo sviluppo è così proseguito fino ad oggi, la cui ultima versione disponibile è stata rilasciata nel corso del 2019: Azure Kinect [5].

Ed è proprio questa la versione utilizzata nella sperimentazione qui presentata.

Riguardo all'hardware, questa versione di telecamere presenta due sensori simili a quelli delle prime Kinect: uno RGB e uno per la profondità. Per il calcolo della profondità viene utilizzato un emettitore di raggi infrarossi che, tramite l'apposito sensore, consente la misurazione del tempo di volo della luce. Quest'ultimo è denominato anche *time-of-flight* e sfrutta degli impulsi di luce che, illuminando la scena per un breve periodo di tempo, emettono luce che viene poi riflessa dagli oggetti presenti. La telecamera, tramite l'apposito sensore, riesce a cogliere tali riflessi e, conoscendo la differenza di tempo fra la partenza degli impulsi e l'arrivo di essi, riesce a calcolare la distanza di un oggetto dal sensore con la seguente formula:

$$D = \frac{c \cdot t_D}{2} \tag{4.1}$$

dove D è la distanza dell'oggetto dal sensore, c è la velocità della luce e t_D è il tempo impiegato dall'impulso per ritornare alla telecamera.

Una volta ottenuto questo valore è possibile determinare la misura della distanza degli oggetti o delle persone dalla telecamera. Per la calibrazione di una rete di telecamere all'infrarosso, le uniche informazioni fornite delle Kinect effettivamente usate dall'algoritmo sviluppato nel corso di questa tesi sono state quelle relative alle immagini il cui spettro della luce è inferiore a quello del visibile, dati forniti dal sensore di profondità che ha la capacità di rilevare quelle determinate lunghezze d'onda.

Nel caso specifico di questa tesi è stato utilizzato il sensore di profondità nella modalità *"NFOV unbinned"* che permette di avere una immagine dalla risoluzione di 640 x 576 pixel con un FoI[4] di 75° x 65° e una frequenza di aggiornamento di 30 Hz[12]. L'integrazione con ROS e l'intero sistema è stata resa possibile grazie alla libreria *Azure_Kinect_ROS_Driver*[5][27], creata da Microsoft per permettere un utilizzo completo del

[4]Field of Interest, ovvero l'area che la telecamera permette di analizzare.

[5]`https://github.com/microsoft/Azure_Kinect_ROS_Driver`

Figura 4.1: Formazione circolare delle telecamere adottata durante la calibrazione. Al suo centro è possibile notare la bacchetta utilizzata.

suo hardware.

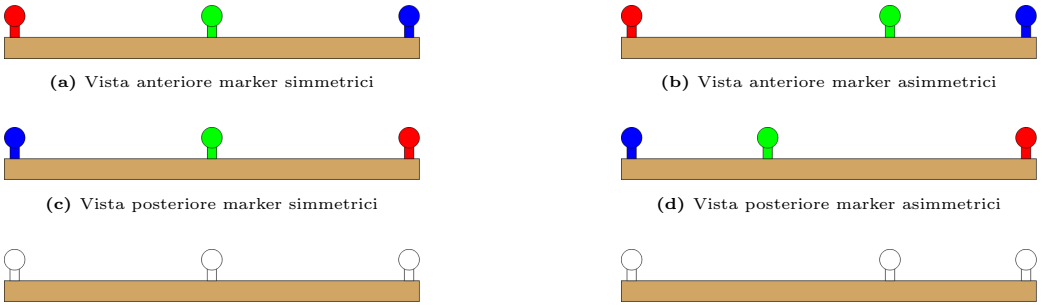
Per la calibrazione del sistema sono state utilizzate 5 Azure Kinect poste in una formazione circolare in maniera tale da permettere ad ognuna di vedere la stessa porzione di spazio in contemporanea.

4.1.2 Wand

Il secondo strumento cardine di questo progetto è la bacchetta ovvero wand. Essa è fondamentale per la sperimentazione poiché è l'elemento che permette di calcolare la posa reciproca delle Kinect.

La realizzazione della wand è stata centrale poiché dalla sua progettazione e strutturazione dipende interamente il calcolo della propria posa rispetto alla telecamera. Si è dunque proceduto svolgendo un'indagine per esaminare strumenti similari presenti nel mercato e valutando la configurazione più congeniale al progetto.

La prima wand osservata è stata quella proposta da *OptiTrack* [35]: una bacchetta, con struttura a T, che presenta dei marker passivi (quindi che non emettono luce propria) solo sull'asta orizzontale. I marker si posizionano in modo asimmetrico nel segmento perché altrimenti la capacità di ordinare i marker nella scena, e di conseguenza il calcolo della posa della bacchetta, risulterebbe impossibile da determinare. Per spiegare efficacemente l'affermazione precedente è riportato un piccolo esempio. Si immagini di avere 3 punti denominati a, b, c, posti alla stessa distanza fra di loro. Quando la telecamera riesce a vedere questi 3 marker nello stesso momento non riuscirà mai ad identificare la corretta posizione dei punti agli estremi del segmento poiché non riesce a percepire se essi sono



(a) Vista anteriore marker simmetrici

(b) Vista anteriore marker asimmetrici

(c) Vista posteriore marker simmetrici

(d) Vista posteriore marker asimmetrici

(e) Vista marker simmetrici senza colori

(f) Vista marker asimmetrici senza colori

Figura 4.2: L'immagine mostra le viste anteriori e posteriori di due segmenti in cui sono stati posti dei marker. Nella colonna a sinistra si può vedere il caso simmetrico (marker posti alla stessa distanze) mentre nella colonna a destra si può vedere il caso asimmetrico. L'immagine 4.2e mostra come sia impossibile riconoscere il colore dei marker agli estremi del segmento. In 4.2f invece il colore dei marker è conosciuto sebbene sia stato rimosso.

rilevati da un punto di vista anteriore o posteriore e quindi un punto esterno può essere nello stesso momento sia a che c. Per ulteriori dettagli vedere immagine 4.2.

L'utilizzo di questo design tuttavia non risulta in linea con lo scopo di questo elaborato poiché non fornisce informazioni accurate con sufficiente costanza; difatti, quando la bacchetta si posiziona perpendicolarmente all'asse ottico della telecamera, è impossibile ricavare la rotazione su di un asse. Nell'immagine 4.3 è presentato un esempio in cui la rotazione non può essere ricavata da tale modello di wand.

La seconda soluzione esaminata è quella proposta da *Vicon* [49]. La wand, anch'essa con una forma a T, presenta invece dei marker attivi (emettono una luce propria così da essere individuati senza bisogno di particolari emettitori di raggi infrarossi) non solo sulla parte orizzontale della bacchetta, ma anche su quella verticale. Questa scelta di progettazione permette di avere una precisione maggiore nell'individuazione della wand all'interno della scena perché va a risolvere il problema dell'impossibilità di ricavare la rotazione da un asse presentato nella wand di *OptiTrack*.

Nell'immagine 4.4 si possono vedere le differenze principali delle due soluzioni appena proposte.

Per lo sviluppo della tesi è stato scelto di procedere con una struttura simile a quella della wand utilizzata da Vicon, perciò con una struttura a T e con i marker posizionati sia sull'asta orizzontale sia su quella verticale che si differenzia da quella prodotta dall'azienda britannica sia per il il posizionamento che per la natura degli stessi marker (passivi).

Per la sua semplicità di utilizzo e la facile reperibilità, si è deciso di utilizzare il legno come materiale per la creazione delle aste del prototipo. Alle due aste sono stati aggiunti comuni marker retroriflettenti sferici il cui diametro è di 14 mm. Visto il proposito di rendere tutto il processo il più semplice possibile è stato deciso di rendere la wand di dimensioni contenute in modo da non essere di nessun ostacolo all'utente. Le sue

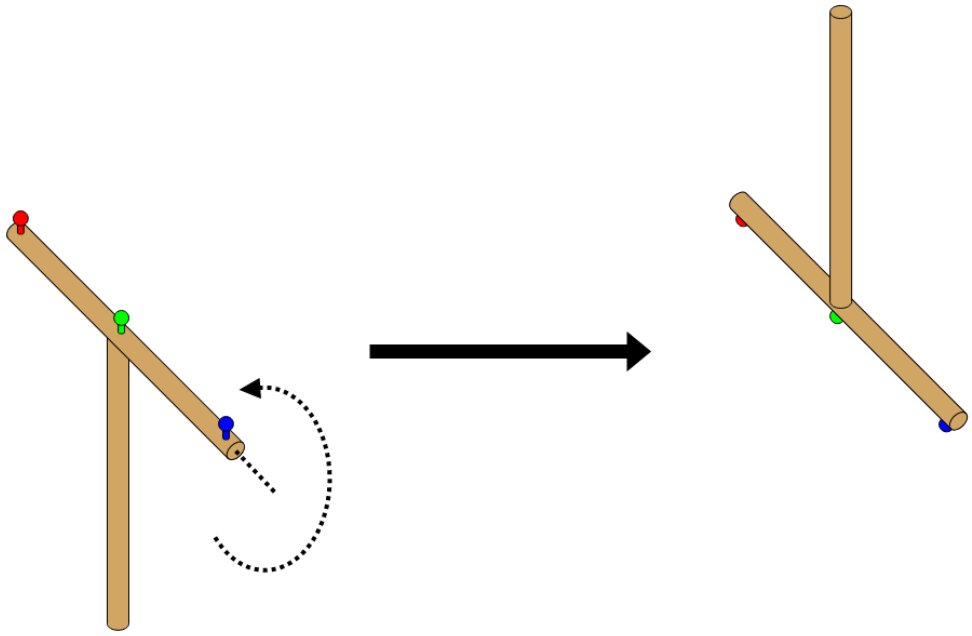


Figura 4.3: Questa immagine stilizzata mostra come, ruotando la bacchetta attorno il suo segmento orizzontale, la posizione dei marker rimanga invariata. In tal modo la determinazione della rotazione attorno all'asse orizzontale risulta impossibile.

dimensioni sono di circa 30 cm per l'asta orizzontale e circa 35 cm per l'asta verticale. Nell'immagine 4.5 si può notare la sua struttura e il posizionamento dei marker. La posizione di questi ultimi rispecchia le seguenti coordinate su assi cartesiani (ad ognuno dei marker è stata assegnata una lettera seguendo l'ordine da sinistra a destra e dall'alto in basso):

- Punto $A = (0.0, 0.25, 0.0)$;
- Punto $B = (0.2, 0.25, 0.0)$;
- Punto $C = (0.3, 0.25, 0.0)$;
- Punto $D = (0.2, 0.125, 0.0)$;
- Punto $E = (0.2, 0.0, 0.0)$;

le cui misure sono state fornite in metri. Una cosa importante è la complanarità dei punti ovvero il fatto che essi giacciono tutti sullo stesso piano, la cui importanza verrà approfondita in sezione 4.2.1. Per finire, a ciascun marker è stata attribuita un'etichetta, elemento fondamentale per poter riconoscere con precisione il marker individuato durante l'esecuzione degli algoritmi.

(a) Optitrack [42]

(b) Vicon [37]

Figura 4.4: Nell'immagine si possono notare le differenze fra le due bacchette presentate. 4.4a presenta marker passivi sul suo segmento orizzontale; 4.4b presenta marker attivi (i piccoli pallini bianchi) sia sul segmento orizzontale che su quello verticale.

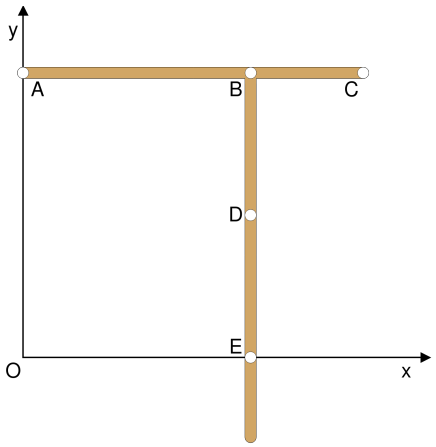


Figura 4.5: Nell'immagine è rappresentata la struttura della wand utilizzata in questa tesi. Sono presenti anche gli assi x e y; z invece presenta un verso uscente rispetto al foglio.

4.2 Algoritmi sviluppati

Dopo un approfondimento sulla strumentazione hardware utilizzata, si può proseguire ad un'analisi di tutte le componenti software e degli algoritmi usati.

Tutti gli argomenti necessari per comprendere appieno il sistema progettato sono stati approfonditi nella sezione 3.2.1. Si può quindi proseguire con la descrizione dei due algoritmi principali sviluppati all'interno di due pacchetti ROS creati durante la realizzazione di questa tesi, ovvero: *find markers* e *wand calibration*. Lo sviluppo e l'uso di pacchetti ROS hanno portato numerosi vantaggi all'intero sistema, come ad esempio la rapida comunicazione fra le telecamere, il software realizzato e l'alta tolleranza ai guasti grazie alla modularità dei *package*.

Il primo avrà come obiettivo principe il rilevamento dei marker all'interno dell'immagine IR, mentre il secondo avrà come scopo il calcolo delle pose reciproche delle telecamere grazie ai dati forniti dal primo *package*. Ognuno di essi creerà un nodo omonimo attraverso

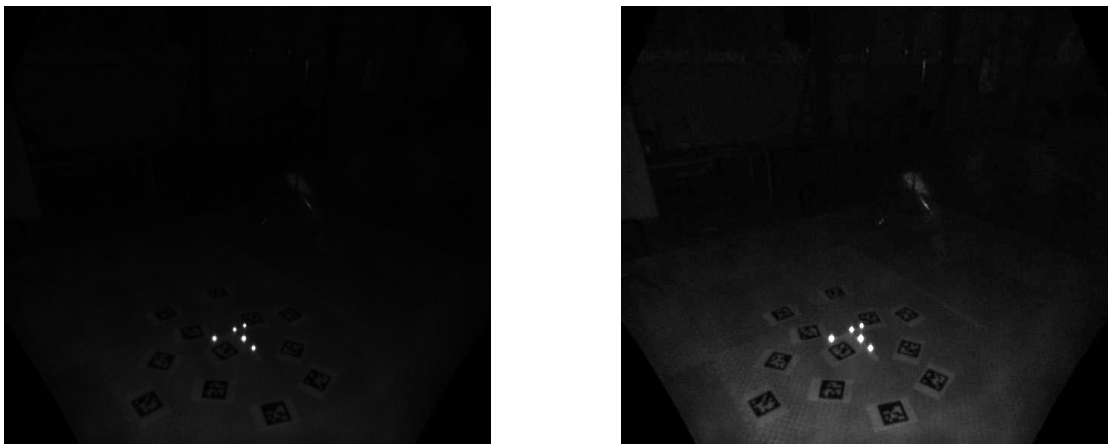
il quale il pacchetto effettuerà tutte le operazioni.

4.2.1 Find Markers

Nella sezione 4.2 è stato introdotto velocemente lo scopo di questo pacchetto, ovvero rilevare le posizioni dei centri dei marker presenti nelle immagini ottenute dalle Kinect. In realtà il package fa molto di più di questo poiché è di fatto il responsabile del calcolo della posa relativa della bacchetta rispetto alla videocamera: di seguito si prenderà in considerazione la modalità con cui questo avviene.

Verrà approfondito ora l'algoritmo attraverso il quale i marker vengono identificati all'interno delle immagini IR ottenute dalle telecamere. Prima di ogni altra cosa è indispensabile ricordare che all'interno dell'infrastruttura di ROS tutte le informazioni e i dati vengono trasportati grazie all'utilizzo dei topic. Questo avviene anche nel caso delle immagini. Il nodo creato da Find Markers effettua iscrizioni a due topic in particolare: quello che trasporta le immagini infrarosse provenienti dalla telecamera e quello che trasporta informazioni relative alla telecamera che sta riprendendo la scena. Come prerequisito per la calibrazione si richiede che le Kinect siano calibrate intrinsecamente: perché ciò avvenga, la matrice di distorsione, che ne contiene i parametri, e la matrice della telecamera devono essere conosciute. Queste informazioni sono pubblicate all'interno dell'apposito topic[6].

Per comprendere al meglio come opera questo nodo l'immagine 4.6a mostra ciò che viene ripreso da una delle kinect della rete utilizzata durante gli esperimenti. Come si può



(a) Immagine originale **(b)** Immagine modificata

Figura 4.6: Queste figure rappresentano le immagini IR riprese da una Azure Kinect. 4.6a rappresenta l'immagine effettivamente ripresa; a 4.6b ne è stato modificato il valore di esposizione di un valore 1.50 EV in modo da permettere al lettore una miglior comprensione. Posti sul pavimento, insieme alla wand, sono visibili anche gli AprilTag usati per la calibrazione di riferimento.

[6]Per approfondimento sui topic utilizzati vedi 4.3.

vedere i marker presenti sulla wand riflettono in modo efficace la luce infrarossa e infatti sono estremamente visibili all'interno della scena. Si sottolinea che la quasi totale assenza di rumore di fondo (vedi immagine 4.6) rende questo uno dei migliori scenari possibili per il rilevamento dei marker.

Per l'identificazione dei marker all'interno della scena è stato fatto uso della libreria OpenCV poiché essa permette di realizzare una elaborazione efficace delle immagini acquisite. Come anticipato nel capitolo 3, uno dei pacchetti fondamentali per la riuscita del progetto è *cv_bridge* [29]. Esso permette di convertire le immagini dai topic (che posseggono una loro particolare codifica) a strutture che OpenCV è in grado di analizzare e modificare. Ovviamente questo package presenta anche la possibilità di effettuare la conversione inversa: una volta modificate le immagini è possibile riconvertirle a immagini ROS compatibili con i topic per permettere la trasmissione dei fotogrammi modificati tramite i bus.

Si passa quindi alla funzione dedicata all'image processing[7] e all'identificazione dei marker e dei loro centri: questo metodo è chiamato all'interno del codice con il nome di *FindCenters*.

FindCenters

Quando questa funzione viene chiamata, il nodo ROS che contiene l'intero codice di questo pacchetto è già stato lanciato e ha iniziato a ricevere i primi dati dalle telecamere all'infrarosso. Come spiegato precedentemente, una volta che il nodo riceve informazioni dai topic a cui è iscritto viene lanciata una callback che permette di utilizzare quei dati per eseguire tutte le operazioni necessarie. Questo metodo viene infatti chiamato all'interno della callback ed è il primo che viene avviato.

L'identificazione dei centri dei marker è uno degli aspetti più delicati da realizzare, dovuto al fatto che una piccola variazione della loro posizione potrebbe condurre ad un calcolo errato della posa della wand rispetto alla telecamera. Per questo è stata condotta una ricerca dettagliata per comprendere appieno quale fosse il metodo migliore per il rilevamento dei marker, oggetti che, dal punto di vista delle Kinect, sono circolari.

È stato analizzato l'articolo di Tomurad *et al.* [44] che dimostra come sia possibile identificare accuratamente dei marker sferici all'interno delle immagini. Questa ricerca si basa sull'utilizzo della Hough Transform[8] per poter rilevare oggetti circolari, tecnica che ha portato a dei rilevamenti il cui errore è estremamente limitato. Si parla infatti di una precisione millimetrica che convertita in pixel si traduce in pochi pixel di errore

[7] L'elaborazione delle immagini.

[8] Tecnica utilizzata nell'ambito dell'analisi delle immagini per poter trovare linee rette. Nel corso degli anni questa è stata ampliata permettendo il riconoscimento di diverse forme come cerchi ed ellissi.

anche in condizioni in cui il cerchio risulta, nelle immagini di test, poco visibile all'occhio umano. Purtroppo a causa della lentezza del metodo utilizzato e della difficoltà di taratura dei parametri, l'applicazione di tale tecnica all'interno di questa tesi risulta difficoltosa. Di fatto, la calibrazione che si vuole effettuare ha l'obiettivo di essere realizzata in tempo reale. La Hough Transform richiede tuttavia una potenza computazionale non indifferente e l'impiego di essa potrebbe portare a un maggiore rischio di rallentamento dell'intero sistema. Inoltre, utilizzare OpenCV ci permette di avere la Hough transform già implementata al suo interno ma il suo utilizzo è fortemente condizionato dal valore dei suoi parametri, come ad esempio i valori di soglia da applicare durante la edge detection[9]: l'errata taratura di uno solo di questi può portare a ottenere risultati altamente imprecisi. Ulteriore particolare da non trascurare è la risoluzione dell'immagine usata, infatti in [44] la risoluzione utilizzata è molto maggiore (1280 x 1024 pixel) rispetto a quella fornita dal sensore *NFOV* delle Azure Kinect (640 x 576 pixel). Questo dettaglio rende maggiormente instabile il rilevamento poiché i marker a medie distanze (2-3 m) nell'immagine risulteranno solo come pochi pixel, rendendo così il sistema suscettibile ai più piccoli rumori di fondo. Si è deciso pertanto di non applicare tale tecnica.

Sono stati poi presi in considerazione gli studi di Fiedler *et al.* Nell'articolo [13] vengono presentate diverse metodologie per il rilevamento di marker sferici all'interno di immagini MRI[10]. L'obiettivo dello studio è equiparabile all'obiettivo che viene posto in questo elaborato: Fiedler *et al.*, in questo frangente, svolge un'accurata analisi delle varie tecniche che sono potenzialmente utilizzabili introducendo un algoritmo che renderebbe più facilmente rilevabili i marker all'interno dell'immagine. Da questo prende ispirazione l'algoritmo sviluppato in questo elaborato nel quale sono state apportate alcune modifiche per garantire l'osservazione dei marker più precisa ed imperturbabile da eventuali rumori di fondo.

Nello specifico, la procedura utilizzata all'interno della funzione FindCenters è visibile nell'algoritmo 1.

La prima operazione eseguita consiste in un'elaborazione dell'immagine: questa può rimanere basilare grazie al fatto che i marker sono estremamente visibili all'interno della scena e quindi non vi è la necessità di andare più a fondo con l'analisi. Le operazioni svolte in questa fase sono tre: un thresholding [11] iniziale, l'applicazione di una sfocatura Gaussiana e un thresholding finale.

[9] Rilevamento dei bordi. Questa tecnica permette di ricavare da un'immagine tutti i bordi degli oggetti presenti in essa. Il valore di soglia introdotto è necessario per far capire all'algoritmo in quali casi considerare un bordo tale e in quali no.

[10] Magnetic resonance imaging. Immagini proveniente dalle risonanze magnetiche.

[11] Tecnica che, dato un valore di luminosità, elimina o mantiene tutti i dati dell'immagine che si trovano al di sopra di quella soglia.

Algoritmo 1 FindCenters

Require: IR_image, image_width
Ensure: (contours, centers)
1: tmp_image ← IR_image
2: contours ← *initialize vector*
3: centers ← *initialize vector*
4: tmp_image ← `cv::threshold`(`cv::THRESH_TOZERO`, tmp_image)
5: tmp_image ← `cv::GaussianBlur`(tmp_image)
6: tmp_image ← `cv::threshold`(`cv::THRESH_BINARY`, tmp_image)
7: allContours ← `cv::findContours`(tmp_image)
8: **for each** single_contour ∈ allContours **do**
9: perimeter ← `cv::arcLength`(single_contour)
10: area ← `cv::contourArea`(single_contour)
11: circularity ← $4\pi * (area/(perimeter * perimeter))$
12: **if** circularity > MIN_CIRCULARITY **then**
13: point, radius ← `cv::minEnclosingCircle`(single_contour)
14: centers ← centers ∪ point
15: contours ← contours ∪ single_contour
16: **end if**
17: **end for**
18: **return** (contours, centers)

Il primo viene applicato con la modalità `THRESH_TO_ZERO`, ovvero tutti i valori al di sotto del valore soglia, scelto sperimentalmente, vengono eliminati, mentre quelli al di sopra rimangono invariati. Per rendere più uniforme possibile il cerchio (che nell'immagine IR rappresenta la sfera del marker), la sfocatura Gaussiana viene applicata con un raggio di dimensione 3 x 3 pixel.

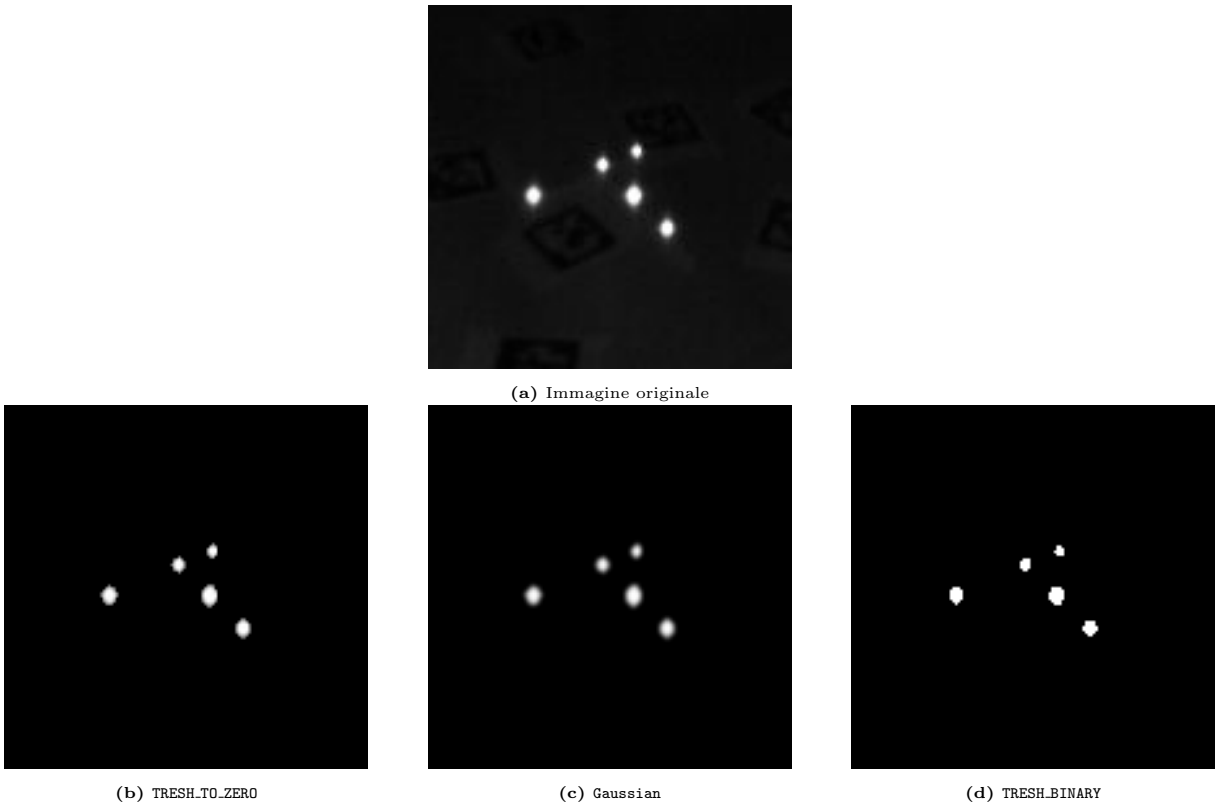
Il thresholding finale invece è binario, ovvero tutti i valori al di sotto della soglia sono posti a 0 (quindi il colore nero) mentre al di sopra sono posti al valore 255 (ovvero il bianco).

Nell'immagine 4.7 si possono notare in dettaglio gli effetti di queste operazioni applicate allo stesso fotogramma durante la ripresa di una delle telecamere.

In seguito a questo image processing viene utilizzata ancora una volta la libreria OpenCV per poter trovare i contour, i "contorni"[12], ovvero ottenere la circonferenza dei marker visibili nello spettro dell'infrarosso.

Una volta ottenuti tutti questi dati è indispensabile filtrarli per conservare solo quelli utili per l'obiettivo finale. Tale procedimento è messo in pratica tramite la verifica della circolarità dei contorni trovati. Per fare ciò si utilizza la seguente formula:

[12]Con questo termine si va ad intendere un insieme continuo di punti, lungo un bordo, che hanno lo stesso colore o la stessa intensità.



(a) Immagine originale

(b) `TRESH_TO_ZERO` **(c)** `Gaussian` **(d)** `TRESH_BINARY`

Figura 4.7: Queste immagini riproducono l'image processing effettuato dal pacchetto FindMarkers. In particolare si può effettuare il confronto con l'immagine originale e quelle derivanti dal processo. Sebbene 4.7d sembri essere di qualità inferiore rispetto a 4.7b e 4.7c è fondamentale poichè `findContours` (vedi algoritmo 1) necessita di bordi ben definiti per dare i migliori risultati. Inoltre l'immagine 4.7a presenta, anche se estremamente poco visibile, uno sfondo in cui sono presenti gli AprilTag.

$$c = \frac{4\pi * A}{P^2} \tag{4.2}$$

dove c rappresenta la circolarità, A indica l'area del marker e P è il perimetro del medesimo marker. L'equazione 4.2, sebbene non sia estremamente precisa, permette una veloce esclusione dei contorni che non esprimono una sufficiente forma circolare.

L'ultima azione compiuta dalla funzione *FindCenters* è il calcolo del centro di tutti i contour rimasti dopo la scrematura del passaggio precedente.

Per trovare il punto di mezzo di ogni marker il primo approccio è stato quello di utilizzare il momento delle sfere individuate. Per fare ciò è necessario effettuare una media pesata dell'intensità dei pixel dell'immagine sottoposta a tale procedura e dopodiché è possibile trovare il centroide di tale figura.

Tale metodo però si e rivelato inutilizzabile a causa del lieve rumore di fondo dell'immagine che muta il bordo dei marker e rende incostante l'identificazione del loro centro. Questa instabilità ha fatto sì che si optasse per una metodologia con risultati più costanti:

la funzione `minEnclosingCircle` di OpenCV. Questa permette, dato un contour, di trovare il cerchio più piccolo che circoscrive tale contorno; in questo modo, anche in presenza di piccole variazioni nella forma del marker, il suo centro rimane in una posizione molto più stabile rispetto al precedente procedimento.

Al termine di tutte queste operazioni la funzione restituisce tutti i contorni rilevati con i relativi centri che soddisfano i requisiti posti in questa sottosezione.

FindPose

In questa sottosezione si riporta come vengono utilizzati i dati ricevuti dalla funzione *findCenters* per trovare la posa della wand all'interno dell'immagine e dunque per individuare la matrice che permette di identificare tale posa partendo dalle coordinate della telecamera.

Una volta ricevuti tutti i dati da *findCenters* si effettuano ulteriori controlli per determinare se i punti restituiti corrispondono effettivamente alla posizione dei marker della bacchetta. Finite queste verifiche verrà applicata la funzione `solvePnP` della libreria di

Algoritmo 2 FindPose

Require: centers, cameraMatrix, distortionMatrix
Ensure: Transform
Transform ← *empty initialization*
rotation ← *empty initialization*
traslation ← *empty initialization*
if `size`(centers)= $n\_markers$ **then**
    objectPoints ← *initialize vector with wand world coordinates*
    imagePoints ← `orderPoints`(centers)
    collinearity ← $true$
    for each point ∈ imagePoints **do**
        if point = $(-1, -1)$ **then**
            collinearity ← $false$
        end if
    end for
    if collinearity **then**
        rvec, traslation ← `cv::solvePnP`(objectPoints, imagePoints, cameraMatrix, distortionMatrix, method)
        rotation ← `cv::Rodrigues`(rvec)
    end if
end if
Transform ← (rotation, traslation)
return Transform

OpenCV che restituirà i valori desiderati (rotazione e traslazione) riguardante la posa della wand.

Tali valori sono ottenuti utilizzando l'algoritmo 2.

Il primo controllo al quale sono sottoposti i dati riguarda il loro numero. Sapendo infatti che i marker presenti sulla wand sono 5, il numero di centri individuati durante la prima fase deve essere uguale a quest'ultimo. Difatti se la loro quantità supera questa soglia sarebbe estremamente complicato saper identificare quale centro appartenga effettivamente ad un marker e quale no. Per questo motivo si è deciso di applicare questa rigida regola, limitando al contempo la possibilità di ottenere un maggior numero di fotogrammi utili per calibrare il sistema di telecamere.

Il metodo `solvePnP` richiede come argomenti due vettori in particolare: objectPoints e imagePoints. Il primo rappresenta un array[13] di dimensione N, che in questo caso sarà 5, il quale contiene un insieme di punti che corrispondono alle coordinate dei marker nel sistema di riferimento del mondo. In breve esso conterrà i punti con gli stessi valori descritti nella sezione 4.1.2, dunque saranno espressi con coordinate tridimensionali. Il secondo vettore invece farà da contenitore per i punti che sono stati individuati nell'immagine infrarossa, espressi in coordinate bidimensionali. In questo preciso momento emerge la fondamentale importanza di riconoscere con precisione quale marker si è individuato. La funzione `solvePnP` richiede infatti che i punti espressi nei due vettori siano in relazione fra loro. Si prenda come riferimento l'immagine 4.5 in cui tutti i punti hanno un'etichetta. Nel vettore objectPoints il primo punto inserito corrisponde, per esempio, al punto A; allora anche nell'imagePoints il primo punto del vettore dovrà corrispondere al punto A della bacchetta all'interno dell'immagine IR. Questo procedimento deve essere svolto per tutti i punti: vi è il bisogno dunque di scrivere una funzione che permetta l'ordinamento dei punti ottenuti dalla funzione *findCenters*.

Il metodo in questione è stato chiamato `orderPoints` poiché, ricevendo come argomento un vettore contenente i punti della wand, ha lo scopo di ordinarli in modo corretto. L'ordinamento viene realizzato grazie alla asimmetricità dei marker nella wand, infatti utilizzando il rapporto delle distanze tra essi all'interno dell'immagine è sempre possibile stabilire quale marker si sta visualizzando e dunque l'algoritmo è in grado di porre i punti nell'ordine corretto. L'algoritmo `orderPoints` presenta anche una funzione aggiuntiva: se i punti non formano la struttura a T o le loro posizioni non corrispondono a quelle previste produrrà un vettore con alcuni punti vuoti, indicando così che le posizioni dei marker non sono valide. Di questa utile funzione ne sono state implementate due versioni.

La prima si basa sul concetto di collinearità[14] sul calcolo dell'area di un triangolo.

[13] Vettore.

[14] Un insieme di punti è collineare se essi giacciono sulla stessa retta.

Dati 3 punti essi si trovano sulla stessa retta se e solo se l'area del triangolo formato dai medesimi 3 punti è pari a 0. Individuate le due triplette di punti che soddisfano questa condizione, i punti sono messi in ordine utilizzando i rapporti fra le loro rispettive distanze. Tuttavia, considerato il rumore di fondo presente nei dati, i punti non risultano quasi mai perfettamente allineati e questo comporta un rilassamento della condizione dell'area del triangolo che risulta dunque quasi costantemente maggiore di 0. Questo rilassamento porta con sé problematiche inverse nel caso in cui la wand si allontani di molti metri dalla telecamera poiché le distanze fra i marker diminuiscono e tale diminuzione conduce al soddisfacimento della regola di collinearità molteplici volte dato che le aree fra i centri diventano di dimensioni ridotte.

La seconda implementazione invece si basa sul concetto del prodotto vettoriale: tre punti sono collineari se e solo se il loro prodotto vettoriale è pari a 0. Anche in questo caso è necessario applicare un rilassamento delle condizioni per la stessa causa del caso precedente. Viene dunque considerato valido $|prodotto| < threshold$ dove rilassamento è il delta entro il quale il valore è accettato. A questo nuovo metodo per calcolare se tre punti appartengono alla stessa retta, si aggiunge anche un procedimento per scoprire quali dei tre punti si trova nella posizione centrale. Quest'ultimo si basa sul concetto del prodotto scalare: dati 3 punti a, b, c se il prodotto scalare di $(b - a)$ e $(c - a)$ è positivo e minore della distanza fra a e b al quadrato allora il punto c si trova in mezzo agli altri due punti.

Quest'ultima implementazione ha portato a un notevole miglioramento dell'efficacia di questa funzione e dunque è stata inserita all'interno del progetto.

Una volta analizzata la coerenza sull'ordinamento dei punti individuati da *findCenters*, si passa ad analizzare la funzione principe di questo primo pacchetto: `solvePnP`.

Questa funzione permette di risolvere il problema della stima della posa della wand, ovvero del trovare una rotazione e una traslazione che minimizzi l'errore di riproiezione da una corrispondenza di punti 2D-3D (cioè che analizzi in entrata la posa della wand nei confronti della telecamera e ne restituisca i corrispettivi punti nell'immagine ripresa). In particolare la funzione calcola la matrice che permette di trasformare le coordinate di un punto (prendendo come origine la bacchetta, che in questo caso rappresenterà il sistema di riferimento mondo) nelle coordinate che hanno come sistema di riferimento la telecamera.

Come già anticipato questa funzione necessita di molteplici argomenti fra cui possiamo nominare: objectPoints, imagePoints, la matrice della camera e la matrice di distorsione.

Ad essi si aggiunge un ulteriore parametro che indica il metodo utilizzato per il calcolo di questa matrice. OpenCV ha implementato numerose soluzioni per il calcolo della posa

basandosi sulla letteratura. Purtroppo molti di questi richiedono un numero specifico di punti oppure una predeterminata configurazione di essi. Il sistema adottato è quindi quello chiamato dalla stessa libreria `SOLVEPNP_ITERATIVE`. Esso si basa su una ottimizzazione di Levenberg-Marquardt [30] la cui funzione permette di trovare una matrice in grado di minimizzare l'errore di riproiezione (che verrà calcolato come la distanza al quadrato fra gli imagePoints osservati e quelli calcolati a partire dagli objectPoints). Tale metodo richiede che vi siano almeno quattro punti complanari, il che corrisponde esattamente al nostro caso.

Dal punto di vista matematico l'equazione per trovare i parametri estrinseci di un singolo può essere così espressa:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{4.3}$$

In questa equazione i simboli u e v rappresentano, in pixel, le coordinate dei marker all'interno delle immagine. I valori f_x e f_y indicano la distanza focale della lente, tale numero è suddiviso nelle sue coordinate degli assi x e y. Il punto centrale della telecamera invece è rappresentato dai valori c_x e c_y che indicano la sua posizione all'interno dell'immagine. La seconda matrice, quella in cui sono presenti solo 1 e 0, indica i coefficienti di distorsione della camera che in questo caso non sono presenti infatti la matrice è un'identità. La terza matrice invece è la matrice di interesse per questa tesi, ovvero quella dei parametri estrinseci, e presenta la matrice di rotazione $\boldsymbol{R}$:

$$\boldsymbol{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{4.4}$$

e il vettore traslazione $\boldsymbol{t}$:

$$\boldsymbol{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{4.5}$$

L'ultimo vettore rappresenta invece le coordinate del marker all'interno del sistema di riferimento preso in considerazione (la lettera W indica che il sistema di riferimento preso in considerazione è il world).

Questa equazione permette di trovare i valori della matrice di rotazione $\boldsymbol{R}$ e il vettore traslazione $\boldsymbol{T}$ fondamentali, appunto, per la trasformazione di un punto 3D nel suo

corrispettivo punto 2D.

L'equazione che permette invece di stimare la posa delle telecamera risulta essere:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{4.6}$$

dove X_c, Y_c e Z_c sono le coordinare del punto espresse con la telecamera come sistema di riferimento.

La funzione `solvePnP` permette di unire le equazione per i singoli marker e tramite la sopra citata ottimizzazione di Levenberg-Marquardt ritorna il valore della traslazione e della rotazione della telecamera, ma al posto dell'intera matrice restituisce un vettore (rvec nell'algoritmo 2). Quest'ultimo può essere facilmente convertito nella matrice di rotazione tramite la formula di Rodrigues implementata direttamente nella libreria di OpenCV.

Ottenuti questi valori è possibile trovare la posa della telecamera rispetto alla wand invertendo la matrice di trasformazione ottenuta (ovvero quella creata a partire dal vettore traslazione e dalla matrice di rotazione). Fatto ciò, al pacchetto non rimane altro che pubblicare su un apposito topic le informazioni ottenute sotto forma di trasformate.

4.2.2 Wand Calibration

Il pacchetto precedente rappresenta l'elemento principale della calibrazione, difatti se il suo funzionamento non è ben studiato e implementato l'intero progetto risulterebbe di poca utilità. Questo package ha il compito di raccogliere tutti i dati generati dal precedente pacchetto, analizzarli e portare a termine la calibrazione della rete di telecamere.

Wand Calibration analizza le trasformate pubblicate da Find Markers. Per fare ciò è indispensabile iscrivere il nodo ROS, generato da questo package, ai topic dove vengono pubblicate. In questo particolare caso il topic in questione è stato denominato con */cameraId/transforms*; dunque in un sistema con molteplici telecamere ognuna di esse avrà un topic dedicato in cui vengono pubblicate le trasformate trovate da Find Markers.

Per comprendere a fondo il problema è essenziale illustrare le trasformate e introdurre il concetto di tf e tf_tree.

All'interno di ROS le pose degli oggetti sono espresse come trasformazioni da un sistema di riferimento a un altro. Per spiegare approfonditamente questo concetto è utile

introdurre un esempio: si presuppone che un robot dotato di una mano debba prendere un oggetto di fronte a sé, per fare ciò deve essere a conoscenza di quanto la mano è distante dall'oggetto e dunque di quanto deve muoverla per poterlo prendere. Lo spostamento che deve fare la mano per poter raggiungere l'oggetto è la trasformata che permette di esprimere la posa dell'oggetto rispetto alla mano. La trasformata dunque è così chiamata poiché trasforma le coordinate dell'obiettivo in quelle della mano. In breve esprime la posa di un oggetto rispetto alla posa di un altro.

Le trasformate sono pubblicate all'interno dei topic in dei "contenitori" che racchiudono altre informazioni utili per lo studio del sistema, come: data e ora di pubblicazione, il nome del sistema di riferimento principale e il nome di quello secondario (nell'esempio di prima sarebbero stati rispettivamente mano e oggetto). Questi contenitori sono chiamati tf e all'interno dell'infrastruttura di ROS sono pubblicati in canali dedicati (questo viene fatto per permettere a un qualsiasi nodo di reperire delle tf pubblicate poichè sono tutte pubblicate nello stesso topic).

Tutte le tf sono messe in relazione tra loro. Riprendendo l'esempio antecedente: una volta che il robot sa dov'è l'oggetto rispetto alla mano, per compiere il movimento in modo corretto deve sapere anche in che posa è la mano rispetto al proprio corpo, e la sua posa all'interno della stanza in cui si trova. Tutti questi legami (o informazioni) sono raccolti in ciò che viene chiamato tf_tree, ovvero un grafo che mostra le relazioni fra i vari sistemi di riferimento. La struttura di questo grafico è particolare in quanto non permette la formazione di cicli all'interno di esso ma solo una conformazione ad "albero", proprio come il termine inglese "tree" lascia intuire.

Questa tipologia di grafi viene spesso ricondotta al concetto di padre e figlio. Per spiegare questa similitudine riprendiamo l'esempio citato: se le coordinate dell'oggetto che il robot vuole prendere sono espresse dal punto di vista della mano, quest'ultima sarà il "padre" mentre l'oggetto sarà il "figlio". Per evitare la formazione di cicli vi è l'esigenza che ogni frame (così vengono chiamati i sistemi di riferimento all'interno di ROS) possa avere molteplici figli ma un solo padre. Per mantenere tale ordinamento in questa tesi vi sarà un'unica telecamera che corrisponderà al "padre" mentre tutte le altre saranno "figli", dunque il nostro sistema di riferimento avrà come suo centro la telecamera genitore.

Per effettuare una corretta calibrazione il pacchetto verifica tutte le trasformate trovate da Find Markers e quando la wand è vista in contemporanea in 2 o più telecamere è dunque possibile salvare le informazioni ricevute per calcolare le posizioni relative delle camere.

Si pone l'esempio di due telecamere le cui matrici di trasformazione rispetto ad un

sistema di riferimento F sono le seguenti: $\boldsymbol{T}_{C_1}^F = [\boldsymbol{R}_1|\boldsymbol{t}_1]$ e $\boldsymbol{T}_{C_2}^F = [\boldsymbol{R}_2|\boldsymbol{t}_2]$, dove $\boldsymbol{R}$ è la matrice di rotazione e $\boldsymbol{t}$ è il vettore traslazione. Avendo questi dati è possibile trovare la posa relativa delle due camere risolvendo la seguente equazione: $\boldsymbol{T}_{C_1}^F \boldsymbol{X} = \boldsymbol{T}_{C_2}^F$. Tale equazione trova dunque soluzione nella seguente formulazione: $\boldsymbol{X} = (\boldsymbol{T}_{C_1}^F)^{-1} \boldsymbol{T}_{C_2}^F$.

Avendo introdotto la configurazione con cui si presenterà la calibrazione una volta terminata, è possibile specificare come essa avvenga all'interno di questo pacchetto. Il nodo ROS creato viene iscritto a tutti i topic che pubblicano le trasformate relative alle posizioni delle telecamere rispetto alla bacchetta. Per poter capire quali telecamere vedono in contemporanea la wand è possibile utilizzare diverse strategie:

- la prima è di usare l'ora di pubblicazione per vedere in quali frame è vista in contemporanea;
- la seconda è quella di aggiornare una mappa[15] che contiene una coppia di valori (nome camera, trasformata) ogni qualvolta la bacchetta viene vista da una telecamera.

I dati raccolti sono aggiornati in base alla frequenza di aggiornamento delle telecamere, che può variare da sistema a sistema. In questo particolare caso la frequenza di aggiornamento delle Kinect usate è di 30 Hz (vedi sezione 4.1.1). Ciò significa che le telecamere riescono a produrre 30 immagini al secondo, ovvero una ogni circa 33 ms. Dunque, considerata la frequenza di aggiornamento, se si utilizza la prima strategia saranno raccolti i dati su quali trasformate sono state pubblicate in un intervallo di tempo di 33 ms, mentre se viene utilizzata la seconda la mappa in cui vengono inserite le trasformate sarà svuotata ogni 33 ms. Entrambe le modalità sono valide ma per questa implementazione si è optato per la seconda in quanto ROS permette di eseguire dei cicli all'interno del codice della durata desiderata

Ogni qualvolta il nodo riceve una trasformata, viene lanciata una callback che salva nella mappa tale valore, insieme alla telecamera che l'ha pubblicato. Nella funzione principale del nodo, la mappa temporanea viene esaminata ad ogni ciclo di 33ms e tutti i dati trovati al suo interno vengono considerati come simultanei. Tutte le informazioni, raccolte a coppie, sono salvate all'interno di un'altra mappa, il cui compito sarà quello di mantenere tutti i dati relativi alle pose delle telecamere in modo definitivo.

[15] Una particolare tipologia di struttura dati. Memorizza gli elementi nella forma di coppia chiave-valore.

Il modo in cui viene eseguita la calibrazione è simile a quello del metodo analizzato nel capitolo 2 per la calibrazione tramite AprilTag, ovvero a coppie. Difatti anche in questo caso la calibrazione avviene a due a due ed infatti nella mappa tutte le trasformate sono salvate a coppie. Per esempio se la bacchetta viene vista in contemporanea dalle telecamere 1, 2 e 3 verranno salvati i dati delle seguenti coppie: 1-2, 1-3 e 2-3. È importante notare che non viene salvata direttamente la posa relativa fra le Kinect ma proprio le trasformate ricevute in quel preciso istante.

Dunque in questo momento si ha un sistema che ogni qualvolta due o più telecamere rilevano i marker, salva i dati in una specifica struttura, che in questo caso è rappresentata da un mappa.

La calibrazione effettiva avviene nella parte finale del pacchetto. Ora che i dati hanno una specifica organizzazione è possibile cominciare ad eseguire il calcolo delle pose relative delle varie telecamere. Per fare ciò è stato sviluppato un algoritmo che permette di calcolare la matrice di trasformazione tra due telecamere tramite il metodo dei minimi quadrati. Ciò avviene costruendo un sistema di equazioni matriciali in cui viene calcolata la matrice di trasformazione che si avvicini il più possibile all'intero insieme di trasformate. Terminata questa esecuzione si otterrà una mappa con al suo interno tutte le possibili coppie di telecamere (come quelle viste precedentemente, ad esempio 1-2, 1-3 e 2-3) del sistema: ad ogni coppia corrisponderà la matrice di trasformazione delle pose relative.

Per rendere il processo più robusto si è deciso di introdurre un sistema di raffinamento dei risultati finali. Basando la calibrazione solo sul calcolo delle posizioni sulle singole coppie, la calibrazione del sistema sarebbe instabile in quanto, ad esempio, una coppia di telecamere potrebbero aver visto solo per pochi fotogrammi i marker in contemporanea e dunque avendo pochi dati a disposizione la loro posa relativa risulterebbe fragile, ottenendo così un risultato poco valido. Per far fronte a ciò si è deciso di individuare la posa finale mettendo in relazione tutte le telecamere del sistema.

Consideriamo un sistema a tre camere: 1, 2 e 3. Nel caso in cui si vogliano trovare le pose relative fra le Kinect 1 e 3 non si baserà l'intero calcolo sui soli fotogrammi in cui entrambe vedono la wand ma si utilizzerà le informazioni della camera numero 2 per migliorare la sua posa poiché si potrà così integrare anche l'informazione in merito alla posa della camera 3 rispetto alle 2. Utilizzando queste informazioni dunque si riesce ad ottenere una soluzione più robusta ad eventuali perturbazioni del sistema.

La pubblicazione delle trasformate finali in questo caso non viene eseguita su un topic

ad hoc ma mediante l'utilizzo del topic di sistema */tf* che raccoglie tutte le trasformate pubblicate da tutti i nodi ROS.

4.3 Struttura dei nodi del sistema

Nelle sezioni precedenti è stato spiegato dettagliatamente il funzionamento dei pacchetti creati durante il corso di questa tesi. In questa sezione si approfondirà il tema dei nodi e delle loro comunicazioni.

Nelle sezioni 4.2.1 e 4.2.2 il passaggio di dati dai nodi è stato spiegato come iscrizioni/pubblicazioni all'interno dei topic senza specificare il rapporto che ogni nodo ha con gli altri presenti nella rete.

Faremo riferimento al caso specifico della sperimentazione di questa tesi, ovvero un sistema di cinque camere Azure Kinect.

In ROS ogni immagine o informazione generata da qualsiasi camera viene pubblicata su un determinato topic. Portando un esempio, le informazioni che sono servite per lo sviluppo di questo progetto si trovavano rispettivamente sui topic:

- */cameraId/ir/image_rect*;
- */cameraId/ir/camera_info*.

Dove *cameraId* indica un codice identificativo, *ir* rappresenta il tipo di camera utilizzato, e *image_rect* e *camera_info* sono i topic che portano le immagini IR e le informazioni sulla camera usata (come la matrice di essa o la matrice di distorsione).

Per il corretto funzionamento degli algoritmi proposti, e per sfruttare al massimo la parallelizzazione che ROS permette di sviluppare tramite l'uso dei suoi nodi, sono stati utilizzati 5 nodi Find Markers, ognuno dei quali analizza le immagini di una singola telecamera e calcola la posa della wand all'interno di essa nel caso in cui sia visibile. Ognuno di questi nodi pubblica successivamente su un topic apposito il risultato del rilevamento, chiamati: */cameraId/transforms*.

Il nodo per effettuare la calibrazione, invece, è unico. Esso si iscrive a tutti i topic che pubblicano le trasformate e tramite questi avviene la calibrazione dell'intera rete di telecamere.

L'immagine 4.8, realizzata tramite l'uso dello strumento *rqt_graph* di ROS, rappresenta la struttura ottenuta.

Il grafo presenta alcuni topic non utilizzati (identificabili dall'assenza di frecce in uscita) in quanto non trasportano dati utili allo scopo finale della tesi.

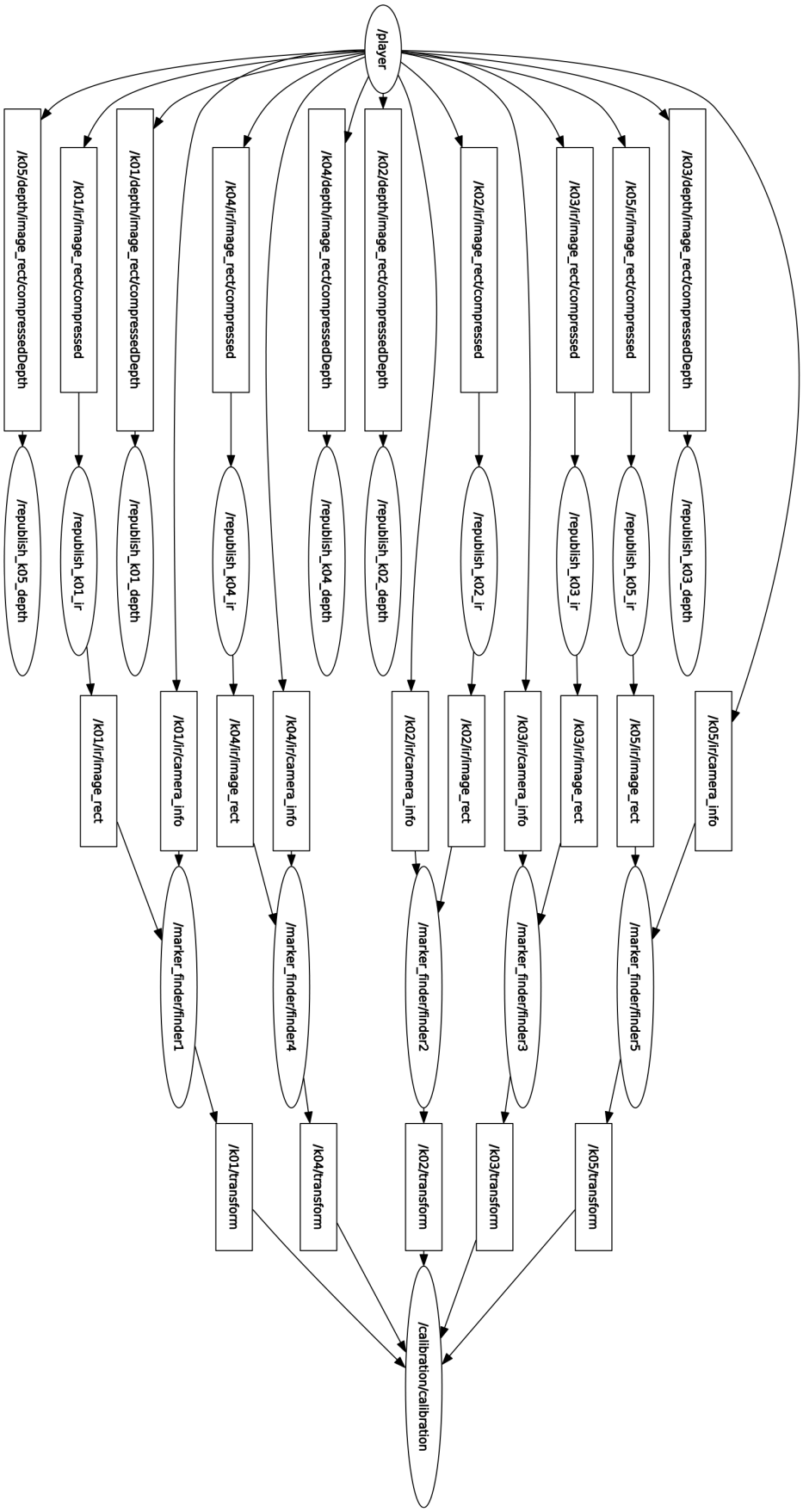


Figura 4.8: Struttura dei nodi del sistema.

L'avvio di tutti i nodi è stato effettuato tramite l'uso di un file *.launch* che permette, grazie all'uso del linguaggio di markup XML, di lanciare più nodi ROS in contemporanea e di gestire in modo semplice i loro parametri (come ad esempio topic a cui iscriversi o topic su cui pubblicare determinate informazioni).

4.4 Conclusioni

In questo capitolo sono stati analizzati gli algoritmi sviluppati nel corso di questa tesi. Prima di entrare nel loro dettaglio è stata approfondita la strumentazione hardware usata: le telecamere e la wand. Delle prime sono state fornite le specifiche tecniche e la motivazione del loro utilizzo. Per la seconda, invece, sono state spiegate le scelte progettuali per la sua realizzazione. Successivamente sono stati ampiamente illustrati i pacchetti ROS proposti: *Find Markers* e *Wand Calibration*.

Find Markers ha il ruolo di stabilire la posa della bacchetta rispetto a ogni telecamera. Per fare ciò sono stati sviluppati due algoritmi: *findCenters* e *findPosition*. Il primo permette di trovare i centri dei marker visualizzati nella scena, mentre il secondo analizza i dati ricevuti dal primo per trovare la posa della wand relativamente a quella della telecamera.

Wand Calibration invece prende i dati generati da Find Markers e effettua la calibrazione del sistema di telecamera seguendo una procedura simile a quella proposta nella sezione 2.1, ovvero eseguendo una calibrazione a coppie i cui risultati sono ottimizzati tramite un apposito algoritmo.

Oltre a questi algoritmi è stata spiegata la struttura dei nodi ROS utilizzati durante la calibrazione, specificando che ne sono stati usati cinque del tipo Find Markers (uno per telecamera) e uno Wand Calibration per la calbrazione della rete.

Capitolo 5

Esperimenti eseguiti

In questa capitolo si affrontano i risultati ottenuti durante il progetto di tesi.

Verrà riportato come è stata garantita la riproducibilità dell'esperimento, permettendo dei confronti veritieri con il metodo utilizzato come riferimento. Saranno poi presentati i risultati della calibrazione del sistema di telecamere tramite il metodo analizzato nella sezione 2.2, ovvero tramite l'utilizzo degli AprilTag. Come ultimo argomento verranno presentati i risultati ottenuti dall'algoritmo sviluppato e sarà fatto un confronto con il metodo precedente.

5.1 Acquisizione dei dati e risultati della calibrazione con AprilTag

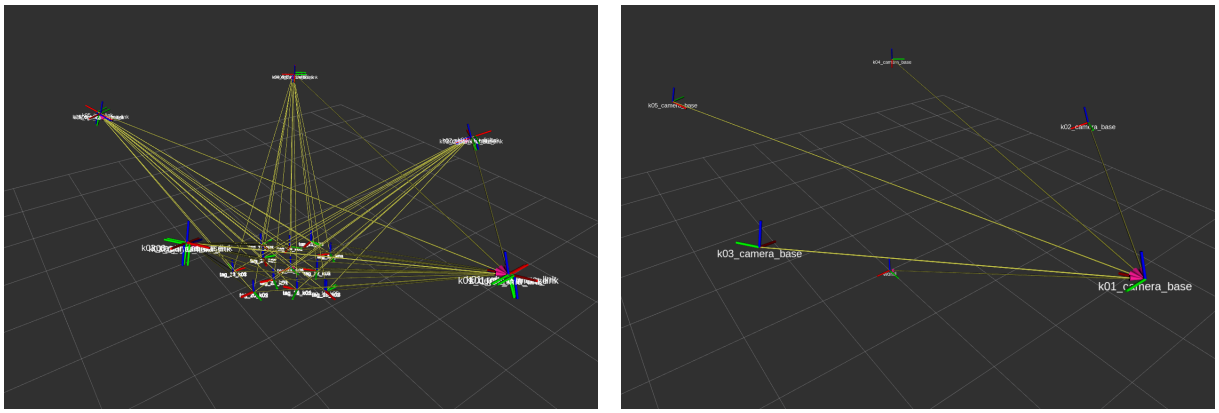
Per permettere un corretto confronto con lo stato dell'arte, è necessario assicurarsi che i test effettuati siano ripetibili. Per questo motivo durante lo sviluppo si è scelto di utilizzare un altro importante strumento messo a disposizione da ROS: le bag (sezione 3.2.1).

Nel capitolo 4 è stato specificato che all'interno di queste sperimentazioni le telecamere usate sono state cinque Azure Kinect poste in una disposizione a cerchio. Sono state utilizzate tuttavia ulteriori strumentazioni: la bacchetta (wand) e 12 AprilTag. Dunque è stata registrata una bag, utilizzata sia per la calibrazione di riferimento sia per quella sviluppata in questa tesi, in cui tutti gli strumenti sono stati posti inizialmente sul pavimento all'interno della scena, in modo tale che fossero visibili a tutte le telecamere. Successivamente la wand è stata presa in mano ed è stata mossa per effettuare il processo di calibrazione.

Calibrazione tramite AprilTag

I risultati della calibrazione di riferimento saranno mostrati tramite immagini realizzate con RViz[1] che rendono possibile la visualizzazione dei tf_tree calcolati dal programma. Come anticipato nella sezione 4.2.2 questi devono presentare una struttura ad albero, dunque non vi devono essere cicli al loro interno; per fare ciò si è identificata una telecamera che venga considerata come "principale", ovvero il punto di riferimento attraverso il quale la posizione delle telecamere è espressa. Durante la sperimentazione è stata individuata tale figura nella Kinect numero 1.

L'immagine 5.1 mostra due versioni della stessa scena. Nella prima immagine è possibile osservare tutte le trasformate calcolate durante la calibrazione. Ne sono presenti molte poiché per ogni singolo AprilTag il pacchetto creato pubblica tutte le pose relative trovate. Per questo motivo è stata aggiunta una seconda immagine che permette di osservare in modo chiaro come siano predisposte le telecamere. In quest'ultima è presente un frame intitolato "world": esso rappresenta il sistema di riferimento, mondo appunto, ma rispetto al quale sono espresse le pose delle telecamere.



(a) Immagine originale **(b)** Immagine contenente solo le informazioni necessarie

Figura 5.1: Queste immagini raffigurano i risultati della calibrazione ottenuti con il metodo degli AprilTag.

Per permettere un confronto tra la calibrazione ottima e quella realizzata in questo elaborato è opportuno esprimere la coordinate delle altre Kinect in funzione di quella "principale", in questo modo si ottiene la figura 5.2. Sempre tramite il software RViz è possibile visualizzare i dati precisi delle posizioni delle telecamere (per ovvi motivi la telecamera numero 1 non sarà in questa lista poiché la sua posizione sarebbe sempre nulla in quanto è utilizzata come sistema di riferimento).

In particolare per ogni telecamera verranno esplicitate la traslazione e la rotazione nella tabella 5.1.

[1]Strumento di visualizzazione 3D per le applicazioni ROS.

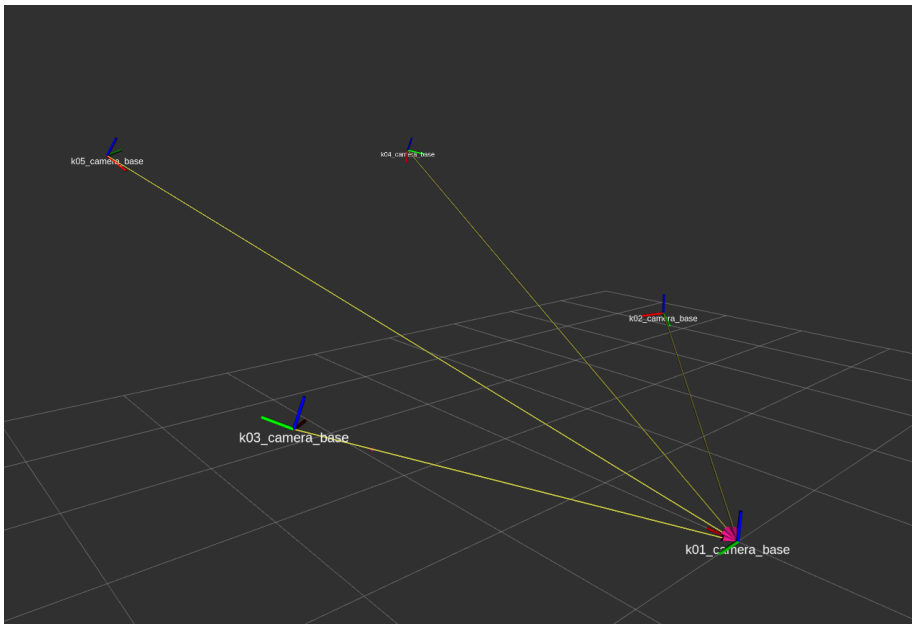


Figura 5.2: Questa immagine rappresenta i risultati della calibrazione espressi in riferimento alla Kinect 1.

Kinect id	Traslazione			Rotazione			
	x	y	z	x	y	z	w
Kinect 2	1.8666	−1.8616	0.7540	−0.18113	−0.01373	0.42621	0.88620
Kinect 3	1.6689	2.0077	0.7341	0.17257	0.00487	−0.40432	0.89818
Kinect 4	5.1755	−1.4423	2.1441	−0.38063	−0.00999	0.88022	0.28327
Kinect 5	5.028	1.9026	2.1644	−0.35950	−0.01919	0.87166	−0.33257

Tabella 5.1: Questa tabella riassume i risultati della calibrazione tramite AprilTag.

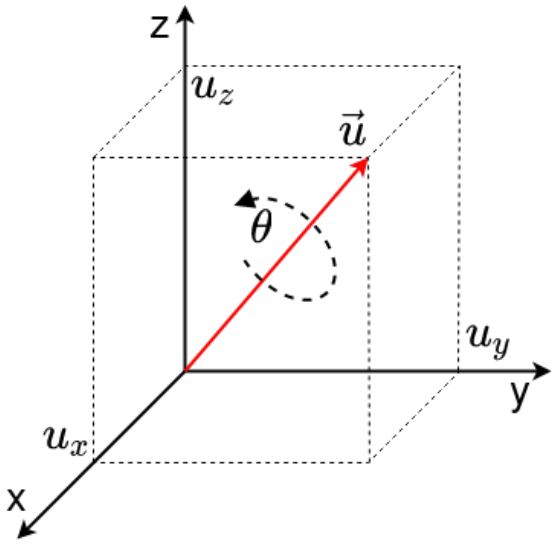


Figura 5.3: Questa immagine rappresenta la struttura base del quaternione. I valori (u_x, u_y, u_z) indicano rispettivamente i valori (X, Y, Z), mentre $\vec{u}$ è il vettore creato a partire da quei valori. θ è invece lo scalare che indica la rotazione attorno all'asse appena creato $(\vec{u})$.

La traslazione è stata fornita secondo i seguenti parametri: X, Y, Z. La rotazione invece è stata ottenuta seguendo la struttura del quaternione: $q = a + b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}$. Dove (i, j, k) rappresentano i 3 assi (X, Y, Z) mentre (b, c, d) sono gli scalari per cui essi sono moltiplicati e a rappresenta la rotazione. L'utilizzo di questa rappresentazione rende più semplice, dal punto di vista computazionale, il calcolo della moltiplicazioni di rotazioni rispetto all'utilizzo di matrici poiché richiedono meno operazioni da svolgere [14][56].

Questa particolare rappresentazione si basa sulla capacità di ogni rotazione di essere rappresentata da un vettore $\vec{u}$ ed uno scalare θ. Il vettore, composto da X, Y, Z, indica l'asse di rotazione dell'oggetto, mentre lo scalare θ denota l'angolo di rotazione attorno ad esso.

Avendo una visione chiara dei risultati ottenuti ora è possibile proseguire con il confronto con il metodo creato in questa tesi.

5.2 Confronto con il metodo implementato

In questa sezione verranno analizzati i risultati della calibrazione del sistema di telecamere all'infrarosso introdotto nella sezione 4.1.

È bene notare che per la modalità con cui è stato implementato il nuovo approccio, l'intero sistema è in continuo apprendimento, ovvero i dati relativi alla wand sono continuamente salvati e sfruttati per migliorare la calibrazione.

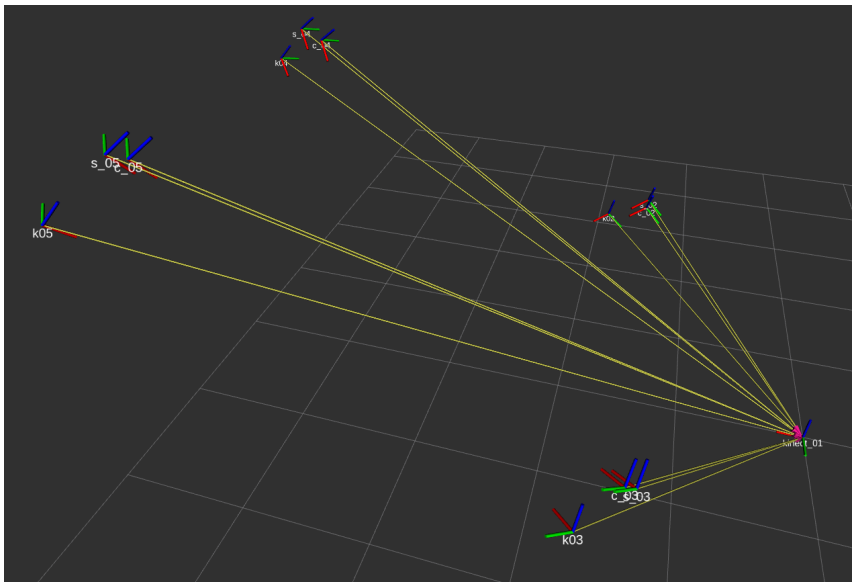


Figura 5.4: Questa immagine confronta i risultati della calibrazione ottenuti con i dati acquisiti in riferimento alla Kinect 1. I dati della calibrazione con AprilTag sono indicati con la lettera k, quelli ottenuti con i soli dati statici con la lettera s e quelli con tutti i dati acquisiti con la lettera c.

I dati raccolti per effettuare la calibrazione sono di due tipi: statici e in movimento. I primi sono tutte quelle informazioni ottenute quando la wand è in una posizione statica. I secondi dati provengono invece da tutte le volte in cui la bacchetta si trova in movimento all'interno della scena. I risultati riportati in questa sezione saranno chiamati per comodità in due modi, statica e completa. La prima utilizzerà solo i dati statici mentre la seconda userà, in modo combinato, dati statici e dinamici.

Nelle tabelle 5.2 e 5.3 sono riportati i dati relativi alla calibrazione ottima, con i dati statici e quelli completi (ovvero utilizzando tutti i dati registrati dalla bag, statici ed in movimento).

Come è possibile vedere dai dati grezzi i valori ottenuti al termine della calibrazione differiscono da quelli ottenuti utilizzando solo i dati della wand in condizione di immobilità. In particolare si può notare un "avvicinamento" delle telecamere nei confronti della Kinect principale, ad esempio si può notare la camera numero 4 si è avvicinata per un totale di circa 10 centimetri.

Dai dati presenti nelle tabelle 5.2 e 5.3 si può capire che l'attuale metodo implementato non raggiunge i risultati desiderati, ovvero è ben distante da poter essere utilizzato in un contesto reale con risultati precisi. Dalla tabella 5.4 si può notare che l'angolazione delle telecamere ottenuta risulta essere errata di circa 0.15 rad, l'equivalente di circa 9°; la posizione, sia con dati statici sia con la totalità dei dati, risulta essere errata di circa 30 centimetri per telecamera. In figura 5.4 è possibile osservare il confronto visivamente.

Tipologia	Kinect 2			Kinect 3			Kinect 4			Kinect 5		
	x	y	z	x	y	z	x	y	z	x	y	z
Ottima	1.8666	−1.8616	0.7540	1.6689	2.0077	0.73409	5.1755	−1.4423	2.1441	5.028	1.9026	2.1644
Statica	1.5172	−1.6308	1.09	1.349	1.7007	0.8822	4.8323	−1.2691	2.4967	4.6179	1.6813	2.5825
Completa	1.5214	−1.605	1.0173	1.4147	1.6889	0.85296	4.6098	−1.2059	2.4318	4.5111	1.6293	2.5467

Tabella 5.2: Risultati della traslazione ottenuti dalle varie calibrazioni.

Tipologia	Kinect 2				Kinect 3			
	x	y	z	w	x	y	z	w
Ottima	−0.18113	−0.01373	0.42621	0.88620	0.17257	0.00487	−0.40432	0.89818
Statica	−0.24543	0.01643	0.38538	0.88937	0.22742	0.01464	−0.35323	0.90735
Completa	−0.24131	0.0057208	0.38263	0.89181	0.23133	0.0051795	−0.36193	0.90303

Tipologia	Kinect 4				Kinect 5			
	x	y	z	w	x	y	z	w
Ottima	−0.38063	−0.00999	0.88022	0.28327	−0.3595	−0.019194	0.87166	−0.33257
Statica	−0.4453	0.00474	0.85534	0.26473	−0.43249	−0.05114	0.83966	−0.3245
Completa	−0.4556	0.00098	0.85145	0.25972	−0.43818	−0.04668	0.83892	−0.31942

Tabella 5.3: Risultati della rotazione ottenuti dalle varie calibrazioni.

Tipologia	Traslazione		Rotazione	
	Media	σ	Media	σ
Statica	0.3106	0.0380	0.1588	0.0117
Completa	0.3327	0.0770	0.1623	0.0158

Tabella 5.4: Errori ottenuti con la calibrazione introdotta in questa tesi. σ indica la deviazione standard. I valori della traslazione sono dati in metri, mentre quelli dalla rotazione in radianti.

5.3 Conclusioni

In questo capitolo sono riassunti i risultati delle varie calibrazione effettuate nel corso della tesi. Dapprima è stato spiegato in che modo, tramite le bag, è stata permessa la riproducibilità del sistema di telecamere. Successivamente sono state analizzate le pose delle telecamere ottenute tramite la calibrazione con AprilTag. Tali esiti sono stati messi a confronto con gli algoritmi sviluppati nel capitolo 4 e si è potuto notare che la nuova metodologia è ancora acerba, le posizione calcolate delle telecamere risultano essere errate di circa 30 cm mentre la rotazione ha un errore di circa 0.15 rad. La media degli errori e la deviazione standard sono consultabili alla tabella 5.4.

Capitolo 6

Conclusioni e possibili miglioramenti

La calibrazione di sistemi di telecamere è spesso lunga, laboriosa e presenta inoltre la necessità di dover utilizzare oggetti di grandi dimensioni come ad esempio le classiche scacchiere. Questa tesi nasce per superare queste difficoltà e rendere l'intero processo estremamente rapido ed efficace, tramite l'uso di una wand con marker retroriflettenti.

Il progetto si è concentrato su una determinata tipologia di sensori, ossia quelli a infrarosso, poiché permettono l'uso di marker e dunque di emulare il processo di calibrazione dei sistemi optoelettronici, considerati come il punto di riferimento per la Motion Capture. A tal fine è stato realizzato un algoritmo che utilizzasse una strumentazione diversa, una bacchetta con dei marker retroriflettenti, più piccola, maneggevole e robusta rispetto alle classiche scacchiere, e che sfruttasse appieno i vantaggi offerti da due importanti librerie: OpenCV e ROS. Il codice sviluppato nel corso di questo elaborato è stato testato su un sistema di cinque telecamere Azure Kinect poste in cerchio così da inquadrare contemporaneamente la stessa scena. La calibrazione è avvenuta muovendo la bacchetta all'interno dell'ambiente di sperimentazione.

I risultati ottenuti dall'algoritmo in questione sono lungi dall'essere paragonati a quello tramite AprilTag analizzato in sezione 2.2. La loro traslazione ha riportato un errore di notevole entità, ovvero nell'ordine delle trentina di centimetri tuttavia la rotazione di ogni telecamera ha ottenuto un risultato migliore con un errore di circa 9°.

La calibrazione utilizzata dunque non rappresenta ancora una valida alternativa all'attuale stato dell'arte, ma pone alcune basi importanti per svincolarsi dai classici metodi di calibrazione. Per migliorare questo sistema e renderlo una valida alternativa in futuro, sono state valutate le possibili cause che hanno portato a questa conclusione.

Durante l'analisi effettuata è stato individuato come principale responsabile il metodo solvePnP utilizzato per il calcolo della matrice di trasformazione 3D-2D, la cui impreci-

sione è alla sua dipendenza degli imagePoints individuati durante la fase findCenters. Difatti, come spiegato nella sezione 4.2.1, esso riesce a trovare la migliore trasformata che permette di minimizzare il risultato del metodo dei minimi quadrati, ma se i dati passati al metodo risultano imprecisi allora il valore restituito da esso risulterà a sua volta errato.

L'analisi si è concentrata sulle possibili cause di tali imprecisione. Sicuramente il rumore di fondo delle immagini gioca un ruolo fondamentale: nonostante le misure adottate per ridurlo al minimo, le deformazioni apportate alle rappresentazioni dei marker non consentono una corretta identificazione dei loro centri. Infatti anche durante la fase statica della calibrazione si riportano movimenti minimi dei centri trovati benché essi non si muovano effettivamente e uno dei motivi a causare questo effetto è riconducibile alla bassa risoluzione dell'immagine acquisita dalle telecamere (che ricordiamo essere 640 x 576). Per risolvere tale problema sono state tentate alcune modifiche all'algoritmo, come la modifica dei valori soglia utilizzati per le operazioni di threshold, per l'elaborazione dell'immagine, ma nessuna di esse ha portato ad un miglioramento della calibrazione. Nell'immagine 6.1 si possono trovare due fotogrammi della stessa scena processati allo stesso modo e si può notare che vi sono alcune differenze nella forma e dimensione del marker.

Altro possibile problema risiede nell'effetto della riflessione della luce. I marker infatti hanno esattamente questo compito, ma l'uso di raggi infrarossi provenienti da molteplici direzioni e la possibile presenza di imperfezione sulla superficie delle sfere usate può causare un mutamento della dimensione assunta da esse durante la calibrazione. Nella figura 6.2 è possibile notare che sulla parte verticale della wand, che è in una posizione statica (posta sul pavimento), il marker centrale risulta essere di dimensioni considerevolmente maggiori rispetto agli altri. L'aumento di tale dimensione porta ad una maggiore difficoltà nel rilevamento corretto del suo centro che, unita al rumore di fondo, rende complicato aumentare la precisione dell'algoritmo.

Infine l'ultima difficoltà riscontrata è ricollegabile alla frequenza di aggiornamento delle Kinect. Il problema in effetti non è conseguenza diretta di esso ma ne è un effetto collaterale. Se alcuni movimenti effettuati con la bacchetta sono troppo rapidi vi è la possibilità che il fotogramma acquisito sia sfocato e che il marker assuma una forma più ellittica che, sebbene rispetti il vincolo di circolarità posto nella sezione 4.2.1, determina uno spostamento del suo centro. Nell'immagine 6.3 è possibile vedere come il fotogramma presenti alcuni marker la cui forma non è più circolare.

Le migliorie da poter effettuare in ulteriori ricerche future dunque risiedono principalmente in ambito hardware come: l'utilizzo di telecamere all'infrarosso con risoluzione più

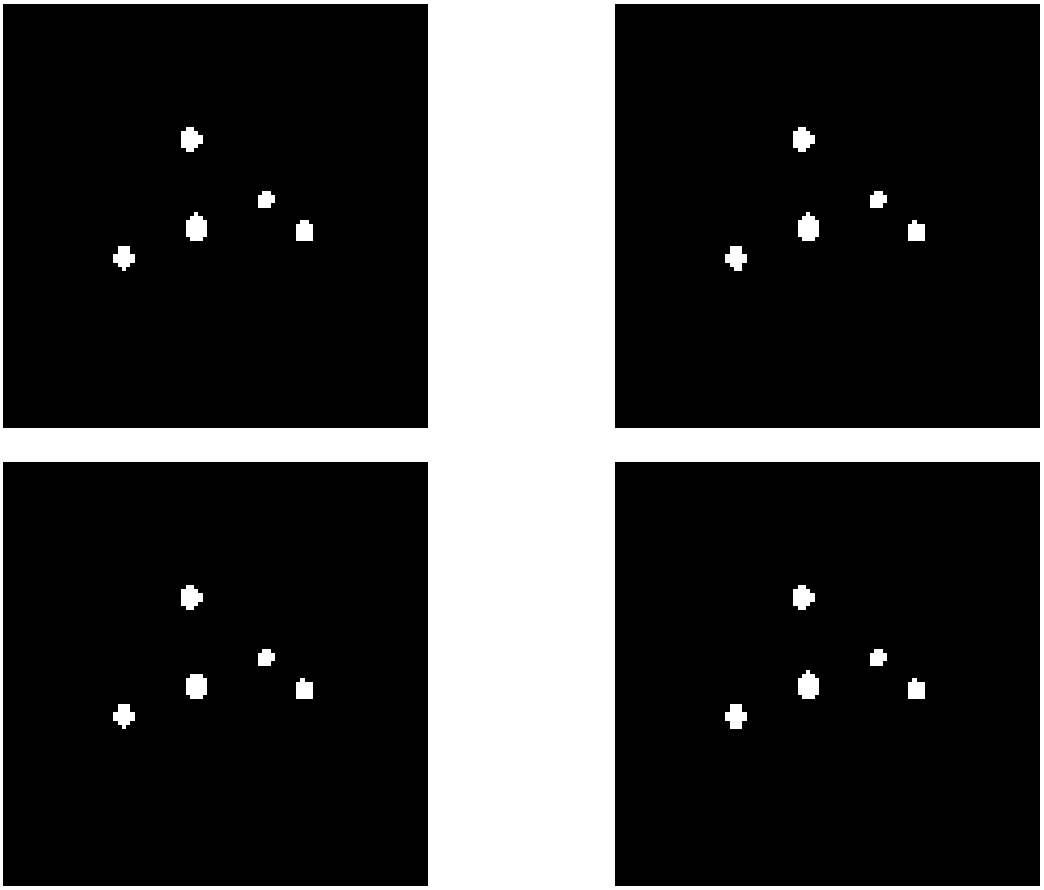


Figura 6.1: Nell'immagine si possono notare le differenze fra i pixel individuati come marker.

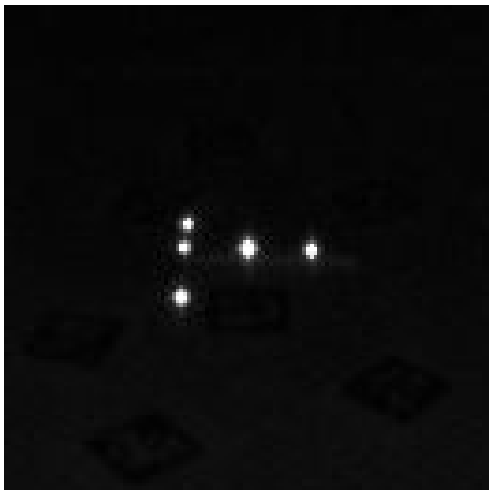


Figura 6.2: I marker presenti sulla wand, sebbene siano della stessa dimensione, riflettono la luce in modo diverso provocando così discrepanze nell'immagine.

Figura 6.3: Nell'immagine si possono notare che i marker perdono la loro forma circolare.

elevata (ad esempio 1280 x 1024) e con una frequenza di aggiornamento più elevata (60 Hz o 90 Hz), e l'utilizzo di marker realizzati con materiali più pregiati (come ad esempio della ceramica) che permettano una riflessione uniforme della luce infrarossa.

È possibile però apportare anche dei miglioramenti dal punto di vista software implementando la possibilità di usare, per le telecamere che ne possiedono la capacità, il sensore di profondità: in questo modo una volta individuata l'area di appartenenza del marker è possibile combinare le informazioni ricevute (infrarosse e di profondità) per incrementare la precisione di questa tipologia di calibrazioni.

Per concludere, questo elaborato ha trattato la calibrazione di un sistema di telecamere all'infrarosso tramite l'utilizzo di una wand alla quale sono stati applicati dei marker retroriflettenti. Per realizzare ciò sono stati creati due pacchetti ROS: *Find Markers* e *Wand Calibration*. Nel primo pacchetto è stato proposto un algoritmo che permette l'identificazione del centro dei marker e della posa della wand rispetto alla singola telecamera. Nel secondo pacchetto sono stati raccolti i dati prodotti dal primo ed è stata eseguita una calibrazione a coppie seguita da un algoritmo di ottimizzazione. Nonostante i risultati mostrino come non sia ancora possibile utilizzare il software sviluppato per calibrazioni accurate, tale esito è stato analizzato e da esso sono tratti dei miglioramenti da applicare in un prossimo futuro. Dunque questa tesi rappresenta un punto di inizio al fine di snellire il processo di calibrazione che allo stato corrente presenta molte difficoltà.

Bibliografia

[1] *Activision Blizzard — Home*. en. URL: https://www.activisionblizzard.com/ (visitato il 08/04/2022).

[2] *American Honda Motor Co., Inc. - Official Site*. en. URL: https://www.honda.com/ (visitato il 08/04/2022).

[3] Andrea Ancillao. «Analysis and Measurement of Human Motion: Modern Protocols and Clinical Considerations». In: *Journal of Robotics and Mechanical Engineering Research* 1.4 (dic. 2016), pp. 30–37. ISSN: 20594909. DOI: 10.24218/jrmer.2016.19. URL: http://verizonaonlinepublishing.com/ROBOTICSPDF/JournalofRoboticsandMechanicalEngineeringResearch19.pdf.

[4] Amazon AWS. *Presentazione di AWS RoboMaker, un nuovo servizio di robotica cloud*. 26 Nov. 2018. URL: https://aws.amazon.com/it/about-aws/whats-new/2018/11/announcing-aws-robomaker-a-new-cloud-robotics-service/ (visitato il 15/03/2022).

[5] *Azure Kinect DK – Develop AI Models — Microsoft Azure*. en. URL: https://azure.microsoft.com/en-us/services/kinect-dk/ (visitato il 08/04/2022).

[6] Filippo Basso, Riccardo Levorato e Emanuele Menegatti. «Online calibration for networks of cameras and depth sensors». In: *OMNIVIS: The 12th Workshop on Non-classical Cameras, Camera Networks and Omnidirectional Vision-2014 IEEE International Conference on Robotics and Automation (ICRA 2014)*. 2014.

[7] Gary Bradski e Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[8] Aurelio Cappozzo et al. «Human movement analysis using stereophotogrammetry: Part 1: theoretical background». en. In: *Gait & Posture* 21.2 (feb. 2005), pp. 186–196. ISSN: 0966-6362. DOI: 10.1016/j.gaitpost.2004.01.010. URL: https://www.sciencedirect.com/science/article/pii/S0966636204000256 (visitato il 06/04/2022).

[9] Lorenzo Chiari et al. «Human movement analysis using stereophotogrammetry: Part 2: Instrumental errors». en. In: *Gait & Posture* 21.2 (feb. 2005), pp. 197–211. ISSN: 0966-6362. DOI: 10.1016/j.gaitpost.2004.04.004. URL: https://www.sciencedirect.com/science/article/pii/S0966636204000682 (visitato il 06/04/2022).

[10] S. Corazza et al. «A Markerless Motion Capture System to Study Musculoskeletal Biomechanics: Visual Hull and Simulated Annealing Approach». In: *Annals of Biomedical Engineering* 34.6 (giu. 2006), pp. 1019–1029. ISSN: 0090-6964, 1573-9686. DOI: 10.1007/s10439-006-9122-8. URL: http://link.springer.com/10.1007/s10439-006-9122-8 (visitato il 06/04/2022).

[11] Stefano Corazza et al. «Markerless Motion Capture through Visual Hull, Articulated ICP and Subject Specific Model Generation». In: *International Journal of Computer Vision* 87 (mar. 2010), pp. 156–169. DOI: 10.1007/s11263-009-0284-3.

[12] Microsoft Docs. *Azure Kinect DK hardware specifications.* 23 Dic. 2021. URL: https://docs.microsoft.com/en-us/azure/kinect-dk/hardware-specification (visitato il 17/03/2022).

[13] Christian Fiedler et al. «A comparative study of automatic localization algorithms for spherical markers within 3D MRI data». In: *Brain Sciences* 11.7 (2021), p. 876.

[14] Ron Goldman. «Understanding quaternions». In: *Graphical Models* 73.2 (2011), pp. 21–49. ISSN: 1524-0703. DOI: https://doi.org/10.1016/j.gmod.2010.10.004. URL: https://www.sciencedirect.com/science/article/pii/S1524070310000172.

[15] *Google*. URL: https://www.google.com/ (visitato il 08/04/2022).

[16] O. Grau et al. «A Robust Free-Viewpoint Video System for Sport Scenes». In: *2007 3DTV Conference*. 2007, pp. 1–4. DOI: 10.1109/3DTV.2007.4379384.

[17] *IBM - Italia*. it-it. URL: https://www.ibm.com/it-it (visitato il 08/04/2022).

[18] *Intel — Soluzioni per data center, Internet delle cose e innovazione...* it. URL: https://www.intel.com/content/www/it/it/homepage.html (visitato il 08/04/2022).

[19] Brandon E Jackson et al. «3D for the people: multi-camera motion capture in the field with consumer-grade cameras and open source software». In: *Biology open* 5.9 (2016), pp. 1334–1342. DOI: https://doi.org/10.1242/bio.018713.

[20] Ole Kroeger, Johannes Huegle e Carsten A. Niebuhr. «An automatic calibration approach for a multi-camera-robot system». In: *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2019, pp. 1515–1518. DOI: 10.1109/ETFA.2019.8869522.

[21] APRIL Robotics Laboratory. *AprilTag*. URL: https://april.eecs.umich.edu/software/apriltag (visitato il 21/03/2022).

[22] Quoc V Le e Andrew Y Ng. «Joint calibration of multiple sensors». In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2009, pp. 3651–3658.

[23] Danylo Malyuta. *apriltag_ros*. 13 Mag. 2020. URL: http://wiki.ros.org/apriltag%5C_ros (visitato il 21/03/2022).

[24] Danylo Malyuta. «Guidance, Navigation, Control and Mission Logic for Quadrotor Full-cycle Autonomy». Inglese. Master thesis. 4800 Oak Grove Drive, Pasadena, CA 91109, USA: Jet Propulsion Laboratory, dic. 2017.

[25] MathWorks. *Correct fisheye image for lens distortion - MATLAB undistortFisheyeImage*. https://it.mathworks.com/help/vision/ref/undistortfisheyeimage.html. (Visitato il 06/04/2022).

[26] Ramona Michienzi et al. «Comparison of forensic photo-documentation to a photogrammetric solution using the multi-camera system "Botscan"». In: *Forensic Science International* 288 (2018), pp. 46–52. ISSN: 0379-0738. DOI: https://doi.org/10.1016/j.forsciint.2018.04.012. URL: https://www.sciencedirect.com/science/article/pii/S0379073818301695.

[27] Microsoft. *Azure Kinect ROS Driver*. URL: https://github.com/microsoft/Azure_Kinect_ROS_Driver (visitato il 04/04/2022).

[28] *Microsoft: cloud, computer, app e giochi*. it-it. URL: https://www.microsoft.com/it-it (visitato il 08/04/2022).

[29] Patrick Mihelich e James Bowman. *cv_bridge*. URL: http://wiki.ros.org/cv%5C_bridge (visitato il 18/03/2022).

[30] Jorge J Moré. «The Levenberg-Marquardt algorithm: implementation and theory». In: *Numerical analysis*. Springer, 1978, pp. 105–116.

[31] Nobuyasu Nakano et al. «Evaluation of 3D Markerless Motion Capture Accuracy Using OpenPose With Multiple Video Cameras». In: *Frontiers in Sports and Active Living* 2 (2020). ISSN: 2624-9367. URL: https://www.frontiersin.org/article/10.3389/fspor.2020.00050 (visitato il 06/04/2022).

[32] Ciaran O Conaire et al. «TennisSense: A platform for extracting semantic information from multi-camera tennis data». In: *2009 16th International Conference on Digital Signal Processing*. 2009, pp. 1–6. DOI: 10.1109/ICDSP.2009.5201152.

[33] OpenCV. *Intel acquires Itseez*. 27 Mag. 2016. URL: https://opencv.org/intel-acquires-itseez/ (visitato il 15/03/2022).

[34] OpenCV. *OpenCV: Home*. URL: https://opencv.org/ (visitato il 05/04/2022).

[35] OptiTrack. *OptiTrack - Motion Capture Systems*. URL: https://optitrack.com/ (visitato il 17/03/2022).

[36] OptiTrack. *Prime 41 - Our largest volume, highest resolution motion capture camera*. URL: http://optitrack.com/cameras/prime-41/index.html (visitato il 06/04/2022).

[37] The Bray Lab PALLS. *Nexus Motion Capture Lab Instructions*. URL: https://sites.tufts.edu/bray/nexus-motion-capture-lab-instructions/ (visitato il 06/04/2022).

[38] Morgan Quigley et al. «ROS: an open-source Robot Operating System». In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[39] ROS. *ROS - Robot Operative System*. URL: https://www.ros.org/ (visitato il 15/03/2022).

[40] Snigdha Roy et al. «High-Precision, Three-Dimensional Tracking of Mouse Whisker Movements with Optical Motion Capture Technology». In: *Frontiers in Behavioral Neuroscience* 5 (2011). ISSN: 1662-5153. DOI: 10.3389/fnbeh.2011.00027. URL: https://www.frontiersin.org/article/10.3389/fnbeh.2011.00027.

[41] *Sony Group Portal - Sony Group Corporation Website*. URL: https://www.sony.com/en/ (visitato il 08/04/2022).

[42] Gears Sports. *CW-500 Calibration Wand*. Giu. 2017. URL: https://www.gearssports.com/product/cw-500-calibration-wand/ (visitato il 06/04/2022).

[43] A. Sundaresan e R. Chellappa. «Markerless Motion Capture using Multiple Cameras». In: *Computer Vision for Interactive and Intelligent Environment (CVIIE'05)*. Lexington, KY, USA: IEEE, 2005, pp. 15–26. ISBN: 9780769525242. DOI: 10.1109/CVIIE.2005.13. URL: http://ieeexplore.ieee.org/document/1623766/ (visitato il 06/04/2022).

[44] Josip Tomurad e Marko Subašić. «Detection and localization of spherical markers in photographs». In: *CCVW* (2016).

[45] Ranjith Unnikrishnan e Martial Hebert. «Fast extrinsic calibration of a laser rangefinder to a camera». In: *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-05-09* (2005).

[46] Eline Van der Kruk e Marco Reijne. «Accuracy of human motion capture systems for sport applications; state-of-the-art review». In: *European Journal of Sport Science* 18 (mag. 2018), pp. 1–14. DOI: 10.1080/17461391.2018.1463397.

[47] Rodrigo Ventura. *camera_calibration/Tutorials/MonocularCalibration*. http://library.isr.ist.utl.pt/docs/roswiki/camera_calibration(2f)Tutorials(2f)MonocularCalibration.html. (Visitato il 06/04/2022).

[48] Vicon. *Calibration — Ensuring Precise Tracking For Your Optical System*. URL: https://www.vicon.com/hardware/devices/calibration/ (visitato il 17/03/2022).

[49] *Vicon — Award Winning Motion Capture Systems*. en-US. URL: https://www.vicon.com/ (visitato il 08/04/2022).

[50] John Wang e Edwin Olson. «AprilTag 2: Efficient and robust fiducial detection». In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, ott. 2016, pp. 4193–4198. ISBN: 978-1-5090-3762-9. DOI: 10.1109/IROS.2016.7759617.

[51] Xiaogang Wang. «Intelligent multi-camera video surveillance: A review». In: *Pattern Recognition Letters* 34.1 (2013). Extracting Semantics from Multi-Spectrum Video, pp. 3–19. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2012.07.005. URL: https://www.sciencedirect.com/science/article/pii/S016786551200219X.

[52] Ros Wiki. *Bags*. 15 Feb. 2022. URL: http://wiki.ros.org/Bags (visitato il 08/04/2022).

[53] Ros Wiki. *Messages*. 16 Ago. 2016. URL: http://wiki.ros.org/Messages (visitato il 08/04/2022).

[54] Ros Wiki. *Nodes*. 4 Dic. 2018. URL: http://wiki.ros.org/Nodes (visitato il 17/03/2022).

[55] Ros Wiki. *Topics*. 20 Feb. 2019. URL: http://wiki.ros.org/Topics (visitato il 18/03/2022).

[56] Fuzhen Zhang. «Quaternions and matrices of quaternions». In: *Linear Algebra and its Applications* 251 (1997), pp. 21–57. ISSN: 0024-3795. DOI: https://doi.org/10.1016/0024-3795(95)00543-9. URL: https://www.sciencedirect.com/science/article/pii/0024379595005439.

[57] Z. Zhang. «A flexible new technique for camera calibration». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334. DOI: 10.1109/34.888718.