UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"SVILUPPO DI UNA INTERFACCIA GRAFICA PER LA CALIBRAZIONE HAND-EYE DI SITEMI MULTI-TELECAMERA"

Relatore: Prof. Stefano Ghidoni

Laureando: Elia Parussolo

Correlatore: Davide Allegro

ANNO ACCADEMICO 2023 – 24

Data di laurea 24/09/2024

Indice

Abstract

La tesi inizia con un'introduzione al contesto della collaborazione uomo-robot e all'importanza della calibrazione Hand-Eye per la localizzazione precisa di oggetti e persone all'interno di un ambiente con robot. Successivamente, vengono esplorate le interfacce grafiche in generale, evidenziando gli elementi chiave e le caratteristiche di design, con attenzione alle differenze tra UX (User Experience) e UI (User Interface). L'elaborato procede con una rassegna di alcuni lavori esistenti (Related works) dedicati ai metodi di calibrazione Hand-Eye e ai software di calibrazione disponibili, come "easy_handeye" e "ATOM". Nella parte centrale la tesi presenta lo sviluppo di GIHMs, un'interfaccia grafica (GUI) progettata per semplificare il processo di calibrazione Hand-Eye di sistemi multi-telecamera. La GUI offre diverse funzionalità, tra cui la visualizzazione in tempo reale degli output delle telecamere, la comunicazione con il robot per il movimento programmato e la registrazione delle pose del braccio robotico. Vengono descritte l'architettura e le funzionalità del software, illustrandone le modalità d'uso e le librerie e framework utilizzati, come ROS (Robot Operating System) e Qt. La tesi si conclude con una discussione sulle sfide e le opportunità relative alla progettazione futura di nuove funzionalità di GIHMs, con una valutazione dei risultati già raggiunti.

1. Introduzione

Negli ultimi decenni, il rapido progresso tecnologico ha trasformato profondamente il modo in cui lavoriamo e interagiamo con l'ambiente che ci circonda. L'evoluzione della robotica e dell'intelligenza artificiale ha aperto la strada a nuove forme di collaborazione tra esseri umani e macchine, ridefinendo i confini dell'innovazione e della produttività. Mentre i robot offrono precisione, velocità e resistenza in compiti ripetitivi o complessi, gli esseri umani forniscono flessibilità, creatività e capacità decisionali. In questo contesto, la collaborazione uomo-robot rappresenta una delle sfide e delle opportunità più significative, lavorando insieme in modo sinergico e combinando le capacità uniche di entrambi non migliora solo l'efficienza operativa, ma anche la sicurezza e la qualità del lavoro svolto.

Il robot è un manipolatore programmabile multiscopo per la movimentazione di materiali, di attrezzi e altri mezzi di produzione, capace di interagire con l'ambiente nel quale si svolge il ciclo tecnologico di trasformazione relativo all'attività produttiva (Società Italiana di Robotica Industriale - SIRI). Nel corso di questa tesi con il termine "robot" indicheremo, come nella definizione sopra citata, un braccio meccanico [S11].

Figura 1.1: Esempio di robot che sarà oggetto in questa trattazione, il Braccio robotico Franka Research 3 [S2]

In un sistema dotato di robot, la calibrazione Hand-Eye è il processo di stima delle trasformazioni geometriche tra sensori di immagine e visione, come telecamere RGB e sensori di profondità, sensori di distanza e prossimità, come i LiDAR[1] (Light Detection And Ranging) e il braccio robotico. Questa operazione consente di localizzare elementi tra cui oggetti e persone nella cella

[1] Il LiDAR è una tecnologia che misura la distanza di un oggetto illuminandolo con una luce laser, è in grado di fornire informazioni tridimensionali ad alta risoluzione sull'ambiente circostante. È una tecnica simile a quella radar basata sul principio dell'eco, solo che utilizza come tramite la luce pulsata anzichè un segnale radio [S1].

robotica rispetto ad un sistema di riferimento unico. È fondamentale e necessaria in un'ampia gamma di applicazioni robotiche, dalla “semplice” localizzazione e manipolazione di oggetti, alla chirurgia assistita da robot, i veicoli autonomi e le operazioni industriali in cui la collaborazione uomo-robot è essenziale.

Nella realizzazione di un ambiente in cui possa avvenire una collaborazione uomo-robot sicura ed efficace, in atto al Intelligent Autonomous Systems Laboratory (IAS-Lab) dell’università di Padova, uno dei laboratori del dipartimento di ingegneria dell’informazione (DEI), mi è stato assegnato il compito di realizzare un’interfaccia grafica (GUI - Graphical User Interface) in grado di semplificare l’operazione di calibrazione di un robot monitorato da una rete di telecamere (vedi figura 1.2).

Figura 1.2: Foto del setup di laboratorio in cui si possono distinguere un braccio robotico monitorato da una rete di telecamere, ciascuna posizionata sull’asta di un treppiede [B9]

Attualmente alcune problematiche dei software di calibrazione Hand-Eye, come il supporto limitato per telecamere multiple e la gestione inefficiente dei tempi di esecuzione, non permettono di effettuare l’operazione di calibrazione in maniera efficiente nel laboratorio. Il nuovo applicativo, disponibile alla pagina github.com/Parelia/GIHMs.git, sopresisce a queste mancanze, gestendo gli output di più telecamere e possiede un sistema di raccolta dati efficiente e veloce. Inoltre verrà integrato con uno script già realizzato in “Multi-Camera Hand-Eye Calibration for Human-Robot Collaboration in Industrial Robotic Workcells” [B9].

2. Interfacce grafiche

Iniziamo fornendo una definizione generale: in informatica, un'interfaccia è il punto di interazione tra due sistemi, dispositivi o programmi software, attraverso il quale possono interagire e scambiarsi informazioni. Questa operazione avviene grazie alla definizione di un insieme di regole e protocolli che vanno ad indicare come i dati debbano essere scambiati e come si possano richiamare eventuali funzioni e servizi.

Esistono diverse tipologie di interfacce, tuttavia, in oggetto di questa trattazione saranno solamente le interfacce utente (UI - User Interface), quelle che fungono da mezzo per l'utente nell'interazione con un programma, sia esso un sistema operativo o un software applicativo. Un'interfaccia di questa categoria può essere testuale (CLI - Command Line Interface), come il prompt dei comandi di windows o il terminale (shell) di linux, dove l'utente interagisce con il sistema tramite comandi di testo; o può essere grafica (GUI - Graphical User Interface), come gli stessi sistemi operativi, dove elementi visivi comunicano le "affordance", ovvero, le possibili azioni che l'utente può compiere [B6].

L'utilizzo delle interfacce grafiche, come gli stessi sistemi operativi, è indispensabile nei dispositivi moderni, dai computer desktop agli smartphone, e testimonia la loro importanza nel rendere la tecnologia accessibile e utilizzabile da tutti.

2.1 Elementi di una interfaccia grafica

L'interfaccia grafica (GUI) non è da confondersi con l'interfaccia utente (UI), di fatto, possiamo dire che sia composta da due componenti principali: l'UX (User Experience - esperienza utente) e l'UI stessa. Una moderna GUI progetta a livello di design la UI[2], e fa quindi parte del suo sottoinsieme orientato all'esperienza utente [S7].
Sistemi in solo testo, come le CLI, possono essere difficili da usare, specialmente per persone non esperte; una GUI deve, quindi, fornire elementi visivi che permettano agli utenti di navigare nell'applicazione, accedere alle sue funzionalità ed eseguire le attività in modo intuitivo e di facile comprensione, con input minimi da tastiera.

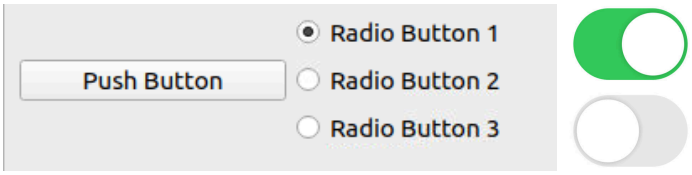
Quando si parla di elementi visivi facenti parte di un'interfaccia grafica, ci si riferisce ad una vasta serie di strutture, tra queste, le principali verranno trattate brevemente.
Chiamiamo "finestra" (window) un elemento visivo che rappresenta una porzione di spazio (content area) all'interno della quale l'utente può visualizzare ed interagire con contenuti o applicazioni.

[2] Il significato di questa frase sarà chiarito nella sezione Grafica e design 2.2, a pagina 7

Generalmente, una finestra contiene alcuni componenti standard, come una sezione superiore in cui viene inserito un titolo (title bar), dei bordi, una barra di scorrimento (scroll bar) e pulsanti di controllo (control buttons) con la funzione di minimizzare o massimizzare la sua dimensione o di chiuderla (vedi figura 2.1). Le finestre hanno anche la funzionalità di essere sovrapposte, ridimensionate e spostate, consentendo all'utente di gestire con più contenuti o applicazioni simultaneamente; è una caratteristica importante perché facilita l'interazione e la navigazione all'interno di un sistema operativo o di un software [B6].

Figura 2.1: Finestra di esempio (senza contenuto), in cui si possono distinguere le sue componenti: la barra del titolo (title bar) in alto, i bordi, una barra di scorrimento (scroll bar) sulla destra e i pulsanti di controllo (control buttons) nell'angolo in alto a destra

Tra le altre componenti ci sono pulsanti (buttons) di vario tipo, come quelli sollevati (push buttons), i "classici" da premere, quelli radio (radio buttons), che permettono di selezionare una sola opzione di un gruppo (set), o ancora, quelli di attivazione (toggle buttons). Ciascun tipo di pulsante è in grado di eseguire una specifica, ed unica, azione o di alterare le dinamiche di utilizzo (come rendere disponibile o meno una funzione del software, nel caso di quelli toggle) [S8].



(a) (b) (c)

Figura 2.2: Esempi di diverse tipologie di pulsanti, push buttons(a), radio buttons (b) e toggle buttons (c)

Esistono poi i menù e gli elenchi (combo box), entrambi compattano diverse opzioni a disposizione dell'utente in una raccolta, ma con tipologie di oggetti differenti, infatti nel primo caso, saranno azioni eseguibili dal software, nel secondo, elementi da selezionare; le caselle (box), possono essere di testo (textbox) o etichette (label), a seconda della loro capacità, caselle a scorrimento (spin box), di solito contenenti valori numerici, ed anche caselle a spunta (check box).

(a) (b) (c) (d) (e)

Figura 2.3: Esempi di diversi elementi di un'interfaccia grafica, in particolare, un menù (a), un elenco (combo box) (b), un'etichetta (label) (c), una casella a scorrimento (spin box) (d) e una casella a spunta (check box) (e).

Infine, è possibile contrassegnare ed identificare gli oggetti non solo con una semplice stringa di testo, ma anche assegnandogli delle immagini, dette "icone", che molto spesso indicano meglio all'utente il loro scopo (vedi figura 2.4).

Figura 2.4: Esempio di icone che possiamo trovare sui pulsanti di un player audio, in ordine, da sinistra a destra: precedente (previous), indietro (backwards), avvia (play), avanti (forwards), successivo (next), coda di ascolto (queue), stop (esci), pausa (pause), aggiungi/rimuovi dai preferiti (favourite) e ricomincia (rewind) [S12]

Le GUI, grazie alla loro semplicità d'uso puntano a fornire ad utenti non esperti un'interazione più diretta e visuale con il software, rispetto a quelle a riga di comando, tuttavia, queste mantengono un ruolo importante in ambiti in cui è necessario un maggiore controllo sull'esecuzione dei vari programmi (precisione nei comandi ed efficienza nella gestione delle informazioni sono cruciali).

Un'interfaccia utente ben progettata dovrebbe puntare a caratteristiche come: l'usabilità, ovvero rendere più facile l'utilizzo e l'apprendimento e comprensione del software; l'aspetto estetico, il quale non sempre rende un prodotto più piacevole da utilizzare, ma elementi di progettazione visivamente accattivanti, come animazioni, transizioni e grafica personalizzata che rendono l'applicazione più coinvolgente; la presentazione delle informazioni, quindi, struttura e segnali visivi senza l'obiettivo di rendere più piacevole il software, ma che puntano a presentare le informazioni complesse in un modo più comprensibile rendendole più assimilabili (particolarmente importante per le applicazioni di monitoraggio e controllo, in cui gli operatori devono essere in grado di processare rapidamente grandi quantità di dati ed effettuare le dovute regolazioni); il feedback e l'orientamento di utilizzo, per fare in modo che l'utente sia in grado di individuare le scorciatoie, informato sulle conseguenze di ciascuna azione e sullo stato di avanzamento (come barre di caricamento, messaggi di stato, ecc…); la personalizzazione, consiste nel permettere agli utenti di modificare l'aspetto dell'interfaccia per soddisfare le loro esigenze, adattandola ai diversi casi d'uso.

Deve essere mantenuta una certa coerenza tra il design e il comportamento dell'interfaccia, andrebbero mantenuti gli stessi elementi, lo stesso linguaggio e le stesse convenzioni in tutto il sistema, in questo modo, gli utenti non devono imparare nuove convenzioni ogni volta che interagiscono con una parte diversa del sistema. L'interfaccia deve fornire agli utenti un feedback visivo o uditivo chiaro e conciso sulle loro azioni, sullo stato del sistema e su qualsiasi errore che si verifica, ma soprattutto prevenire dal commettere quest'ultimi; questo può essere ottenuto fornendo, oltre ai feedback, istruzioni di utilizzo chiare e definendo vincoli per impedire agli utenti di compiere azioni non valide, l'esperienza utente risulterà così più fluida e meno frustrante [B8].

2.2 Grafica e design

Il termine "design" può assumere diverse sfaccettature e talvolta creare confusione, tuttavia, una delle definizioni che più si addice al ruolo che possiede nel contesto delle interfacce utente, come disse Alina Wheeler, è "Il design è l'intelligenza resa visibile". Infatti, quando diciamo che per realizzare una buona interfaccia grafica è necessario svolgere anche un adeguato studio di design, alludiamo alla ricerca, all'accortezza e alla valorizzazione dei componenti tale da confezionare un software di livello utilizzabile da chiunque. Tra le filosofie di design più condivise troviamo anche quella di Alan Cooper, che disse "Non importa quanto sia bello, non importa quanto sia interessante la tua interfaccia, sarebbe meglio se ce ne fosse di meno" [S6].
Per ciò, quando si sviluppa una GUI, occorre essere orientati alla creazione di interfacce semplici con elementi di design facilmente comprensibili e di layout ad hoc, in cui ogni elemento svolge una funzione chiara e precisa [S7].

Scendendo nel dettaglio dei ruoli di una GUI, ricordando la divisione tra UX e UI, possiamo dire che all'esperienza utente (UX) compete l'interazione complessiva con il software, quali siano gli utilizzi per cui è richiesto, concentrandosi sul migliorare aspetti come l'usabilità, l'accessibilità e la soddisfazione dell'utilizzatore. L'obiettivo dell'UX design è di creare interazioni fluide e intuitive con gli elementi a schermo. Una delle tecniche utilizzate per migliorare l'esperienza con il software è l'User Journey Mapping, ovvero, tracciare il percorso che un utente deve seguire per raggiungere un certo obiettivo. Per esempio, se in un'app bancaria l'utente vuole visualizzare il proprio saldo, dovrà prima accedere all'app, poi navigare alla sezione "conto corrente" e da lì visualizzarlo. Mappare tutti i singoli passaggi da compiere aiuta, infatti, ad identificare i punti critici e a migliorare la fluidità di utilizzo.

L'interfaccia utente (UI), invece, riguarda l'aspetto visivo dell'interfaccia, come questa appare all'utilizzatore, il lato a suo servizio (user friendly). Il design dell'UI si focalizza sulla coerenza

visiva, assicurando che ogni elemento sia funzionale e comprensibile, infatti, una finestra ben organizzata risulterà molto più facile da comprendere agli occhi di un utente. Tra le tecniche UI che influenzano positivamente il lato UX troviamo l'utilizzo di ombre, grazie alla loro capacità di fornire tridimensionalità, vengono inserite solo su determinati oggetti, come i pulsanti da cliccare, i quali risultano così in "rilievo" indicando inconsciamente all'utente la possibilità di interagire con loro. Un altro esempio è la modifica dei colori, infatti, per comunicare che un pulsante, in un certo momento, non può essere premuto, possiamo attenuarne i colori ottenendo un effetto "velo" che diffida l'utente dal premerlo (chiaramente, anche il codice del programma non consentirà questa azione) [B7].

Insieme, UX e UI lavorano sinergicamente per creare un'interfaccia che sia chiara e intuitiva, non solo piacevole alla vista, ma anche semplice da usare.

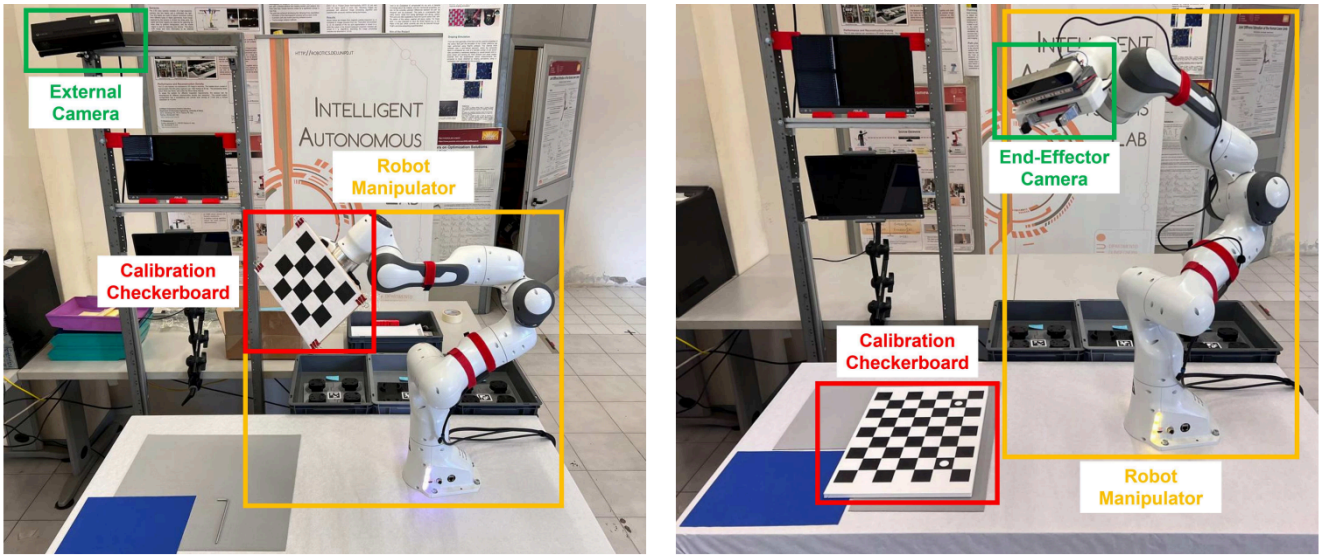
2.3 Storia delle interfacce grafiche

La storia delle interfacce grafiche ha segnato una svolta nell'interazione tra utenti e computer, inizia negli anni '60 grazie alle geniali intuizioni di Douglas Engelbart, un ingegnere americano, che sviluppò il concetto di interazione con un computer attraverso il mouse presso lo Stanford Research Institute (SRI); nel 1968, con il suo team, completò la famosa dimostrazione nota come "Mother of All Demos", basata sul "NLS" (oN-Line System), dove il movimento di un piccolo un piccolo box rettangolare con tre pulsanti, connesso ad un computer tramite filo, si traduceva nel movimento a schermo di un puntatore che localizzava una specifica area sullo schermo, segnando la nascita del mouse. Negli anni '70, il nuovo centro di ricerca Xerox PARC (Palo Alto Research Center) in California, avviò lo sviluppo del Xerox Alto, il primo computer a introdurre i concetti di finestre, icone e menu a discesa, gettando le basi delle moderne GUI. Tuttavia, lo Xerox Alto non fu mai commercializzato, ma il suo successore Xerox Star del 1981 implementò molte delle sue idee, influenzando notevolmente i computer futuri anche avendo scarso successo commerciale. Il concetto di GUI raggiunse il grande pubblico con l'Apple Macintosh nel 1984, che si ispirò ai lavori di Xerox PARC per creare un'interfaccia user-friendly e un mouse per interagire col sistema, anche se un anno prima Apple aveva lanciato il Lisa, il primo vero computer dell'azienda con una GUI. Nel 1985 Microsoft lanciò la prima versione di Windows, un'interfaccia grafica che operava sopra il sistema operativo MS-DOS, inizialmente rudimentale, ma che subirà una rapida evoluzione. Negli anni '90 le interfacce grafiche diventarono lo standard, con Windows e Mac OS a guidare il mercato grazie a GUI sempre più sofisticate, culminate in prodotti come Windows 95 e, alla fine del decennio, Mac OS 9 [S10].

Le prime realizzazioni di software grafici, com'è accaduto nei primi tempi della diffusione delle interfacce sia su Windows che su Linux, svolgevano solamente il ruolo di "wrapper" (involucro), ovvero, manovravano uno o più programmi in grado di operare esclusivamente da CLI, come uno strato superiore (grafico) della stessa, senza sostituirla direttamente. Le interfacce grafiche si sono poi evolute nel tempo, diventando sempre più sofisticate, complesse e divenendo anche oggetto di design.

3. Related works

Esistono due metodi per posizionare una telecamera rispetto ad un robot, ciascuno con applicazioni specifiche. Il primo è l'eye-in-hand, dove la telecamera è montata direttamente sul braccio del robot, inquadrando il "punto di intervento" del robot (vedi figura 3.1 (b)), il secondo è l'eye-on-base, dove la telecamera è fissa e posizionata separatamente al robot, solitamente alla sua base o in un punto esterno, offrendo una visione statica del robot e dell'ambiente che lo circonda (vedi figura 3.1 (a)).



(a) (b)

Figura 3.1: Si mostrano la configurazione eye-on-base (a) e eye-in-hand (b) [S5]

Per consentire ad un braccio robotico di afferrare un oggetto, infatti, è necessario che il sistema conosca con precisione la posizione relativa dei sensori rispetto alla base dello stesso, eseguendo una giusta pianificazione del movimento e completando le varie operazioni evitando gli ostacoli.

3.1 Metodi di calibrazione Hand-Eye

In robotica, l'operazione chiamata calibrazione Hand-Eye ha lo scopo di determinare la matrice di trasformazione omogenea tra il sistema di coordinate di una telecamera e il sistema di coordinate dell'end-effector (mano) del robot. Il processo prevede la stima della suddetta matrice attraverso vari fotogrammi che inquadrano un pattern di riconoscimento che può essere momentaneamente installato sulla mano del robot, o posizionato su una superficie, a seconda del posizionamento della telecamera (eye-in-hand o eye-on-base). Inoltre, questo pattern di calibrazione varia a seconda del software utilizzato, può essere infatti sia una "semplice" scacchiera che un disegno più complesso, (vedi figura 3.2).

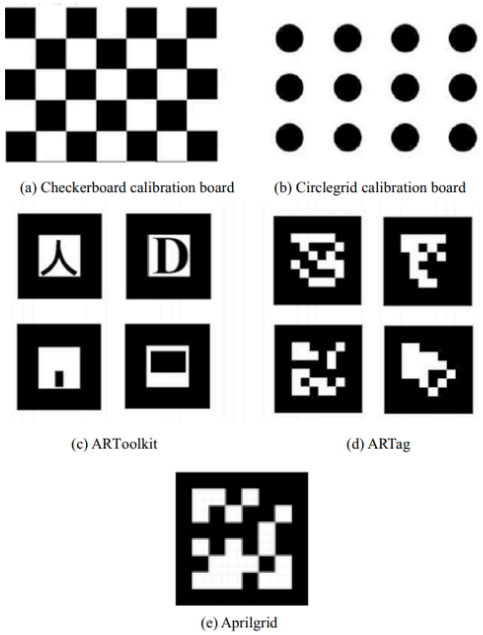


Figura 3.2: Esempi di pattern di calibrazione, in questo caso sono riportati quelli del tool easy_handeye [B4]

È giusto sottolineare che con "varie immagini", quando parliamo di sistemi che inquadrano un robot (eye-on-base, vedi seguito), intendiamo fotogrammi ciascuno raffigurante una diversa posa dello stesso, la giusta assegnazione posa-immagine è cruciale alla validazione dei risultati.

I tool Hand-Eye, per svolgere il proprio compito, devono affrontare diverse sfide come l'accuratezza della stima della posa, la robustezza al rumore nei dati del sensore e la necessità di procedure di calibrazione efficienti. Queste, oltre che sul miglioramento dell'accuratezza e dell'affidabilità dei tool stessi, sono tutte problematiche su cui la ricerca odierna si concentra.

3.1.1 Limiti degli approcci tradizionali hand-eye

Alcuni dei limiti principali che si possono individuare negli approcci tradizionali di calibrazione Hand-Eye sono l'incapacità di gestire sistemi con più sensori e la loro funzionalità ristretta a specifiche tipologie di sensori.
Gli approcci tradizionali spesso affrontano la calibrazione come una serie di calibrazioni a coppie (ogni coppia di sensori viene calibrata individualmente), è una modalità che non si adatta bene a sistemi complessi con più sensori. Infatti, oltre ad aumentare la complessità dell'operazione e il numero di interazioni totali, l'approccio sequenziale può portare a un accumulo di errori e potrebbe non catturare le relazioni tra tutti i sensori simultaneamente.
Sempre gli approcci tradizionali spesso sono progettati per specifici sensori, come una determinata telecamera RGB, il che li rende difficili da adattare ad altre modalità di sensori senza modifiche

sostanziali. Questa mancanza di generalità limita la loro applicabilità in scenari robotici moderni in cui sistemi multi-sensore e multimodali stanno diventando sempre più diffusi.

Tra le problematiche che possono impattare sull'efficienza di questi software c'è l'acquisizione dati incompleta o imprecisa a causa di un percorso di misurazione ostruito. Questo problema si verifica quando componenti fisici bloccano la linea visiva tra telecamere e robot o interferiscono con la capacità del sensore di effettuare misurazioni precise (come riflessi, ombre ecc…). Inoltre, i fotogrammi ottenuti possono non essere validi a causa di una scorretta inclinazione della telecamere o del pattern del robot. Infatti, alcune pose del robot potrebbero ridurre i pixel a favore di camera (anche a zero, nel caso in cui questo si rivolgesse nella direzione opposta), rendendo quindi molto più difficile effettuare il riconoscimento del pattern e complicando la procedura di calibrazione [B3].

3.1.2 Algoritmi di calibrazione

Diversi algoritmi si rivelano particolarmente utili nell'ambito della calibrazione in robotica, di seguito ne verranno trattati brevemente alcuni tra i più utilizzati.

- Il Simulated Annealing, è un algoritmo di ottimizzazione ispirato al processo fisico di ricottura dei metalli, in cui un materiale viene riscaldato e poi lentamente raffreddato per raggiungere uno stato energetico più stabile. In termini di ottimizzazione, anche questo algoritmo cerca di trovare il minimo globale di una funzione di costo, tuttavia, adotta una particolare strategia per evitare di rimanere intrappolato nei minimi locali: una soluzione corrente viene man mano aggiornata mentre si esplora lo spazio di ricerca e, in base ad una probabilità determinata dalla differenza tra la funzioni di costo e un parametro chiamato temperatura, vengono accettate nuove soluzioni. Inizialmente, la temperatura è alta, permettendo all'algoritmo di accettare anche peggioramenti nella funzione di costo (soluzioni, in teoria, peggiori), mentre, con il passare del tempo, la temperatura si abbassa, riducendo la probabilità di accettare soluzioni peggiori e concentrandosi invece su miglioramenti.
- L'ottimizzazione a sciame di particelle (Particle Swarm Optimization - PSO) è un algoritmo ispirato al comportamento collettivo degli sciami naturali, come stormi di uccelli o banchi di pesci, di fatto, opera gestendo un gruppo di soluzioni candidate chiamate "particelle". Ogni particella aggiorna la sua posizione tenendo conto della miglior soluzione da lei trovata e la miglior soluzione all'interno dello sciame, cercando un equilibrio tra provare nuove soluzioni e migliorare quelle già esistenti.

- L'algoritmo di Powell è un metodo di ottimizzazione numerica che non richiede il calcolo delle derivate, il che lo rende adatto a problemi in cui la funzione obiettivo non è facilmente derivabile (minimizzazione di funzioni non lineari). Sviluppato da Michael J.D. Powell, è progettato per problemi multidimensionali ed opera effettuando una ricerca direzionale iterativa per individuare il minimo di una funzione di costo; l'algoritmo si sposta, infatti, lungo una serie di direzioni (chiamate direzioni coniugate) sulle quali effettua delle ricerche unidimensionali, inoltre, ad ogni nuova iterazione le stesse vengono aggiornate per esplorare meglio lo spazio di ricerca.
- Il metodo Levenberg-Marquardt, viene utilizzato per risolvere problemi di ottimizzazione non lineari, in particolare, nei casi che coinvolgono funzioni di costo che possono essere espresse come somma dei quadrati residui, come spesso accade negli scenari di calibrazione. La somma dei quadrati residui (Residual Sum of Squares - RSS), nell'analisi di regressione, serve a valutare la bontà di adattamento di un modello (un indice dell'errore complessivo), infatti, è la somma dei quadrati delle differenze (o residui) tra i valori osservati e i valori stimati o previsti da un modello statistico.
- Il Kalman Filter (KF) è un algoritmo di stima iterativa che fornisce una previsione ottimale dello stato di un sistema dinamico e viene utilizzato principalmente quando si ha un sistema che può essere descritto da modelli lineari e rumorosi. Il KF opera in due fasi: una fase di previsione, in cui viene stimato lo stato del sistema, e una fase di aggiornamento, dove la stima viene corretta basandosi sulle nuove informazioni provenienti dai sensori. Questo processo iterativo permette di ottenere stime molto precise anche quando i dati osservati sono affetti da rumore, per questo, è ampiamente impiegato nella robotica per la localizzazione e la navigazione di robot mobili, poiché consente di fondere informazioni provenienti da vari sensori, come GPS e accelerometri, migliorando la precisione della posizione e del movimento del robot.
- L'Extended Kalman Filter (EKF) è una variante del Kalman Filter che può gestire sistemi non lineari, per farlo, effettua una linearizzazione locale del sistema applicando una specifica matrice, la Jacobiana, i quali elementi sono tutte le derivate parziali prime della funzione a cui viene applicata. Questo algoritmo è molto utile per risolvere problemi come la localizzazione e la mappatura simultanea (Simultaneous localization and mapping - SLAM), dove il robot deve costruire una mappa dell'ambiente sconosciuto mentre ne stima la propria posizione. La capacità di trattare modelli non lineari rende l'EKF particolarmente adatto a scenari più complessi, come la navigazione di droni e veicoli autonomi in ambienti irregolari o con sensori imprecisi.

- Il RANSAC (Random Sample Consensus) è un algoritmo utilizzato per stimare parametri di un modello matematico in presenza di dati rumorosi o contenenti outlier (valori anomali che non seguono il comportamento tipico della maggior parte degli altri dati), è utile in contesti in cui una parte significativa dei dati è corrotta o non rappresenta il modello corretto. Il suo approccio consiste nel selezionare casualmente piccoli sottoinsiemi di dati, stimare un modello sugli stessi e valutare quanti altri punti si adattino a quel modello che verranno detti, di conseguenza, inliers. Ripetendo questo processo più volte, l'algoritmo trova il modello che meglio si adatta alla maggior parte dei dati, ignorando gli altri (outlier). Grazie alla sua robustezza, questo algoritmo è spesso impiegato per la calibrazione di telecamere e sensori o per risolvere problemi di corrispondenza di caratteristiche visive nei sistemi SLAM.

3.2 Software di calibrazione

Attualmente esistono diversi software di calibrazione, tra i più utilizzati troviamo easy_handeye (sezione 3.2.1) e Atom (sezione 3.2.2).

3.2.1 easy_handeye

Questo software, utilizzato anche in laboratorio, è stato progettato per monitorare l'interazione di un braccio meccanico con una singola telecamera, sia nella configurazione eye-in-hand che nella configurazione eye-on-base. Pur non essendo particolarmente ricco di funzionalità avanzate, è in grado di elaborare in tempo reale la posizione relativa tra telecamera e braccio robotico quando quest'ultima inquadra il pattern selezionato. easy_handeye supporta diversi pattern (vedi figura 3.1) e i dati visivi raccolti vengono visualizzati tramite RViz (vedi pagina 25) in una riproduzione spaziale 3D del braccio robotico, fornendo così anche una rappresentazione digitale del processo (vedi figura 3.3).

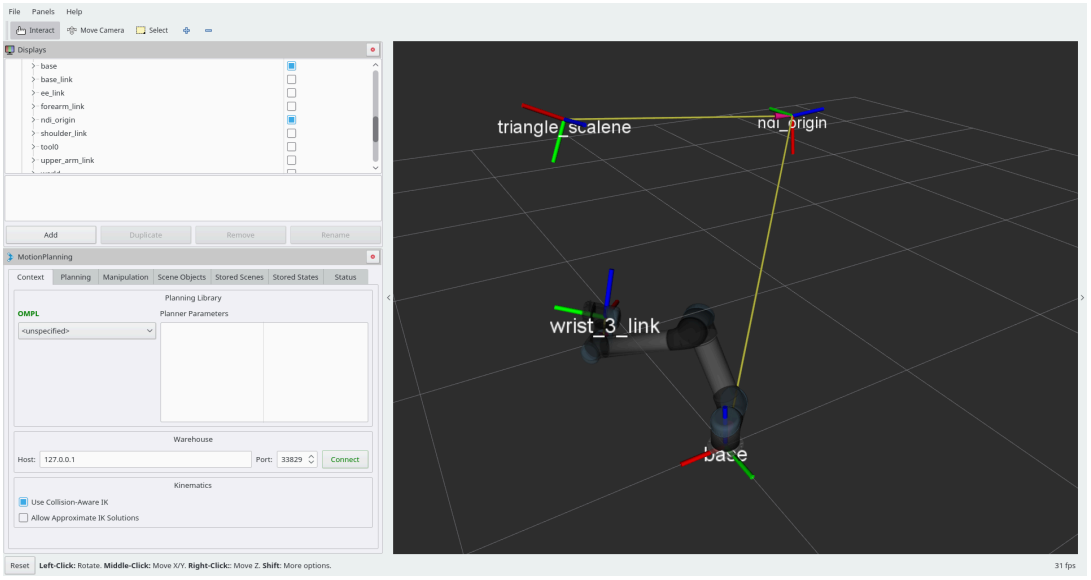


Figura 3.3: La finestra RViz di easy_handeye per riprodurre in 3D il robot e il sensore (ndi_origin) [S3]

Il software permette di filtrare e manipolare i campioni acquisiti, selezionando solo quelli necessari, anche se non tiene conto della posa assunta dal robot (funzione che invece sarà richiesta al mio software). Il suo limite principale di easy_handeye è la stessa gestione di una singola telecamera per volta, in applicazioni più complesse, che richiedono una visuale più completa o multi-angolo, potrebbe essere necessaria una configurazione multi-telecamera (altra caratteristica che il software da sviluppare dovrà integrare) [B4].

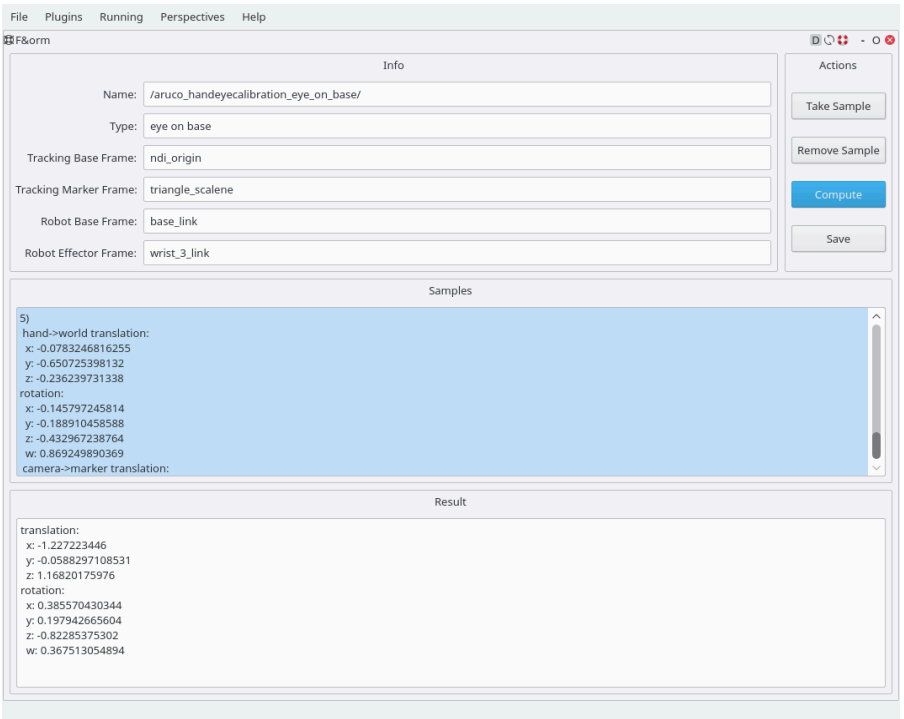


Figura 3.4: Menù mostrato da easy_handeye durante la fase di acquisizione di campioni [S3].

3.2.2 ATOM (Atomic Transformations Optimization Method)

ATOM è un insieme di strumenti di calibrazione per sistemi robotici multi-sensore e multi-modali, basato sull'ottimizzazione delle trasformazioni atomiche fornite da una descrizione del robot basata su ROS (Robot Operating System - vedi pagina 22). Fornisce diversi script per facilitare tutte le fasi della procedura di calibrazione e supera una delle limitazioni degli approcci tradizionali Hand-Eye gestendo anche sistemi complessi con più sensori e modalità.

La gestione di più telecamere è possibile grazie alla sua capacità di ottimizzare le pose di tutti i sensori contemporaneamente (considerando le loro relazioni interdipendenti) utilizzando un paradigma sensore-pattern (sensor-pattern paradigm), cioè, i sensori non vengono calibrati l'uno rispetto all'altro, ma rispetto a uno schema (pattern) di calibrazione comune. In questo modo, la posa (la posizione con orientamento nello spazio) del modello di calibrazione e quella dei singoli sensori vengono stimate simultaneamente rispetto al pattern comune, questo funge così da punto di riferimento fisso per tutti i sensori, semplificando il processo rispetto ai metodi tradizionali sensore-sensore (sensor-to-sensor).

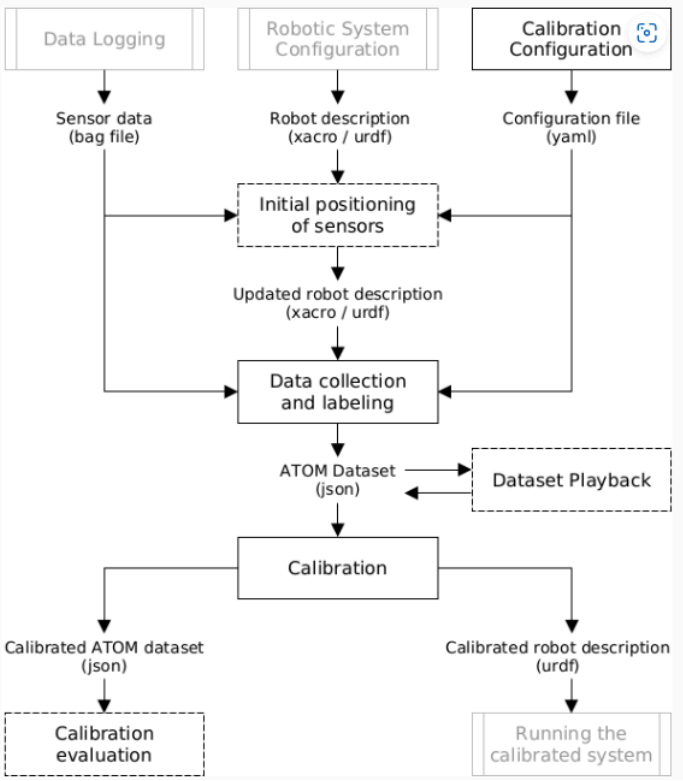


Figura 3.5: Schema che riassume la procedura di calibrazione eseguita da ATOM (calibration pipeline) [B2]

ATOM può gestire anche scenari in cui non tutti i sensori rilevano il pattern di calibrazione in ogni frame o istanza temporale, questo, è particolarmente importante per sistemi con più sensori in cui alcuni campi visivi si sovrappongono. Utilizzando un approccio di ottimizzazione basato sulla

somma dei quadrati degli errori, per il quale un sensore che non rileva il pattern in una particolare raccolta di dati viene escluso, è possibile lavorare anche con set di dati incompleti e fornire comunque stime di calibrazione accurate. ATOM è progettato per essere robusto alle stime iniziali per le pose dei sensori, grazie all'uso di un algoritmo di ottimizzazione basato sul gradiente, il Reflective Trust Region algorithm, che appartiene alla famiglia degli algoritmi della trust region, che controllano la dimensione dei passi dell'algoritmo per garantire che rimangano in una "regione affidabile" dove le approssimazioni siano valide. Il termine "reflective" si riferisce al modo in cui l'algoritmo gestisce le variabili che devono rispettare determinati vincoli [B1] [B2].

4. Sviluppo dell'interfaccia grafica per la calibrazione hand-eye di una rete di telecamere

Per poter facilitare il processo di calibrazione Hand-Eye di una rete di telecamere, migliorando la procedura macchinosa del laboratorio, in cui ciascun passaggio va eseguito manualmente e per ciascuna telecamera su diversi terminali, era necessario implementate diverse funzionalità. Tra queste, poter visualizzare l'output video di ciascuna telecamera e salvarne le immagini, poter comunicare con il robot facendolo muovere, e salvare le pose del braccio robotico in file di configurazione YAML. In più, il software doveva integrare, come già detto, uno script di calibrazione già precedentemente realizzato in "Multi-Camera Hand-Eye Calibration for Human-Robot Collaboration in Industrial Robotic Workcells" [B9].

Il compito effettivo del nuovo applicativo consisteva nel automatizzare il processo di ottenimento ed organizzazione delle immagini, facendole corrispondere alla posa assunta dal robot. In particolare, era necessario predisporre del materiale per il lancio dello script, organizzando una cartella di progetto in cui posizionare in apposite sottocartelle un file con delle specifiche di avvio ("CalibrationInfo.yaml" in figura 4.1), le foto delle varie telecamere (varie "Camera" in figura 4.1) e le pose del robot ("dataset" in figura 4.1).

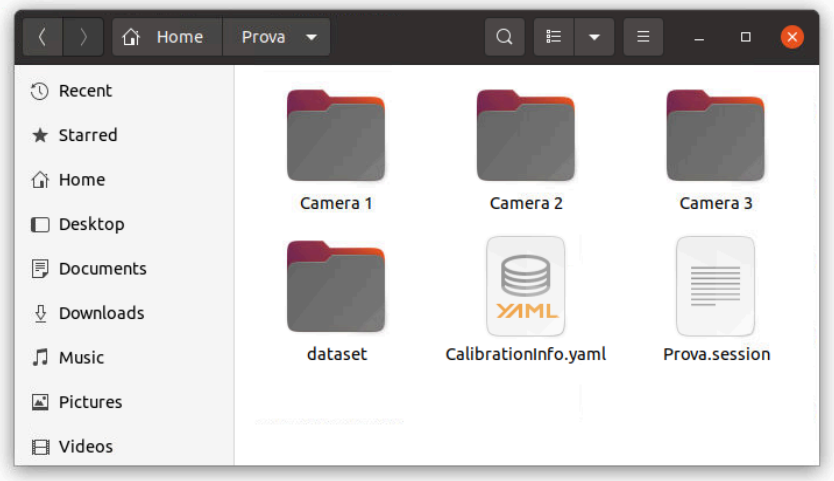


Figura 4.1: Esempio di struttura di una cartella di progetto, in questo caso, quella di "Prova" in cui le telecamere della rete sono tre

Il software progettato per soddisfare le richieste è stato chiamato "GIHMs, a simple Graphical User Interface for Hand-Eye Calibration of Multi-Camera Systems".

4.1 GIHMs

Realizzato in C++ per mezzo del framework Qt ed integrato nei sistemi ROS, necessario per interagire con i vari dispositivi elettronici, GIHMs vuole essere uno strumento semplice ed efficace per compiere la calibrazione Hand-Eye di un sistema multi telecamera.

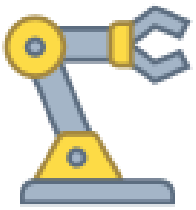


Figura 4.2: Icona di GIHMs

L'idea di sviluppo ha portato alla creazione di un software che opera per "sessioni", ovvero, cartelle strutturate in modo da rispettare quelle dello script, nelle quali vengono salvati tutti i dati. In particolare, una cartella viene ritenuta tale (sessione) e validata, grazie alla presenza di un file ".session" che ne riprende il nome. All'interno di questo, strutturato come un file di configurazione YAML, vengono salvate tutte le impostazioni relative alla prova in corso. L'inserimento di questo file di controllo, offre così la possibilità di manipolare il suo contenuto e quello della cartella anche in un secondo momento (inteso diverso da quello di acquisizione), permettendo di dividere un eventuale test in più fasi o di aggiornare o di rivedere una vecchia prova.

L'intero progetto è stato diviso in tre grandi componenti, ciascuno elaborato in una classe e con uno specifico scopo:

- MainWindow per funzionare sia da "pagina principale" che da nucleo stesso dell'applicazione, gestendo gli altri elementi. Infatti, gli altri oggetti vengono creati al suo interno e tutte le interazioni vengono elaborate dalle sue funzioni.
- SetupDialog per raccogliere tutti i dati necessari e le varie specifiche d'ambiente, inserite dall'utente, per l'operazione da eseguire. Alla pressione del suo tasto conferma, vengono sia trasmessi dei dati alla classe principale, che generate le varie sottocartelle di progetto con i relativi file di configurazione.
- CameraView per riprodurre in tempo reale gli output delle telecamere, in modo da individuare prontamente errori di posizionamento ed anomalie. Fornita di due modalità, permette di visualizzare globalmente o singolarmente i vari output video.

4.1.1 Funzionalità e modalità d'uso

Il software viene lanciato come un normale nodo ROS, da linea di comando (si presuppongono un nodo master attivo in un altro terminale (roscore), e il giusto posizionamento della workspace (source …/devel/setup.bash)), componendo il seguente comando "rosrun first_gui_pkg start_gui". Durante tutta l'esecuzione verranno inviati al terminale dei messaggi ROS_INFO che notificano il compimento delle varie azioni intraprese attraverso la GUI (vedi figura 4.3).

```
[ INFO] [1726584363.713935584]: "Prova" session created successfully
[ INFO] [1726584366.378253371]: Setup dialog opened
[ INFO] [1726584368.902621989]: Setup saved
```

Figura 4.3: Messaggi ROS_INFO inviati sul terminale da GIHMs

All'avvio dell'applicazione vengono presentati un menu e alcuni pulsanti:

- L'utente ha la possibilità di scegliere di creare una nuova sessione (unica "tipologia" di progetto), riaprire l'ultima modificata, se presente, o di aprirne una vecchia (vedi figura 4.4).

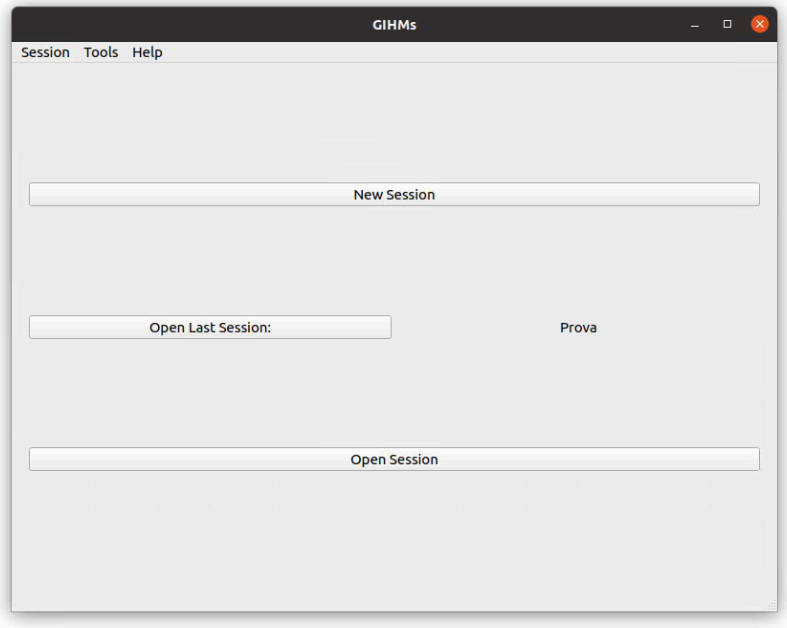


Figura 4.4: Finestra iniziale all'apertura di GIHMs

Una volta scelta la sessione di lavoro la pagina si aggiorna mostrando un pannello di controllo:

- In un primo momento, le diverse funzionalità non risultano attive e disponibili, e rimarranno tali finché non verrà completato un semplice setup aperto in una nuova finestra di dialogo dall'unico pulsante accessibile ("Open Setup" vedi figura 4.5).

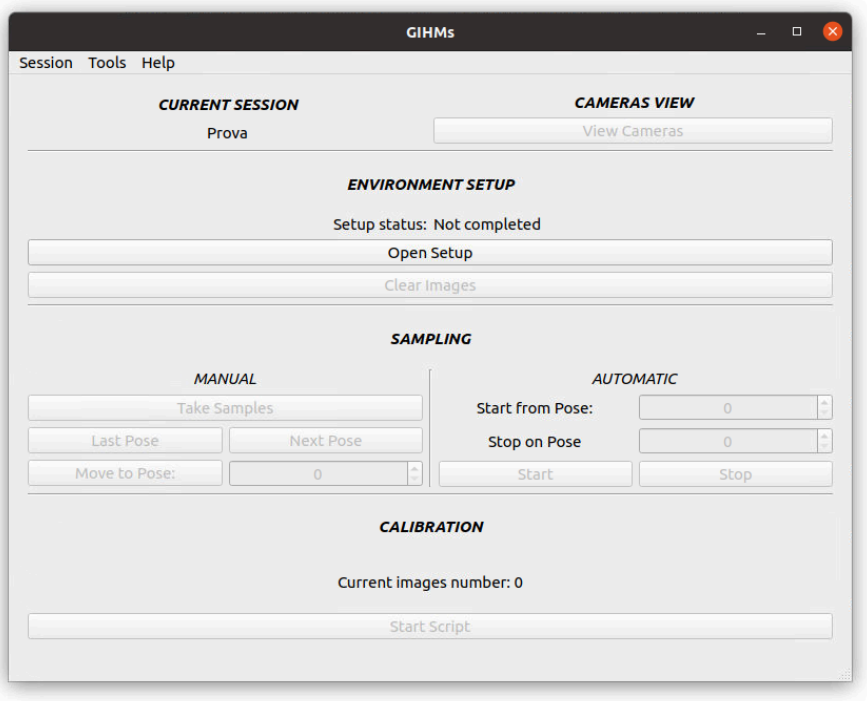


Figura 4.5: Finestra principale di GIHMs alla creazione di un nuovo progetto (sessione). Quando il setup non è ancora stato completato le altre opzioni risultano disabilitate

Per procedere con l'utilizzo del software è necessario, infatti, fornire alcune indicazioni riguardo all'ambiente in cui lavorare. Inoltre, c'è la possibilità di selezionare una traiettoria (quella corrente verrà sovrascritta) per il robot da copiare, già precedentemente costruita. Alcuni template vengono forniti con il download del software, come la semisfera ("Semisphere" vedi figura 4.6).

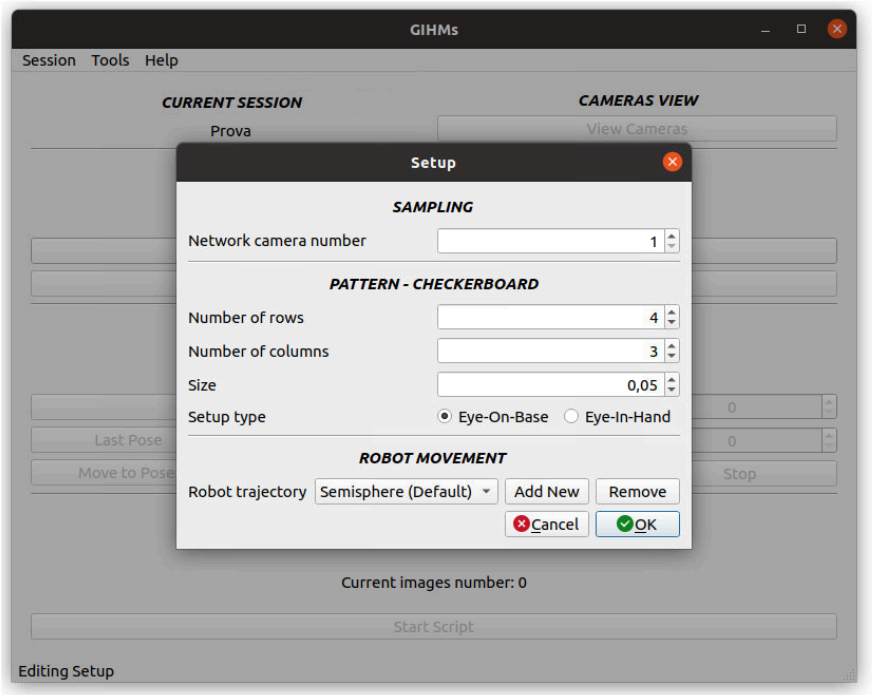


Figura 4.6: Finestra di dialogo del setup di GIHMs con le varie opzioni da compilare

Eseguita l'operazione di setup, tra le varie funzionalità che diventano disponibili troviamo:

- La possibilità di avviare una procedura di calibrazione in maniera automatica, se già disponibile una traiettoria per il movimento del robot (lato destro di "Sampling", vedi figura 4.7).
- Una serie di comandi che permettono di effettuare l'acquisizione dei dati manualmente passo dopo passo (step-by-step) (lato sinistro di "Sampling", vedi figura 4.7).

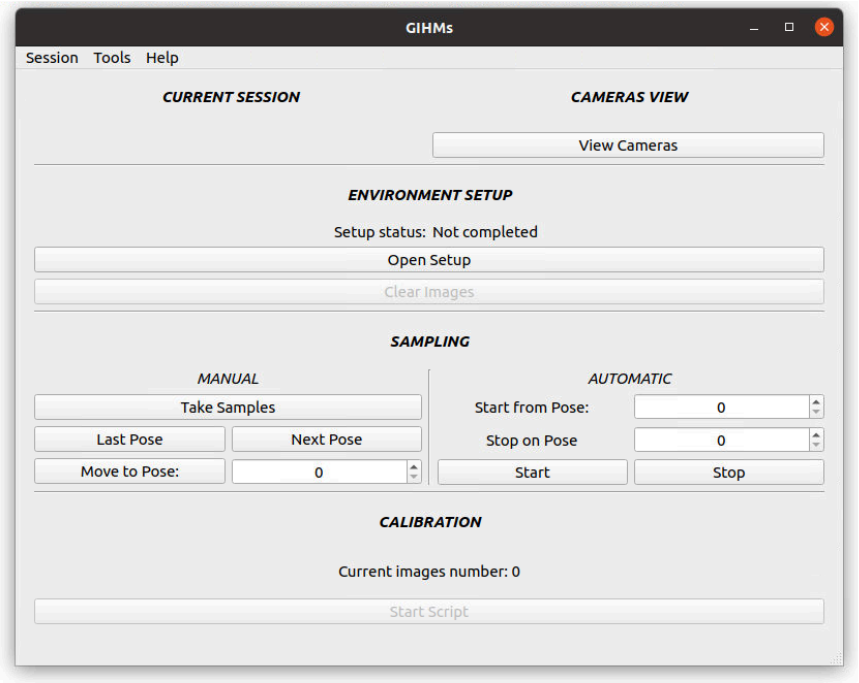
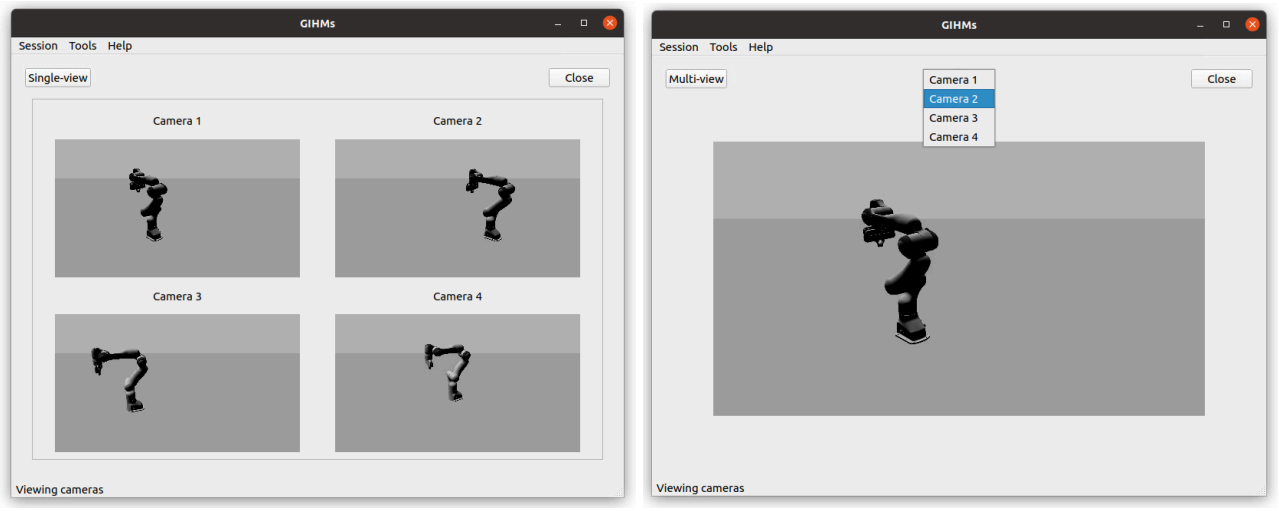


Figura 4.7: Finestra principale di GIHMs dopo che è stato completato il setup, varie opzioni risultano ora abilitate

- La possibilità di lanciare singolarmente lo script di calibrazione, un volta presenti i campioni necessari (immagini e pose), con il pulsante "Start Script" (vedi figura 4.7).
- È poi possibile visualizzare l'output delle telecamere connesse premendo il tasto "Cameras View" (vedi figura 4.7). In questo modo verrà cambiato l'aspetto della GUI, non mostrando più i pulsanti di prima (di fatto cambia la pagina di una raccolta di widget). In questa nuova configurazione, si potrà scegliere di visualizzare una grafica unita con tutte le telecamere (vedi figura 4.8 (a)), oppure di visionarne, a scelta, una alla volta (vedi figura 4.8 (b)).



(a) (b)

Figura 4.8: Visione dell'output delle telecamere del simulatore Gazebo da GIHMs, in modalità multipla (a) e singola (b)

4.1.2 Librerie e frameworks

Prima di affrontare la progettazione del software, è stato necessario studiare l'ambiente di lavoro utilizzato nel laboratorio, ovvero, ROS. Dopo aver esplorato le funzionalità e le caratteristiche di ROS, è stata effettuata anche una piccola ricerca per scegliere un framework in grado di aiutarmi nella realizzazione. I risultati hanno portato a scegliere Qt ("cute") sia per le sue caratteristiche tecniche e la sua compatibilità con ROS, sia per il vastissimo utilizzo che ne viene fatto in programmi moderni, potendo così conoscere uno strumento che facilmente si potrà incontrare di nuovo in futuro.

ROS

ROS (Robot Operating System) è un insieme di librerie e strumenti software che aiutano a creare applicazioni robotiche. ROS è una piattaforma open-source ampiamente utilizzata nella robotica che fornisce i servizi standard di un sistema operativo come l'astrazione dell'hardware, il controllo dei dispositivi tramite driver, la comunicazione tra processi e la gestione delle applicazioni (package). Consente anche l'integrazione di pacchetti dedicati, come quelli per la calibrazione di tecnologie LiDAR o di altri sensori, oltre a supportare algoritmi avanzati per la correzione di errori di posizionamento e allineamento. Grazie alla sua architettura modulare, ROS consente di adattare facilmente le procedure di calibrazione a diverse applicazioni robotiche, sia industriali che accademiche.

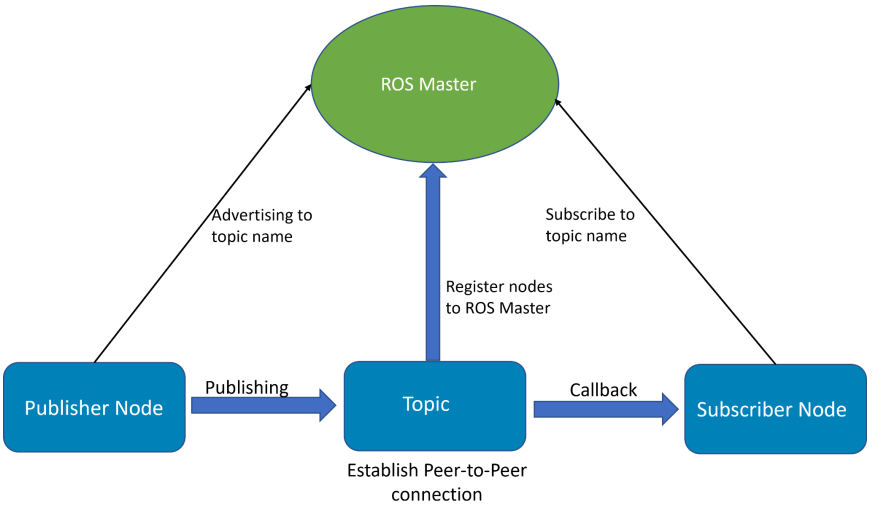


Figura 4.9: L'immagine rappresenta uno schema del funzionamento di base di ROS con il modello publisher-subscriber e l'uso dei topic. In basso al centro dello schema c'è un topic, che funge da canale di comunicazione per lo scambio di dati. A sinistra si trova un publisher, che invia messaggi al topic, mentre a destra è presente un subscriber, che riceve tali messaggi. Le frecce mostrano il flusso di informazioni attraverso il topic, illustrando la comunicazione asincrona e continua tipica di ROS. Tutti gli elementi vengono monitorati e fanno riferimento ad un nodo ROS Master (in alto). Lo schema è semplificato, ma efficace nel visualizzare il flusso di dati all'interno di un sistema ROS [S13]

Qt

Il framework Qt (“cute”) è un framework di sviluppo software multipiattaforma (desktop, embedded e mobile) utilizzato per la creazione di interfacce utente e applicazioni.

Rilasciato per la prima volta nel 1995, Qt si è evoluto nel tempo e, ad oggi, offre diversi tipi di approcci per la progettazione di interfacce utente, sia più tradizionali come Qt Widgets, basato su widget[3], sia come Qt Quick, più moderno e dichiarativo che utilizza il linguaggio QML. Il framework, scritto in C++, è in grado di offrire prestazioni elevate e di integrarsi alle librerie già esistenti, facilitando la realizzazione di interfacce grafiche complete e accattivanti. Più recentemente sono state introdotte interfacce e plug-in, come PyQt per il python, che consentono di interagire con il toolkit anche negli altri linguaggi di programmazione. Qt forte della sua storia quasi trentennale, si mostra come un framework affidabile, efficiente e ben supportato.

[3] Crasi di window e gadget, indica il componente grafico di base dell’interfaccia utente di un programma, che facilita l’interazione con il programma stesso [S11].

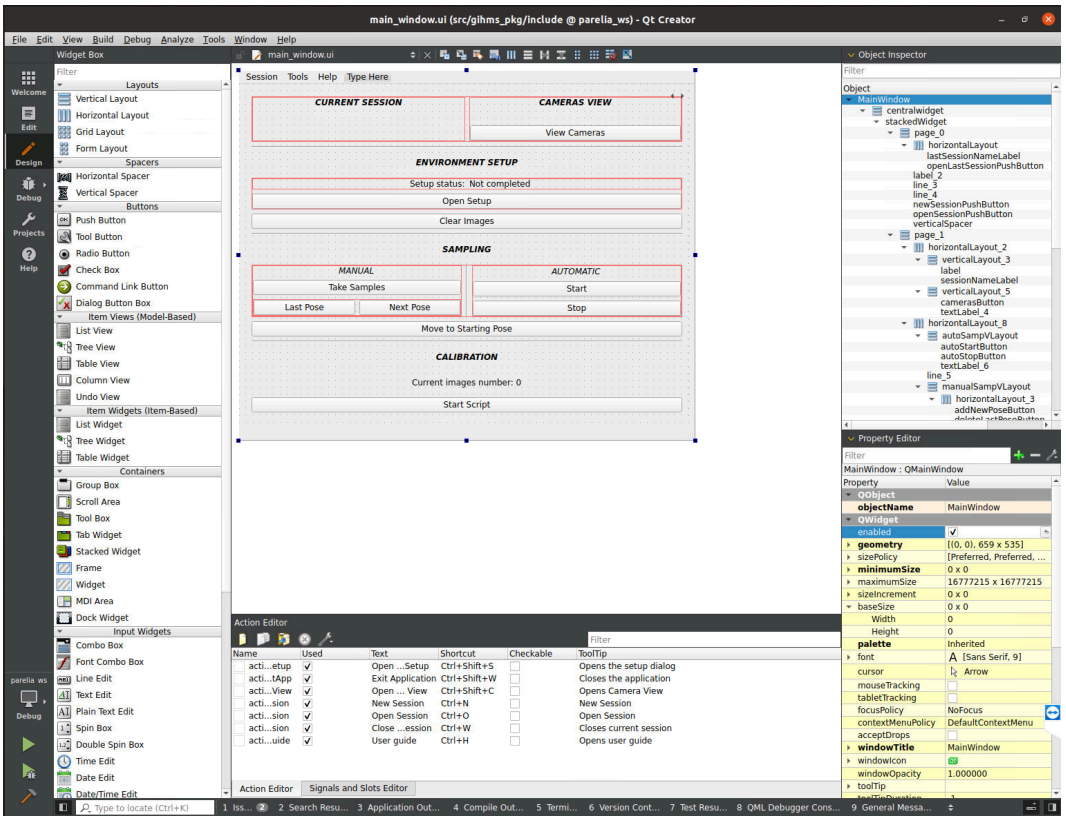


Figura 4.10: L'immagine mostra la finestra di Qt Creator, un ambiente di sviluppo integrato (IDE) per la creazione di applicazioni Qt. Ci troviamo nella sezione di design dove al centro della schermata vediamo una preview della finestra durante la sua progettazione grafica.

Per convertire le immagini trasmesse dalle telecamere come messaggio ROS, è stato utilizzata la libreria OpenCV, anche se non a pieno delle sue potenzialità.

OpenCV

OpenCV (Open Source Computer Vision Library) è una libreria open source ampiamente utilizzata per applicazioni di visione artificiale e elaborazione di immagini, dalla robotica alla realtà aumentata, sviluppata in C++. Offre una vasta gamma di funzionalità per il riconoscimento di oggetti, rilevamento di volti, elaborazione video, analisi di immagini in tempo reale e molto altro [S9].

Durante la progettazione del software, per manipolare file YAML (Ain't Markup Language), utilizzati sia come file di configurazione sia come file di posa del robot, è stata utilizzata la libreria yaml-cpp. Mentre per l'interazione con il robot è stata realizzata attraverso il plug-in di motion planning MoveIt.

yaml-cpp

Yaml-cpp è una libreria C++ utilizzata per il parsing e la generazione di file YAML, un formato di

serializzazione dati leggibile dall'uomo, spesso utilizzato nell'ambito della robotica per configurare parametri, descrivere modelli di robot e definire scenari complessi [S4].

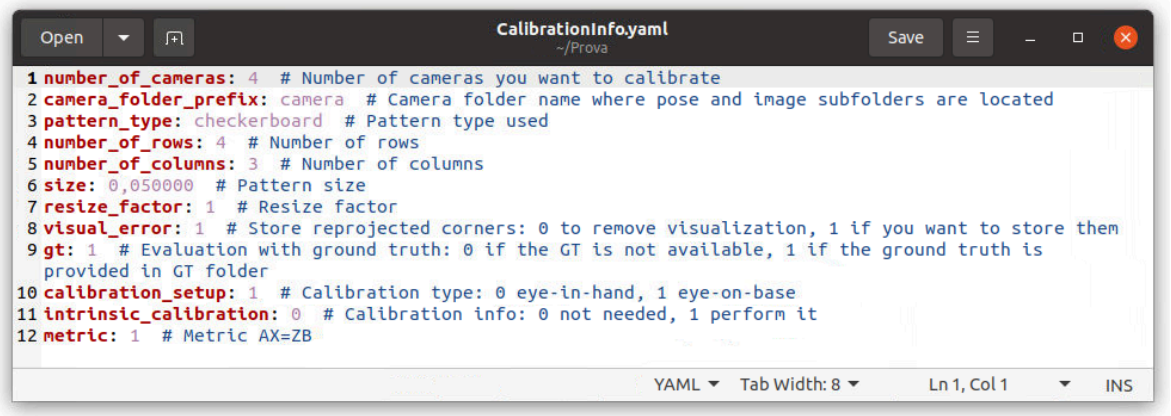


Figura 4.11: L'immagine mostra il file CalibrationInfo, un file YAML, generato da GIHMs al completamento della fase di setup

MoveIt

Una delle principali piattaforme per la pianificazione del movimento dei robot in ROS, fornisce strumenti avanzati per la pianificazione cinematica e dinamica, la manipolazione di bracci robotici e la gestione di ambienti complessi. Sfrutta i file YAML per configurare numerosi aspetti come i limiti di giunto, le impostazioni del planning del movimento e i modelli cinematici [B10].

Dato il suo utilizzo nei software di calibrazione trattati nella sezione Software di calibrazione 3.2 (pagina 14), e per una possibile integrazione in GIHMs in futuro, parliamo brevemente di RViz.

RViz

RViz è uno strumento di visualizzazione 3D utilizzato nella robotica per monitorare in tempo reale il comportamento dei robot e l'ambiente circostante. Integrato con ROS, permette di visualizzare vari tipi di dati sensoriali come quelli provenienti da tecnologie LiDAR (Light Detection And Ranging), telecamere o scanner laser, assieme a modelli 3D dettagliati dei robot, consentendo di analizzare il movimento e lo stato delle giunture. Oltre a supportare simulazioni e a interagire con simulatori come Gazebo (utilizzato anche in questo progetto), offre strumenti per modificare pose e interagire con l'ambiente in modo intuitivo [S5].

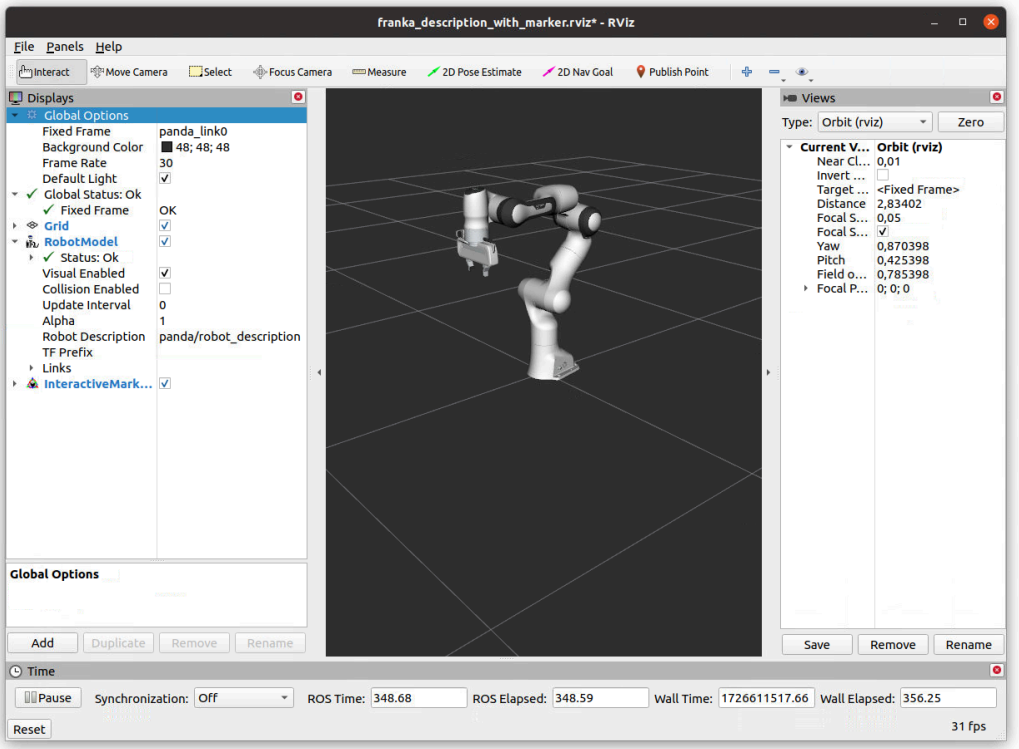


Figura 4.12: Finestra di RViz in cui viene riprodotto un modello 3d del Braccio robotico Franka Research 3 (lo stesso di figura 1.1)

5. Discussione

Un esempio di ambiente in cui verrà impiegata l'interfaccia, ovvero il laboratorio dell'università, si è rivelato un elemento chiave nella determinazione dei compiti (e conseguente carico lavorativo) del progetto. Trattandosi di un setup in cui tutte le telecamere ed il robot sono già configurati e connessi ad un computer master, ha permesso di escludere dal software tutta la parte relativa alla creazione dell'ambiente di lavoro, che sarebbe risultata dispendiosa da implementare.

Gli obiettivi più importanti da raggiungere con il nuovo software erano l'ottimizzazione, in termini di semplicità e velocità, del processo di calibrazione, visto che quello attuale risultava troppo lungo ed elaborato, e l'interazione generale di un sistema multi telecamera, non attualmente esistente (almeno, nei termini richiesti).

Dopo una prima analisi delle richieste, è iniziata la fase di definizione degli oggetti, delle azioni e delle loro rappresentazioni sullo schermo per l'interfaccia. Questa, ha comportato la creazione di scenari utente che andassero a simulare specifiche condizioni di utilizzo e il compimento di obiettivi mirati (come la necessità di riaprire un vecchio file per apportare una modifica). Individuare quali oggetti e azioni dovessero essere disponibili, e definire esattamente lo scopo e la finalità delle varie componenti, è stato l'aspetto chiave. In questo caso, capire come strutturare la cartella di progetto come richiesto per lo script di calibrazione è stato fondamentale per impostare correttamente tutta la parte di gestione e salvataggio dei dati.

Un concetto importante applicato durante la realizzazione dell'interfaccia utente è stato la separazione di quest'ultima dalle funzionalità intrinseche dell'applicazione. Dividere il lato grafico e organizzativo, realizzato tramite i file User Interface[4] di Qt, da quello tecnico, costituito dal codice vero e proprio, è un principio fondamentale nella progettazione di sistemi interattivi efficaci ed efficienti. In questo modo, modifiche alla UI non richiedono modifiche al codice delle funzionalità principali e viceversa.

La composizione del layout di una finestra in un'interfaccia grafica ha presentato diverse difficoltà, come l'adattamento alle diverse risoluzioni di schermo e la gestione del ridimensionamento dinamico degli elementi. La soluzione è stata raggiunta con l'utilizzo, abbastanza macchinoso, dei layout di Qt. Inoltre, garantire una disposizione coerente e usabile dei componenti, mantenendo sia l'estetica che la funzionalità, è stato complesso. Tra le problematicità incontrate, le quali hanno

[4] File con estensione .ui, sono file XML generati da Qt per la progettazione grafica delle interfacce utente. Questi file descrivono il layout e i componenti visivi di una finestra, come pulsanti, caselle di testo e menu. Durante la compilazione, vengono convertiti in codice C++, permettendone l'integrazione con la logica dell'applicazione.

richiesto diverse modifiche e più fasi di elaborazione, durante il mantenimento di una struttura semplice ed ordinata, sono stati l'allineamento dei componenti, non sempre convincente, e la difficoltà nel rendere l'interfaccia realmente intuitiva per un nuovo utente.

Il focus durante la realizzazione è stato posto sul cercare di comprimere in meno click possibili l'intero processo di calibrazione, in modo da massimizzare l'efficienza di utilizzo, ma contemporaneamente, di riuscire a renderlo flessibile e adattabile in futuro a nuovi ambienti (diversi dal laboratorio). Per questo, è stata inserita la funzionalità di avvio automatico dell'operazione di raccolta dati, infatti, una volta completato il setup, potrà essere avviata e procederà a raccogliere il numero di campioni indicato senza bisogno di avere ulteriori indicazioni. Durante questa operazione, saranno bloccate le altre funzionalità del software, tranne la visualizzazione delle telecamere, sarà possibile, infatti, osservarle comodamente senza interferire nel processo in background.

L'elaborazione della meccanica di acquisizione delle immagini non si è rivelata troppo complessa, anche perchè in parte documentata nella wiki di ROS, infatti, la comunicazione con le telecamere è stata gestita tramite strumenti ROS e con l'aiuto, per la conversione tra formati, di OpenCV. All'interno di GIHMS non è possibile modificare i topic di lettura delle immagini, le informazioni vengono lette nei topic del tipo "/opt_00/kinect2/rgb/image_raw", i quali vengono adattati dinamicamente in base al numero assegnato alla telecamera. In questo modo il formato è impostato ad RGB8[5], tuttavia, la funzione di conversione "MatToQImage" di "CamerasView" può permettere, in caso di utilizzo futuro, e con piccole modifiche, di elaborare anche di immagini di altra tipologia. La modifica del topic di lettura consentirebbe ad un software di interagire con diverse tipologie di sensori e di adattarsi ad ambienti diversi.

Dalla creazione del primo prototipo dell'interfaccia, si sono susseguiti diversi cicli di progettazione, test e perfezionamento, in cui, man mano che il software progrediva, nuove funzionalità si sono rivelate interessanti, alcune sono anche state integrate, ma altre sono state lasciate come idee future.

Non potendo sempre accedere al laboratorio, i test necessari sono stati realizzati con l'aiuto del simulatore Gazebo e di RViz, i quali hanno permesso di ricreare l'ambiente del laboratorio in formato virtuale. Lavorare con questi strumenti si è rivelato comunque utile al completamento del lavoro, tuttavia,

[5] Il numero "8" rappresenta il numero di bit per canale del formato.

6. Conclusioni

Dopo aver analizzato la struttura, gli elementi e gli aspetti importanti delle interfacce grafiche; aver analizzato i metodi di calibrazione Hand-Eye ed alcuni tra i software di calibrazione esistenti con le loro problematicità; è iniziata la realizzazione di un'interfaccia grafica in grado di sopperire alle lacune evidenziate, “GIHMs”.

A lavoro ultimato, il software GIHMs, anche se non completo tutte le funzionalità concepite, è in grado di svolgere la funzione principale per cui è stato richiesto. Riesce, infatti, a comunicare simultaneamente con il robot e le telecamere, acquisendo tutti i dati e le immagini necessarie per effettuare la calibrazione Hand-Eye. Tuttavia, varie difficoltà occorse durante lo sviluppo, come la ricerca di librerie e l’impossibilità di testare correttamente il software, hanno rallentato il lavoro e non hanno permesso di confezionare un prodotto visivamente accattivante come progettato inizialmente.

Attualmente GIHMs presenta anche alcune limitazioni, tra le quali, non poter utilizzare il software senza selezionare una “robot trajectory” e la mancanza di una sezione dedicata alla configurazione dei topic delle telecamere. Infatti, sebbene si fosse ipotizzato, in fase di progettazione, di inserire la funzionalità di muovere il robot verso una posa generata casualmente per creare nuove traiettorie uniche, ciò non si è rivelato possibile a causa del costo progettuale che avrebbe richiesto. Inoltre, non è possibile, per l’utente, visualizzare e modificare il topic ROS di comunicazione con le varie telecamere che attualmente è impostato per l’utilizzo nel laboratorio dell’università. Queste mancanze vengono lasciate come proposte di implementazioni future.

Sono state ipotizzate anche altre funzionalità che potrebbero essere aggiunte al software, una particolarmente interessante, è l’integrazione con RViz per riuscire a creare e modellare un modello 3D digitale dell’ambiente di lavoro. Un’altro ottimo spunto, sarebbe quello di consentire l’utilizzo di reti non omogenee, basterebbe, infatti, realizzare un sistema di inserimento delle tipologie di telecamera (o sensore) e del relativo numero di elementi in utilizzo.

In conclusione, GIHMs è un’interfaccia grafica semplice per effettuare la calibrazione di un sistema multi telecamera. Sebbene al momento offra un numero limitato di funzionalità, mostra un grande potenziale per sviluppi futuri.

Il software è disponibile per il download al link github.com/Parelia/GIHMs.git come pacchetto ROS.

Bibliografia

[B1] Oliveira, M., Pedrosa, E., de Aguiar, A. P., Rato, D. F. P. D., dos Santos, F. N., Dias, P., & Santos, V. (2022). ATOM: A general calibration framework for multi-modal, multi-sensor systems. *Expert Systems with Applications*, *207*, 118000.

[B2] Gomes, M., Oliveira, M., & Santos, V. (2023). ATOM Calibration Framework: Interaction and Visualization Functionalities. *Sensors*, *23*(2), 936.

[B3] Enebuse, I., Foo, M., Ibrahim, B. S. K. K., Ahmed, H., Supmak, F., & Eyobu, O. S. (2021). A comparative review of hand-eye calibration techniques for vision guided robots. *IEEE Access*, *9*, 113143-113155.

[B4] Jiang, J., Luo, X., Luo, Q., Qiao, L., & Li, M. (2022). An overview of hand-eye calibration. *The International Journal of Advanced Manufacturing Technology*, *119*(1), 77-97.

[B5] Evangelista, D., Allegro, D., Terreran, M., Pretto, A., & Ghidoni, S. (2022, September). An unified iterative hand-eye calibration method for eye-on-base and eye-in-hand setups. In *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)* (pp. 1-7). IEEE.

[B6] Tidwell, J. (2010). *Designing interfaces: Patterns for effective interaction design*. " O'Reilly Media, Inc.".

[B7] Krug, S. (2000). *Don't make me think!: a common sense approach to Web usability*. Pearson Education India.

[B8] Preece, J., Rogers, Y., & Sharp, H. (2002). Interaction Design: beyond human. *United Kingdom: Computer Interaction, John Willey Sons*.

[B9] Allegro, D., Terreran, M., & Ghidoni, S. (2024). Multi-Camera Hand-Eye Calibration for Human-Robot Collaboration in Industrial Robotic Workcells. *arXiv preprint arXiv:2406.11392*.

[B10] David Coleman, Ioan A. Șucan, Sachin Chitta, Nikolaus Correll, Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study, *Journal of Software Engineering for Robotics*, 5(1):3–16, May 2014. doi: 10.6092/JOSER_2014_05_01_p3

Sitografia

[S1] consystem.it [Consultato il 18 09 2014]

[S2] generationrobots.com [Consultato il 18 09 2014]

[S3] github.com/IFL-CAMP/easy_handeye [Consultato il 18 09 2014]

[S4] github.com/jbeder/yaml-cpp [Consultato il 18 09 2014]

[S5] github.com/ros-visualization/rviz [Consultato il 18 09 2014]

[S6] gutflg.com [Consultato il 18 09 2014]

[S7] idem-tech.it [Consultato il 18 09 2014]

[S8] medium.com [Consultato il 18 09 2014]

[S9] opencv.org [Consultato il 18 09 2014]

[S10] storiainformatica.it [Consultato il 18 09 2014]

[S11] treccani.it [Consultato il 18 09 2014]

[S12] it.vecteezy.com [Consultato il 18 09 2014]

[S13] docs.wsr.studica.com [Consultato il 18 09 2014]