



UNIVERSITÀ
POLITECNICA
DELLE MARCHE

FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA TRIENNALE IN INGEGNERIA MECCANICA

Implementazione in ROS2 di algoritmi di task allocation per la collaborazione uomo-robot

**ROS2 implementation of task allocation algorithms for human-robot
collaboration**

Candidato:
Lorenzo Martella

Relatore:
Prof. Giacomo Palmieri

Correlatore:
Dott. Federico Neri
Dott. Cecilia Scoccia

Anno Accademico 2025-2026



UNIVERSITÀ
POLITECNICA
DELLE MARCHE

FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA TRIENNALE IN INGEGNERIA MECCANICA

Implementazione in ROS2 di algoritmi di task allocation per la collaborazione uomo-robot

**ROS2 implementation of task allocation algorithms for human-robot
collaboration**

Candidato:
Lorenzo Martella

Relatore:
Prof. Giacomo Palmieri

Correlatore:
Dott. Federico Neri
Dott. Cecilia Scoccia

Anno Accademico 2025-2026

UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA TRIENNALE IN INGEGNERIA MECCANICA
Via Brecce Bianche – 60131 Ancona (AN), Italy

Sommario

Il presente lavoro di tesi affronta il problema della *task allocation* in un contesto di collaborazione uomo-robot (*Human-Robot Collaboration*, HRC), applicato a un'operazione di pick-and-place industriale. È stata sviluppata un'architettura integrata basata su ROS2 e MATLAB, nella quale un manipolatore collaborativo UR5e e un operatore umano lavorano in parallelo su un banco di lavoro condiviso, suddividendo dinamicamente i compiti tra le due risorse.

Il sistema impiega un algoritmo basato sul problema del commesso viaggiatore (*Traveling Salesman Problem*, TSP) per allocare i task in modo da minimizzare i tempi complessivi di esecuzione, aggiornando iterativamente le matrici dei costi sulla base di misurazioni sperimentali reali acquisite tramite microinterruttori di finecorsa e schede Arduino. La pianificazione delle traiettorie del robot è affidata al framework MoveIt2, mentre l'interfaccia luminosa a LED guida l'operatore nell'esecuzione dei task assegnati.

Sono state condotte 21 sessioni sperimentali, i cui risultati hanno mostrato una progressiva ottimizzazione della distribuzione dei task tra robot e operatore, con una significativa riduzione dei tempi attribuibili all'operatore umano grazie all'effetto di apprendimento e all'aggiornamento iterativo del modello di costo. L'analisi ha inoltre evidenziato l'influenza del layout fisico del banco di lavoro sulle preferenze dell'algoritmo di allocazione.

Il lavoro si conclude con una discussione dei limiti riscontrati e dei possibili sviluppi futuri, tra cui l'integrazione di sistemi di sicurezza dinamica conformi alla norma ISO/TS 15066 e l'implementazione di una misurazione rigorosa del *makespan* effettivo della cella.

Abstract

This thesis addresses the problem of task allocation in a human-robot collaboration (HRC) context, applied to an industrial pick-and-place operation. An integrated architecture based on ROS2 and MATLAB was developed, in which a UR5e collaborative manipulator and a human operator work in parallel on a shared workbench, dynamically distributing tasks between the two resources.

The system employs an algorithm based on the Traveling Salesman Problem (TSP) to allocate tasks in a way that minimizes overall execution time, iteratively updating the cost matrices based on real experimental measurements acquired through limit switches and Arduino boards. The robot's trajectory planning is handled by the MoveIt2 framework, while an LED-based visual interface guides the operator through the assigned tasks.

A total of 21 experimental sessions were conducted, the results of which showed a progressive optimization of the task distribution between robot and operator, with a significant reduction in the time required by the human operator due to the learning effect and the iterative update of the cost model. The analysis also highlighted the influence of the physical layout of the workbench on the preferences of the allocation algorithm.

The work concludes with a discussion of the limitations encountered and possible future developments, including the integration of dynamic safety systems compliant with the ISO/TS 15066 standard and the implementation of a rigorous measurement of the cell's effective makespan.

Indice

1	Introduzione	1
1.1	Contesto generale (industria 4.0, collaborazione uomo-robot)	1
1.2	Perché nasce la necessità della task allocation	1
1.3	Obiettivo della tesi	2
1.4	Struttura del documento	2
2	Stato dell'arte	3
2.1	Robotica collaborativa (Cobot)	3
2.2	Task allocation in sistemi uomo-robot	4
2.3	Pick and place industriale	6
3	Materiale utilizzato	9
3.1	Robot UR5e	9
3.1.1	Control Box	10
3.1.2	Teach Pendant e PolyScope	11
3.2	Pinza Robotiq Hand-E	12
3.3	Configurazione della stazione robotica	13
3.3.1	Layout delle stazioni di pick-and-place	14
3.3.2	Sistema LED e finecorsa (Arduino)	16
4	Struttura software – ROS2	19
4.1	Ambiente ROS2	19
4.1.1	Architettura ROS2	19
4.1.2	Workspace e pacchetti	20
4.2	Driver UR5e	20
4.2.1	Infrastruttura di rete e protocollo RTDE	20
4.2.2	Procedura di inizializzazione sequenziale	20
4.2.3	Regimi di controllo e modulazione della velocità	21
4.2.4	Architettura gerarchica e limiti del driver	21
4.3	MoveIt2	21
4.3.1	Cinematica diretta (FK)	22
4.3.2	Server Python	23
4.4	Nodo pick_place_completo (C++)	24
4.4.1	Subscriber e coda comandi	24
4.4.2	Sequenza pick and place	24
4.4.3	Registrazione coordinate stazioni	27

5	Struttura software - MATLAB	29
5.1	Introduzione a MATLAB	29
5.2	Toolbox installati	29
5.3	Interfacciamento hardware Arduino	30
5.3.1	Sistema di segnalazione luminosa (Arduino UNO R4 WiFi) .	31
5.3.2	Sistema di acquisizione eventi (Arduino Mega 2560)	32
5.4	Algoritmo di task allocation (TSP)	33
5.4.1	Matrici dei tempi T_{rob} e T_{op}	33
5.4.2	Funzione <code>function_allocation_T_P</code>	35
5.5	Sviluppo incrementale del sistema	36
5.5.1	Fase di validazione hardware e logica di base (Step 1–5) . . .	36
5.5.2	Acquisizione tempi operatore (Step 7)	37
5.5.3	Acquisizione tempi robot (Step 8)	38
5.5.4	Elaborazione dei dati: media matrici (Step 9)	39
5.5.5	Task allocation e ottimizzazione (Step 6)	40
5.6	Integrazione MATLAB–ROS2	41
5.6.1	Publisher sul topic <code>/pick_place_cmd</code>	42
5.6.2	Comunicazione con <code>pick_place_completo</code>	43
6	Pipeline del sistema completo e risultati	45
6.1	Architettura generale e sequenza di avvio	45
6.2	Protocollo operativo del supervisore	46
6.3	Risultati sperimentali	46
6.3.1	Stazioni di pick più utilizzate: Robot vs Operatore	47
6.3.2	Stazioni di place più utilizzate: Robot vs Operatore	48
6.3.3	Heatmap delle transizioni	49
6.3.4	Distribuzione dei task per sessione	50
6.3.5	Evoluzione del tempo cumulativo per sessione	51
6.3.6	Evoluzione della somma totale delle celle della matrice dei tempi	53
6.4	Matrici dei tempi aggiornate	54
7	Conclusioni e sviluppi futuri	57
7.1	Sintesi dei risultati raggiunti	57
7.2	Sviluppi futuri	57

Elenco delle figure

3.1	Robot collaborativo UR5e	9
3.2	Control Box del robot UR5e	10
3.3	Porte esterne della Control Box UR5e: (1) Teach Pendant port, (2) SD card slot, (3) Ethernet port, (4) USB ports, (5) Mini DisplayPort, (6) 10A mini blade fuse.	11
3.4	Pinza elettrica parallela Robotiq Hand-E.	12
3.5	Vista dall'alto della stazione robotica: robot UR5e al centro, stazioni pick a sinistra e stazioni place a destra con i LED di segnalazione colorati.	14
3.6	Vista dall'alto della stazione robotica: robot UR5e al centro, stazioni pick (1–5) a sinistra e stazioni place (6–10) a destra. I colori corrispondono ai LED fisici di segnalazione.	15
3.7	Alloggiamento stampato in 3D: vano per il cubetto (sinistra) e foro per il LED (destra).	15
3.8	Microinterruttore di finecorsa con linguetta metallica	16
4.1	Rappresentazione schematica del manipolatore UR5e a 6 giunti: sistema di riferimento fisso $\{b\}$ ancorato alla base e sistema di riferimento mobile $\{e\}$ solidale con l'end-effector (la flangia), collegati dalla matrice di trasformazione omogenea $T_e^b(q)$	22
4.2	Schema della traiettoria completa pick-and-place: trasferimento MoveJ tra i punti di offset delle due stazioni.	25
4.3	Sequenza di approccio alla stazione di pick: MoveJ verso il punto alto (+13 cm), MoveL di discesa e chiusura pinza.	26
4.4	Sequenza di rilascio alla stazione di place: MoveL di discesa, apertura pinza e MoveL di risalita.	26
5.1	Circuito interno del modulo di controllo: schede Arduino Mega 2560 e UNO R4 WiFi, alimentatore switching, busbar e cablaggio dei LED e finecorsa.	30
5.2	Modulo relè FC-31 con relè elettromeccanico Songle SRD-05VDC-SL-C utilizzato per il pilotaggio dei LED di segnalazione.	31
5.3	Struttura della matrice dei tempi 10×10 : Quadrante A e D (celle nulle, transizioni non operative), Quadrante B (tempi pick $\rightarrow$ place) e Quadrante C (tempi place $\rightarrow$ pick).	34
5.4	Operatore in fase di prelievo (pick) alla stazione 1.	37

Elenco delle figure

5.5	Operatore in fase di deposito (place) alla stazione 6.	37
5.6	Manipolatore UR5e in fase di prelievo (pick) alla stazione 1.	39
5.7	Manipolatore UR5e in fase di deposito (place) alla stazione 6.	39
5.8	Flusso di elaborazione dati tra gli Step 7, 8, 9 e 6: acquisizione, mediazione e aggiornamento iterativo delle matrici dei tempi.	40
5.9	Operatore e robot operano in parallelo sulle rispettive stazioni durante l'esecuzione dello Step 6.	41
5.10	Dettaglio della collaborazione uomo-robot: entrambe le risorse eseguono task simultaneamente sulle stazioni assegnate.	41
5.11	Architettura software ROS2: flusso dati tra MATLAB (client supervisor), il nodo <code>pick_place_completo</code> (T_4), MoveIt2 (T_2) e il driver UR5e (T_1).	42
6.1	Distribuzione delle assegnazioni per le stazioni di pick tra robot e operatore, aggregata sulle 21 sessioni sperimentali.	47
6.2	Distribuzione delle assegnazioni per le stazioni di place tra robot e operatore, aggregata sulle 21 sessioni sperimentali.	48
6.3	Heatmap della distribuzione delle coppie pick $\rightarrow$ place assegnate alle due risorse nel corso delle 21 sessioni sperimentali. Le tonalità arancioni indicano una maggiore assegnazione al robot (R), le tonalità blu una maggiore assegnazione all'operatore (O).	49
6.4	Distribuzione del numero di task rilevati per robot e operatore per ciascuna sessione sperimentale (Sessione 1 esclusa), basata sul conteggio delle celle variate nel Quadrante B della matrice dei tempi.	50
6.5	Evoluzione dei tempi cumulativi per sessione del robot (arancione) e dell'operatore (verde) nelle sessioni dello Step 6 (Sessione 1 esclusa).	51
6.6	Evoluzione della somma totale delle celle delle matrici T_{op} e T_{rob} per ciascuna delle 21 sessioni sperimentali: somma totale combinata (blu), somma del robot (arancione) e somma dell'operatore (verde).	53

Elenco delle tabelle

2.1	Confronto tra i principali produttori di cobot	3
3.1	Caratteristiche tecniche del robot UR5e	9
3.2	Corrispondenza tra stazioni e colori dei LED	16
5.1	Esempio di matrice T_{op} acquisita il 09/06/2026. I valori in verde corrispondono ai Quadranti B e C (tempi in secondi); i valori nulli in blu ai Quadranti A e D (transizioni non operative).	38
5.2	Esempio di matrice T_{rob} acquisita il 09/06/2026. I valori in verde corrispondono ai Quadranti B e C (tempi in secondi); i valori nulli in blu ai Quadranti A e D (transizioni non operative).	39
5.3	Aggiornamento iterativo della matrice T_{op} nello Step 6: confronto tra due iterazioni consecutive. Le celle evidenziate in giallo indicano i valori aggiornati con le misurazioni più recenti.	41
6.1	Numero di assegnazioni per ciascuna stazione di pick, robot vs operatore, aggregate sulle 21 sessioni sperimentali.	47
6.2	Numero di assegnazioni per ciascuna stazione di place, robot vs operatore, aggregate sulle 21 sessioni sperimentali.	48
6.3	Confronto tra T_{op} alla sessione 1 (08/06/2026) e alla sessione 21 (12/06/2026). Le celle in giallo indicano i valori aggiornati; le celle in verde i Quadranti B e C; le celle in blu i Quadranti A e D (zeri). . .	54
6.4	Confronto tra T_{rob} alla sessione 1 (08/06/2026) e alla sessione 21 (12/06/2026). Le celle in giallo indicano i valori aggiornati; le variazioni sono contenute nell'ordine dell'1–5%, coerentemente con il comportamento deterministico del manipolatore.	55

1 Introduzione

1.1 Contesto generale (industria 4.0, collaborazione uomo-robot)

Il paradigma Industria 4.0 nasce in Germania nel 2011 con l'obiettivo di ottimizzare i processi produttivi aziendali attraverso l'introduzione di sistemi robotizzati capaci di svolgere operazioni rischiose o ripetitive al posto dell'operatore umano. I principali vantaggi di tale approccio risiedono nell'incremento della produttività e nella riduzione dei margini di errore, grazie all'esecuzione di movimenti predefiniti tramite programmazione. Nel decennio precedente, le prime applicazioni robotiche erano limitate allo svolgimento di operazioni all'interno di celle protettive, nelle quali l'accesso da parte degli operatori era consentito esclusivamente a ciclo completato. Ciò era dovuto all'assenza di sistemi di rilevazione che impedivano qualsiasi forma di collaborazione tra robot e operatore. A partire dalla metà degli anni 2010 si iniziano a registrare le prime applicazioni di collaborazione uomo-robot, che hanno portato alla progressiva eliminazione delle celle protettive, consentendo a operatore e robot di condividere il medesimo spazio di lavoro. Da quel periodo si osserva una significativa crescita nell'adozione dei robot collaborativi, comunemente denominati cobot. Tali sistemi operano in parallelo all'operatore grazie a sensori di rilevazione della presenza umana, in risposta ai quali il robot riduce automaticamente la propria velocità grazie a sistemi di controllo della sicurezza integrati. Il mercato dei cobot è in rapida espansione, trainato dalla crescente domanda di automazione flessibile e scalabile, accessibile anche alle piccole e medie imprese (PMI). L'integrazione con tecnologie emergenti come l'intelligenza artificiale e il machine learning amplia ulteriormente le capacità operative di questi sistemi. I dati confermano questa tendenza: il mercato ha registrato un fatturato di 121,7 milioni di dollari nel 2024, con proiezioni di crescita costante fino al 2034, trainate principalmente dai cobot a basso payload impiegati in operazioni di assemblaggio e pick and place [1].

1.2 Perché nasce la necessità della task allocation

La necessità della task allocation nasce dall'esigenza di ottimizzare l'efficienza dei processi produttivi e ridurre i costi operativi. Con l'incremento dell'automazione, non si mira a sostituire integralmente la manodopera umana, bensì a distribuire i compiti in modo ottimale tra operatore e robot, sfruttando le capacità di ciascuno. Il robot eccelle in operazioni ripetitive, precise e fisicamente gravose, mentre l'operatore

1 Introduzione

umano è insostituibile nella supervisione, nel controllo qualità e nella gestione di situazioni impreviste. Una corretta allocazione dei task consente inoltre di ridurre il carico ergonomico e cognitivo sull'operatore, migliorando le condizioni di lavoro e minimizzando i tempi complessivi di esecuzione.

1.3 Obiettivo della tesi

Nel presente elaborato è stato condotto uno studio sull'allocazione ottimale dei task in un contesto di collaborazione uomo-robot. L'applicazione scelta è un'operazione di pick and place, nella quale il robot e l'operatore lavorano in parallelo con il compito di spostare dei cubetti da una stazione di partenza a una stazione di arrivo. L'allocazione dei task è stata ottimizzata attraverso un algoritmo basato sul problema del commesso viaggiatore (TSP — Travelling Salesman Problem), con l'obiettivo di minimizzare i tempi complessivi di esecuzione.

1.4 Struttura del documento

Il presente elaborato è strutturato come segue. Il Capitolo 1 introduce il contesto industriale, le motivazioni e gli obiettivi del lavoro. Il Capitolo 2 presenta lo stato dell'arte relativo alla robotica collaborativa, agli algoritmi di task allocation e all'utilizzo di ROS2 nella ricerca attuale. Il Capitolo 3 descrive il materiale hardware utilizzato, tra cui il robot UR5e, la pinza Robotiq Hand-E e il sistema Arduino. Il Capitolo 4 illustra l'architettura software sviluppata in ROS2, comprendente i driver, MoveIt2 e il nodo di pick and place. Il Capitolo 5 descrive l'implementazione in MATLAB dell'algoritmo di task allocation e l'integrazione con ROS2. Il Capitolo 6 presenta la pipeline del sistema completo e i risultati sperimentali ottenuti. Infine, le Conclusioni riassumono i risultati raggiunti e propongono possibili sviluppi futuri.

2 Stato dell'arte

2.1 Robotica collaborativa (Cobot)

Con la loro crescita esponenziale, i cobot hanno portato una altrettanta crescita delle aziende che li producono. Tra le più affermate troviamo Universal Robots, KUKA, ABB e Fanuc. Esse si specializzano tutte nello stesso settore ma con approcci molto differenti nella struttura ingegneristica del robot, preferendo il marchio in base al tipo di applicazione di cui si necessita. Infatti, per ogni marchio esiste un modello più utilizzato a cui viene solitamente associata una specifica applicazione tipica, come mostrato in Tabella 2.1.

Tabella 2.1: Confronto tra i principali produttori di cobot

Produttore	Modello	Payload	Reach	Applicazione tipica
Universal Robots	UR5e	5 kg	850 mm	Pick and place, assemblaggio
KUKA	LBR iiwy	11 kg	1300 mm	Assemblaggio, controllo qualità
ABB	YuMi	0,5 kg	559 mm	Elettronica, piccoli assemblaggi
Fanuc	CR-35iA	35 kg	1813 mm	Movimentazione carichi pesanti

I cobot si possono classificare in base alla loro architettura in tre categorie: fissi, mobili e ibridi. I robot fissi operano su banchi o piedistalli, assistendo gli operatori nel loro raggio d'azione limitato. I robot mobili, invece, si spostano autonomamente nell'ambiente di lavoro grazie a sistemi AGV (Automated Guided Vehicle) o AMR (Autonomous Mobile Robot), utilizzati principalmente per il trasporto di materiali. I cobot ibridi combinano entrambe le caratteristiche: un braccio collaborativo montato su una piattaforma mobile, che consente di assistere l'operatore in diverse postazioni. Tuttavia, essendo montati su piattaforme mobili, presentano payload tipicamente inferiori rispetto ai cobot fissi. Le modalità di interazione uomo-robot sono ulteriormente suddivise in coesistenza, sincronizzazione, cooperazione e collaborazione, a seconda del grado di condivisione dello spazio e dei compiti. L'analisi del mercato evidenzia come la configurazione antropomorfa rappresenti oltre il 90% dei modelli disponibili, con payload compresi tra 5 e 20 kg e un'accuratezza tipicamente inferiore a 0,20 mm [2].

Per la programmazione di questi sistemi collaborativi è necessario tenere conto delle normative ISO (International Organization for Standardization). La norma di riferimento per i cobot è la **ISO/TS 15066:2016** [3], che definisce i requisiti di sicurezza per i sistemi robotici collaborativi, integrando le norme ISO 10218-1 e ISO 10218-2. L'obiettivo è combinare le prestazioni ripetitive del robot con le capacità individuali dell'operatore. La norma definisce quattro modalità di operazione collaborativa:

- **Safety-rated monitored stop** — il robot si ferma quando l'operatore entra nello spazio collaborativo;
- **Hand guiding** — l'operatore guida fisicamente il robot tramite un dispositivo apposito;
- **Speed and separation monitoring** — il robot riduce la velocità in base alla distanza dall'operatore, mantenendo una distanza minima di protezione calcolata tramite una formula specifica;
- **Power and force limiting** — limiti di forza e pressione sul corpo umano per evitare lesioni in caso di contatto accidentale, ad esempio 140 N sull'addome e 110 N sul viso.

2.2 Task allocation in sistemi uomo-robot

La task allocation in sistemi di collaborazione uomo-robot rappresenta uno dei temi più studiati nell'ambito dell'Industria 4.0, con l'obiettivo di ottimizzare i tempi di esecuzione migliorando al contempo il benessere ergonomico e la produttività dell'operatore. Si tratta di un processo strategico che divide il lavoro in modo ottimale ed efficiente tra operatori umani e robot collaborativi, creando una sinergia positiva tra le capacità cognitive, l'adattabilità e l'esperienza dell'uomo unite alla precisione, velocità e resistenza del cobot.

Baratta et al. [4] analizzano questo problema in un contesto di assemblaggio industriale, osservando come l'affaticamento dell'operatore rappresenti un fattore critico: maggiore è l'impegno fisico richiesto da un task, maggiore è la fatica accumulata, con conseguente rallentamento dei tempi di esecuzione. Gli autori propongono e confrontano quattro scenari operativi: solo operatore umano, operatore con supporto robotico, operatore con postazione ergonomica flessibile, e combinazione di tutte e tre le soluzioni. I risultati mostrano che l'introduzione del cobot consente una riduzione del 17% dei tempi di ciclo e un aumento del 23% dell'efficienza dell'operatore. Tuttavia, l'aggiunta della postazione ergonomica, pur riducendo la fatica accumulata del 52%, comporta una diminuzione dell'efficienza del 3%. La scelta dello scenario ottimale dipende quindi dall'obiettivo prioritario: massimizzare l'efficienza produttiva o preservare il benessere dell'operatore nel lungo periodo.

Un ulteriore aspetto critico nella task allocation riguarda la sicurezza dell'operatore. Faccio et al. [5] propongono un modello matematico che introduce il vincolo della distanza tra operatore e cobot nell'ottimizzazione della task allocation. In conformità con la normativa ISO/TS 15066 [3], la velocità del cobot viene modulata progressivamente in base alla distanza dall'operatore: quando i due si trovano a distanza sufficiente il robot opera alla velocità massima consentita, mentre in prossimità dell'operatore la velocità viene ridotta fino all'arresto completo in caso di avvicinamento eccessivo. Assegnando i task in modo da massimizzare il tempo in cui operatore e robot rimangono a distanza di sicurezza, il modello proposto consente una riduzione del makespan¹ fino al 18% rispetto all'approccio tradizionale [5].

La condivisione dello spazio di lavoro tra robot e operatore introduce inevitabilmente problematiche di sicurezza. Tuttavia, una riduzione eccessiva della velocità penalizza la produttività della cella collaborativa. Lettera et al. [6], ricercatori dell'Università Politecnica delle Marche, propongono un'architettura ibrida basata su ROS2 che implementa lo scenario SSM in modo ottimizzato, combinando tre moduli. Il primo è un modulo di tracciamento dell'operatore tramite il sistema OptiTrack, che rileva in tempo reale la posizione delle mani degli operatori attraverso guanti dotati di marcatori retroriflettenti. Il secondo è un modulo di stima delle intenzioni, che utilizza un filtro di Kalman lineare per prevedere la velocità e la posizione futura dell'operatore, consentendo al robot di anticipare situazioni di rischio anziché reagire solo quando il pericolo è già imminente. Il terzo è un modulo di controllo della velocità, che regola la velocità del robot in modo continuo e progressivo in base alla distanza effettiva dall'operatore, evitando rallentamenti inutili nelle situazioni di basso rischio. Il sistema è stato validato sperimentalmente con il robot UR5e in scenari industriali con uno e due operatori simultanei.

Per risolvere il problema della task allocation in modo ottimale, si ricorre spesso ad algoritmi di ottimizzazione combinatoria, ovvero algoritmi che cercano la soluzione migliore tra un numero finito ma molto grande di possibili combinazioni, utilizzando strategie intelligenti per trovare la soluzione ottimale in tempi ragionevoli. Uno dei più noti è il TSP (Travelling Salesman Problem — Problema del Commesso Viaggiatore), che nella sua formulazione classica consiste nel trovare il percorso più breve che attraversa un insieme di città, visitandole tutte una sola volta e tornando al punto di partenza. Sebbene originariamente pensato per problemi di instradamento, il TSP è stato adattato con successo al contesto della task allocation, dove le "città" corrispondono alle stazioni di lavoro e il "percorso" rappresenta la sequenza ottimale di operazioni da eseguire.

Nel presente lavoro, l'algoritmo TSP è stato adattato per gestire due agenti che operano in parallelo — il robot e l'operatore — ciascuno con una propria matrice dei tempi di percorrenza tra le stazioni. L'algoritmo assegna i task a ciascun agente in

¹Il makespan rappresenta il tempo totale necessario per completare tutti i task, dalla partenza del primo alla conclusione dell'ultimo.

modo da minimizzare il tempo totale di completamento, sfruttando le differenze di velocità tra robot e operatore nelle diverse combinazioni di stazioni pick e place.

2.3 Pick and place industriale

Una delle applicazioni più diffuse e consolidate in ambito robotico industriale è quella del pick-and-place. L'introduzione di manipolatori collaborativi (cobot) progettati per operare in aree di lavoro condivise con l'operatore umano rappresenta, dal punto di vista dell'ottimizzazione dei processi, una delle soluzioni più efficaci per incrementare la produttività flessibile. Come descritto nei paragrafi precedenti, la programmazione e il controllo di questi sistemi deve tuttavia bilanciare rigorosi requisiti di sicurezza dinamica e strategie avanzate di task allocation.

La letteratura scientifica recente si divide principalmente tra due paradigmi per la generazione delle traiettorie e la manipolazione degli oggetti: gli approcci basati su Intelligenza Artificiale e quelli guidati dalla programmazione geometrica classica. Con l'avvento dell'Industria 4.0 e della successiva transizione verso l'Industria 5.0, sono emersi modelli basati sul Reinforcement Learning (RL) e sulla computer vision. In questi scenari, il robot sfrutta sensori di visione RGB-D per percepire l'ambiente circostante e reti neurali (come l'architettura MobileNet) per calcolare i punti di presa in tempo reale su oggetti non strutturati o disposti in modo casuale, raggiungendo tassi di successo stimati intorno all'89,9% nell'afferraggio di geometrie mai viste in fase di addestramento [7].

Tuttavia, i sistemi basati esclusivamente sull'apprendimento end-to-end e sulla visione artificiale presentano forti limiti quando l'applicazione richiede il rilascio dell'oggetto in coordinate spaziali estremamente precise (come nel caso del palettizzato industriale, dell'assemblaggio meccanico a tolleranza stretta o del posizionamento in stazioni fisse di lavorazione). In tali contesti, le incertezze del deep learning non garantiscono l'accuratezza e la ripetibilità millimetrica necessarie per evitare collisioni o fallimenti operativi.

Come evidenziato in contesti analoghi di letteratura dedicati all'integrazione di cobot [8], quando le posizioni di pick e di place sono fisse e geometricamente predefinite, la programmazione classica guidata da vincoli cinematici rigidi rimane lo standard di riferimento industriale, poiché garantisce traiettorie deterministiche, fluide e perfettamente ripetibili.

È in questo solco metodologico che si inserisce il presente lavoro di tesi. Per l'implementazione degli algoritmi di task allocation in ambiente collaborativo si è scelto di adottare un approccio di programmazione deterministica basato sul framework ROS2 [9] e sul modulo di pianificazione MoveIt2 [10]. Sebbene la letteratura evidenzia come la transizione dei driver industriali verso l'architettura distribuita DDS di ROS2 presenti tuttora criticità di stabilità legate al caricamento dei modelli URDF e dei plug-in cinematici rispetto al più maturo ROS1 [11], l'utilizzo di ROS2 si rende indispensabile per sfruttare i requisiti di real-time nativi tramite middleware RTPS

e gli standard di sicurezza avanzati essenziali per una reale interazione uomo-robot. Attraverso le librerie di pianificazione dello spazio di lavoro fornite da MoveIt2, quali OMPL per il calcolo di percorsi liberi da collisioni [12] e FCL per la collision detection in tempo reale [13], l'applicazione a stazioni fisse qui sviluppata coniuga l'altissima accuratezza dei movimenti tradizionali con la flessibilità software necessaria a gestire la redistribuzione dinamica dei compiti tra l'operatore e il manipolatore.

3 Materiale utilizzato

3.1 Robot UR5e

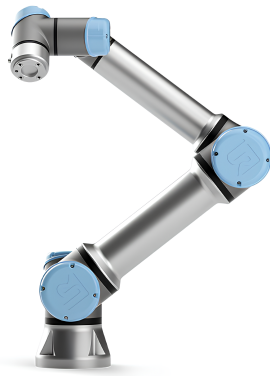


Figura 3.1: Robot collaborativo UR5e

Nel presente lavoro è stato utilizzato il robot UR5e di Universal Robots. Il nome del modello si interpreta come segue: **UR** è l'acronimo dell'azienda produttrice, **5** indica il carico utile massimo (payload) che il robot è in grado di sollevare, ovvero 5 kg, e infine la lettera **e** sta per *e-Series*, la famiglia di robot di generazione successiva rispetto alla vecchia serie CB3. La “e” riassume i concetti chiave su cui è stata progettata questa evoluzione: Evoluzione, Potenziamento, Facilità d'uso, Eccellenza e Per tutti — tutti termini che in inglese iniziano per “e”.

Le principali caratteristiche tecniche del robot sono riportate in Tabella 3.1.

Tabella 3.1: Caratteristiche tecniche del robot UR5e

Parametro	Valore
Carico utile (Payload)	5 kg
Raggio d'azione (Reach)	850 mm
Gradi di libertà	6 giunti rotanti ($\pm 360^\circ$ ciascuno)
Ripetibilità della posa	$\pm 0,03$ mm (ISO 9283)
Peso	20,6 kg (cavo incluso)
Velocità massima dei giunti	$\pm 180^\circ/\text{s}$
Grado di protezione	IP54

3 Materiale utilizzato

Il grado di protezione IP54 indica che il robot è protetto contro la polvere (cifra 5) e contro gli spruzzi d'acqua provenienti da qualsiasi direzione (cifra 4). Da queste caratteristiche si evince che il robot è adatto per task di pick and place in contesti industriali con carichi non superiori a 5 kg, in ambienti con presenza moderata di polvere e senza acqua ad elevata pressione.

Il funzionamento del braccio meccanico è reso possibile dalla presenza della Control Box, descritta nella sezione seguente.

3.1.1 Control Box



Figura 3.2: Control Box del robot UR5e

Mentre il braccio meccanico esegue fisicamente i movimenti, la Control Box gestisce tutta l'intelligenza del sistema: i calcoli cinematici, l'alimentazione elettrica e la sicurezza. Le sue funzioni principali sono le seguenti.

Elaborazione dei dati e controllo dei motori. Quando viene inviato un comando di movimento, la Control Box traduce le coordinate spaziali in impulsi elettrici da fornire ai motori, in modo che i 6 giunti presenti nel braccio si muovano coerentemente con le richieste del client. Il sistema di controllo ad anello chiuso monitora costantemente la posizione, la velocità e la corrente di ogni singolo giunto tramite encoder assoluti: in caso di spegnimento o di emergenza, al riavvio il robot è consapevole della propria configurazione, poiché conosce esattamente l'angolo di ciascun giunto.

Gestione della sicurezza. La Control Box contiene sistemi elettronici a doppio canale che monitorano costantemente lo stato del cobot. Se l'operatore tocca il robot o ostacola il suo percorso, i sensori rilevano un aumento anomalo dello sforzo e la Control Box interrompe l'alimentazione ai motori in una frazione di millisecondo, eseguendo un arresto di sicurezza.

Interfaccia di comunicazione. La Control Box funge da gateway di comunicazione tra il robot e il mondo esterno. Per la programmazione classica, alimenta

e gestisce il software PolyScope in esecuzione sul Teach Pendant. Tramite la porta Ethernet, il PC su cui è in esecuzione il nodo ROS2 si collega alla Control Box, che espone interfacce di comunicazione ad alta frequenza (500 Hz per la e-Series). ROS2 invia i flussi di traiettoria calcolati da MoveIt2 e riceve in risposta lo stato aggiornato dei giunti (*joint states*) e i dati del sensore di forza integrato.

Interfaccia I/O per gli attuatori. La Control Box dispone di ingressi e uscite (I/O) digitali e analogici a 24V, utilizzati per collegare e controllare gli strumenti di fine braccio (end-effector), qualora non siano collegati direttamente alla flangia del robot. Sul pannello frontale sono inoltre presenti le porte esterne mostrate in Figura 3.3: la porta Teach Pendant, lo slot per SD card, la porta Ethernet, le porte USB, il Mini DisplayPort e il fusibile da 10A.

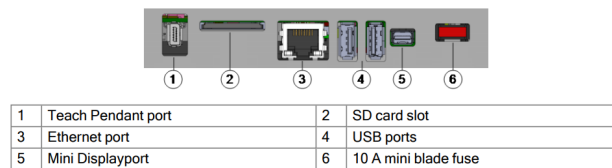


Figura 3.3: Porte esterne della Control Box UR5e: (1) Teach Pendant port, (2) SD card slot, (3) Ethernet port, (4) USB ports, (5) Mini DisplayPort, (6) 10A mini blade fuse.

3.1.2 Teach Pendant e PolyScope

Il Teach Pendant (spesso chiamato in italiano “tastiera di programmazione”) è il terminale portatile utilizzato dall’operatore per interagire fisicamente con il robot, programmarlo e movimentarlo. Nel caso della e-Series di Universal Robots, il Teach Pendant si presenta come un tablet industriale touch-screen da 12 pollici, collegato alla Control Box tramite un cavo robusto ad alta resistenza.

Nel presente lavoro il Teach Pendant è stato utilizzato per le seguenti funzioni.

Interfaccia grafica di programmazione (PolyScope). Tramite PolyScope è stata eseguita una prima programmazione dei movimenti del robot, utile per individuare le posizioni delle stazioni di pick e place. PolyScope è un linguaggio a blocchi — non una programmazione testuale classica — che consente di costruire programmi trascinando comandi come `moveJ` o `moveL` all’interno del file di programma. Viene utilizzato anche per la calibrazione e la configurazione delle impostazioni del robot.

Movimentazione manuale (Jogging e Lead-Through). Il Teach Pendant consente di muovere fisicamente il robot in due modi:

- **Jogging a schermo:** premendo le frecce virtuali sul touch-screen è possibile muovere il robot lungo gli assi cartesiani (X , Y , Z) o ruotare i singoli giunti;
- **Pulsante FreeDrive:** sul retro del Teach Pendant è presente un pulsante che, se tenuto premuto, sblocca i freni dei motori e consente di guidare il

3 Materiale utilizzato

braccio robotico direttamente con la mano, memorizzando i punti di pick e place tramite apprendimento per guida diretta (*lead-through*). Questa modalità corrisponde alla funzione di *Hand guiding* definita dalla norma ISO/TS 15066, descritta nella Sezione 2.1.

La movimentazione manuale è stata fondamentale per determinare con precisione le coordinate delle stazioni di pick e place, successivamente utilizzate nella programmazione con ROS2.

Dispositivo di sicurezza hardware. Il Teach Pendant non è un semplice monitor, ma un dispositivo di sicurezza industriale certificato. Integra infatti:

- **Pulsante di emergenza (E-Stop):** il classico fungo rosso che, se premuto, taglia istantaneamente l'alimentazione ai motori tramite i circuiti di sicurezza della Control Box;
- **Interruttore di abilitazione (Dead-man switch):** una leva sul retro che l'operatore deve mantenere premuta a metà per consentire il movimento del robot in modalità manuale. Se l'operatore stringe troppo forte o rilascia la leva, il robot si arresta immediatamente.

Nel presente lavoro il Teach Pendant è stato utilizzato in una fase iniziale per comprendere il comportamento del robot e per eseguire una prima programmazione delle stazioni di pick e place. Successivamente è stato impiegato per connettere il robot al PC e abilitare il controllo remoto tramite ROS2.

3.2 Pinza Robotiq Hand-E



Figura 3.4: Pinza elettrica parallela Robotiq Hand-E.

La Robotiq Hand-E è una delle pinze (o *gripper*) industriali più diffuse per i robot collaborativi, progettata specificamente per compiti di precisione come il pick-and-place a stazione fissa. Viene fissata alla flangia del robot tramite viti, collegata

tramite cavo e controllata elettronicamente. A differenza delle pinze pneumatiche (che funzionano ad aria compressa), la Hand-E è una pinza completamente elettrica a movimento parallelo (*Adaptive Parallel Gripper*): le due dita scorrono linearmente su una ruota dentata, mantenendosi sempre perfettamente parallele e sincronizzate. Questo design offre un vantaggio significativo nella programmazione classica, poiché è sufficiente programmare la posizione di una ganascia per ottenere il comportamento simmetrico dell'altra.

Caratteristiche meccaniche e operative.

- **Corsa (Stroke):** 50 mm di apertura massima tra le dita; può afferrare sia dall'esterno che dall'interno (allargandosi all'interno di un foro o di un tubo);
- **Forza di presa regolabile:** da 20 N a 130 N, impostabile via software — fondamentale per applicazioni collaborative o con oggetti fragili;
- **Dita personalizzabili:** i *fingertip* metallici possono essere sostituiti con dita stampate in 3D customizzate. Nel presente lavoro non è stato necessario modificare le dita originali, poiché gli oggetti da manipolare erano cubetti a geometria semplice.

Alimentazione e comunicazione. La pinza si collega direttamente alla flangia dell'UR5e tramite un connettore a 8 pin che fornisce sia l'alimentazione elettrica a 24V sia i canali di comunicazione seriale. La Control Box del robot funge da ponte, ricevendo i comandi dalla pinza e trasmettendoli al software di controllo.

Controllo avanzato: posizione, velocità e forza. La Hand-E dispone di un encoder interno e di un controllo elettronico che permette di gestire tre parametri fondamentali:

- **Posizione:** apertura e chiusura della pinza con precisione;
- **Velocità:** rapidità di chiusura delle dita;
- **Forza e rilevamento oggetto:** quando le dita incontrano la resistenza dell'oggetto, i motori si fermano alla forza prestabilita e la pinza invia un segnale di *Object Detection* al robot, confermando l'avvenuto afferraggio.

Nel presente lavoro la pinza ha rappresentato un componente fondamentale, in quanto responsabile del prelievo e del rilascio dei cubetti nelle stazioni fisse. L'attivazione della pinza veniva gestita tramite il server Python dedicato, integrato nell'architettura ROS2 descritta nel Capitolo 4.

3.3 Configurazione della stazione robotica

Il braccio robotico è installato in posizione centrale rispetto al banco di lavoro, una configurazione geometrica che garantisce la massima copertura del volume di lavoro

3 Materiale utilizzato

(*workspace*) e l'accessibilità a quasi tutti i punti del piano. Sul tavolo sono vincolate meccanicamente, tramite accoppiamenti a vite, due stazioni dedicate rispettivamente al prelievo (*pick*) e al deposito (*place*). Entrambe le stazioni presentano alloggiamenti sagomati atti a ospitare il componente da manipolare — un cubetto di materiale rigido — garantendone il posizionamento e l'orientamento univoco.

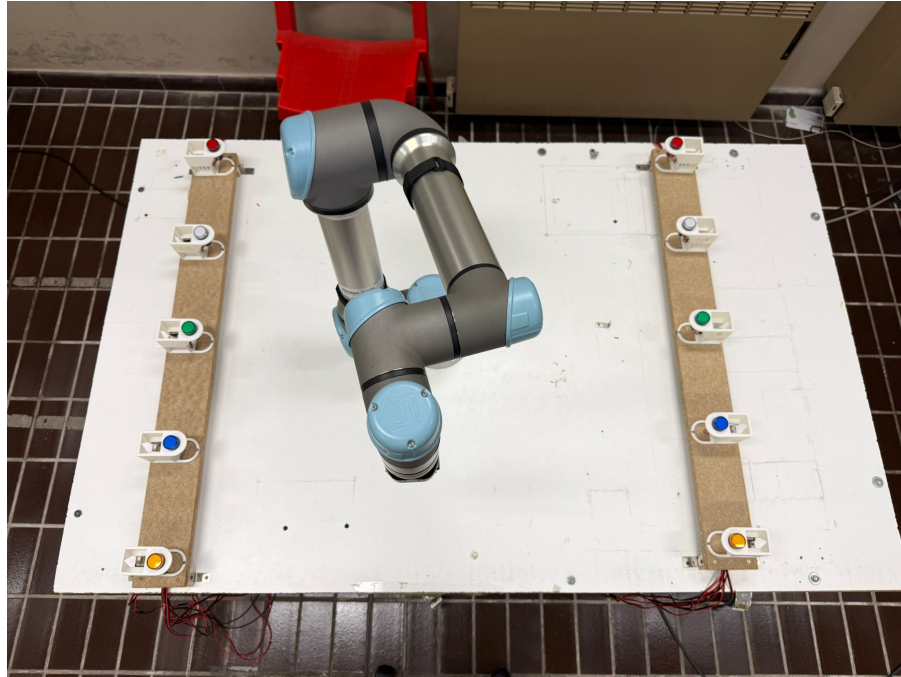


Figura 3.5: Vista dall'alto della stazione robotica: robot UR5e al centro, stazioni pick a sinistra e stazioni place a destra con i LED di segnalazione colorati.

3.3.1 Layout delle stazioni di pick-and-place

La stazione di partenza e quella di arrivo presentano un design speculare, come mostrato in Figura 3.6. Ciascuna di esse è dotata di 5 alloggiamenti geometrici a distanza fissa e predefinita. Ogni singolo alloggiamento integra un LED di segnalazione visiva e un microinterruttore di finecorsa (*mechanical limit switch*), preposto a rilevare la presenza o l'assenza del cubetto.

3.3 Configurazione della stazione robotica

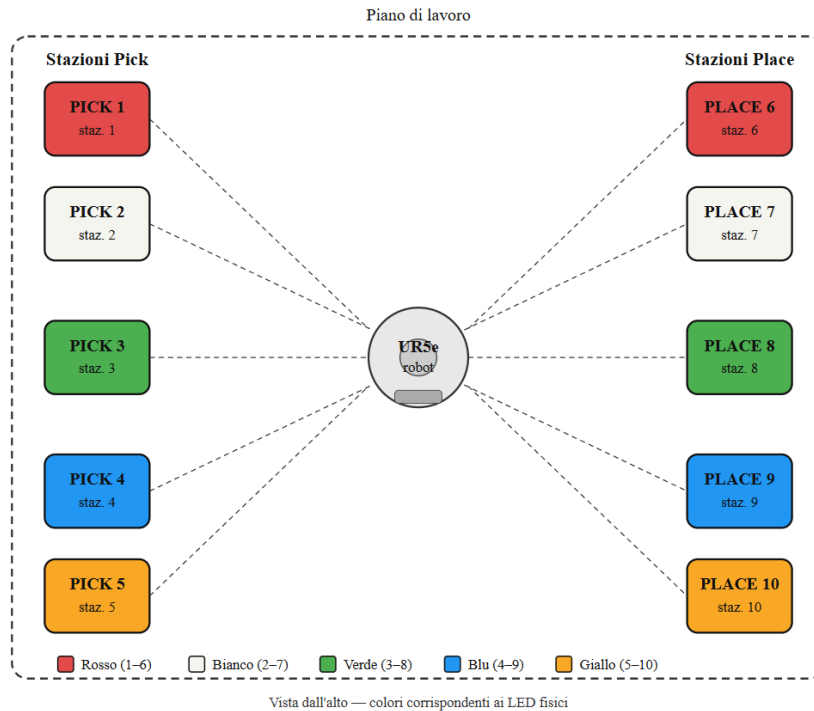


Figura 3.6: Vista dall'alto della stazione robotica: robot UR5e al centro, stazioni pick (1–5) a sinistra e stazioni place (6–10) a destra. I colori corrispondono ai LED fisici di segnalazione.

La struttura portante delle stazioni è realizzata in legno, mentre i singoli alloggiamenti sono stati sviluppati tramite manifattura additiva (stampa 3D). Nello sviluppo iniziale sono stati testati alloggiamenti stampati in resina; tuttavia, a causa della scarsa compattezza e dell'elevata fragilità del materiale, si è optato per la sostituzione con componenti realizzati in PLA (polilattico), che hanno garantito una maggiore robustezza meccanica e stabilità dimensionale. Come mostrato in 3.7, ogni alloggiamento è composto da due elementi principali: un vano quadrato atto a ospitare il cubetto e un foro circolare per l'alloggiamento del LED di segnalazione visiva.

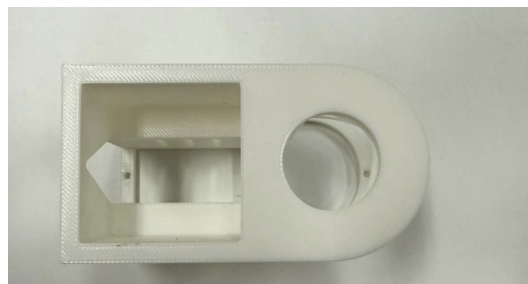


Figura 3.7: Alloggiamento stampato in 3D: vano per il cubetto (sinistra) e foro per il LED (destra).

Gli alloggiamenti presentano dei fori passanti inferiori per il cablaggio dei componenti elettronici. I conduttori elettrici dei LED e dei finecorsa convergono all'interno

3 Materiale utilizzato

di un modulo di controllo schermato, che ospita sia lo stadio di alimentazione per i segnalatori luminosi, sia le schede a microcontrollore dedicate alla logica di controllo. Dal modulo elettronico fuoriescono un cavo di alimentazione di rete e due connessioni USB, connesse al PC di controllo tramite un hub USB per consentire la trasmissione seriale dei dati.

3.3.2 Sistema LED e finecorsa (Arduino)

La gestione della sensoristica e degli attuatori luminosi è affidata a un'architettura hardware bivalente basata su piattaforma Arduino, interfacciata in ambiente MATLAB per l'elaborazione dei dati ad alto livello.

Gestione dei LED (Arduino UNO R4 WiFi). Il sistema gestisce un totale di 10 LED, uno per ciascuno dei 10 alloggiamenti complessivi. I diodi luminosi sono controllati dal microcontrollore tramite una logica digitale invertita (*active-low*), in cui lo stato logico 0 (0 V) corrisponde all'accensione del LED (*ON*) e lo stato logico 1 (5 V) corrisponde allo spegnimento (*OFF*). Ad ogni coppia di stazioni pick-place è associato un colore specifico, come indicato in Tabella 3.2.

Tabella 3.2: Corrispondenza tra stazioni e colori dei LED

Colore LED	Stazione pick	Stazione place	Pin Arduino
Rosso	1	6	D3 / D7
Bianco	2	7	D2 / D8
Verde	3	8	D4 / D9
Blu	4	9	D5 / D10
Giallo	5	10	D6 / D11

Gestione dei finecorsa (Arduino Mega 2560). La presenza del cubetto all'interno dell'alloggiamento viene rilevata meccanicamente tramite la pressione della linguetta metallica del rispettivo finecorsa, come mostrato in Figura 3.8.



Figura 3.8: Microinterruttore di finecorsa con linguetta metallica

3.3 Configurazione della stazione robotica

I pin sono configurati in modalità *pull-up*, per cui lo stato logico a riposo è 1 (5 V) e la pressione del finecorsa causa una transizione a 0 (0 V). Questa configurazione consente di implementare una routine di monitoraggio temporale dei task: la variazione di stato logico causata dal rilascio della linguetta durante la fase di pick avvia un temporizzatore software in ambiente MATLAB. Il timer arresta il conteggio non appena il cubetto viene depositato nella stazione di destinazione, attivando il rispettivo finecorsa nella fase di place.

Questo sistema di *time-tracking* permette di quantificare sperimentalmente il tempo di ciclo di ciascun task di manipolazione, fornendo i parametri empirici necessari all'algoritmo di task allocation per ottimizzare la collaborazione uomo-robot.

4 Struttura software – ROS2

4.1 Ambiente ROS2

ROS2 (Robot Operating System 2) rappresenta l'architettura middleware open-source scelta per l'orchestrazione software della cella robotica. Non si tratta di un sistema operativo nel senso tradizionale del termine, bensì di un framework di comunicazione distribuito che fornisce astrazioni hardware, driver, librerie e protocolli per semplificare lo sviluppo di applicazioni robotiche complesse.

4.1.1 Architettura ROS2

L'architettura di ROS2 si basa su una rete decentralizzata di entità computazionali denominate Nodi, ciascuna operante come processo indipendente dedicato a un compito specifico. La comunicazione tra tali nodi è mediata dal middleware DDS (*Data Distribution Service*), che supporta tre paradigmi fondamentali:

- **Topic:** Canali di comunicazione asincrona basati sul modello Publisher/Subscriber. Un nodo Publisher trasmette dati su un topic e qualsiasi nodo Subscriber li riceve in modo asincrono, senza necessità di feedback immediato. Tale approccio favorisce l'accumulo dei messaggi in code logiche, ottimizzando il flusso di dati continuo.
- **Service:** Canali di comunicazione sincrona e bidirezionale basati sul modello Client/Server. Il Client invia una richiesta e attende il completamento dell'operazione, sospendendo temporaneamente il proprio flusso di esecuzione fino alla ricezione della risposta dal Server. Questo paradigma garantisce un controllo deterministico delle azioni sequenziali.
- **Action:** Strutture di comunicazione progettate per compiti a lungo termine che richiedono un'interazione dinamica. A differenza dei Service, le Action offrono tre vantaggi strategici:
 - **Feedback continuo:** Il server fornisce aggiornamenti periodici sullo stato di avanzamento dell'operazione, consentendo al client di monitorare il processo in tempo reale.
 - **Capacità di interruzione (*Preemption*):** Il client può annullare l'attività in qualsiasi momento, garantendo un controllo reattivo in scenari operativi critici.

- **Natura *Goal-Oriented*:** Il client invia un obiettivo (*Goal*) finale, e il server opera autonomamente per soddisfarlo, gestendo internamente la logica di raggiungimento del risultato e notificando il completamento (*Result*).

4.1.2 Workspace e pacchetti

L'organizzazione del codice all'interno dell'ambiente di sviluppo avviene tramite la strutturazione di un **Workspace**. All'interno del workspace, il codice sorgente è suddiviso in **Pacchetti**, ovvero unità funzionali minime contenenti i nodi, i file di configurazione, i *launch file* e le definizioni dei messaggi personalizzati. Per questo progetto è stato configurato un workspace integrato in ambiente Linux, compilato mediante lo strumento standard `colcon`, garantendo il corretto linkaggio delle librerie C++ (`rclcpp`) e Python (`rclpy`).

4.2 Driver UR5e

Per consentire l'interfacciamento e la comunicazione bidirezionale in tempo reale tra ROS2 e il manipolatore fisico, è stato integrato il pacchetto ufficiale `ur_robot_driver`, sviluppato in sinergia da Universal Robots e dal consorzio ROS-Industrial. Questo pacchetto funge da strato di astrazione hardware (*Hardware Abstraction Layer*), traducendo le primitive di alto livello provenienti dai nodi ROS2 in comandi a basso livello decifrabili dall'unità di controllo del cobot.

4.2.1 Infrastruttura di rete e protocollo RTDE

Il collegamento fisico tra il PC di sviluppo e la Control Box dell'UR5e avviene tramite cavo Ethernet, con protocollo TCP/IP configurato sull'indirizzo IP statico 169.254.0.4 assegnato alla Control Box. Una volta stabilita la connessione, il driver sfrutta l'interfaccia nativa **RTDE** (*Real-Time Data Exchange*) di Universal Robots, che permette di sincronizzare i dati a una frequenza di 500 Hz, operando secondo una logica bidirezionale. In uscita dal robot, il driver campiona costantemente i dati degli encoder assoluti dei giunti e li pubblica sul topic `/joint_states`. In ingresso, il driver riceve le traiettorie cinematiche calcolate da MoveIt2 — di cui verrà fornita una trattazione dettagliata nella Sezione 4.3 — e le invia alla Control Box per la modulazione di corrente nei motori.

4.2.2 Procedura di inizializzazione sequenziale

L'instaurazione del canale di controllo richiede una procedura di inizializzazione sequenziale vincolata a passaggi di sicurezza sia software che hardware. Il primo passo prevede l'avvio del driver sul PC tramite il seguente comando:

```
ros2 launch ur_robot_driver ur_control.launch.py \
```



```
ur_type:=ur5e robot_ip:=169.254.0.4
```

L'esecuzione avvia il Controller Manager del framework `ros2_control`, abilitando il modulo `scaled_joint_trajectory_controller`. Solo successivamente l'operatore seleziona il programma **External Control** sul Teach Pendant e preme Play, cedendo il controllo cinematico all'ambiente ROS2. Invertire questa sequenza provocherebbe un errore di timeout.

4.2.3 Regimi di controllo e modulazione della velocità

Nel regime **Local** abbinato all'applicativo External Control, il PC acquisisce la facoltà di muovere il braccio e modularne i parametri dinamici tramite i fattori di scaling di MoveIt2. L'autorità finale rimane vincolata al Teach Pendant: l'attivazione iniziale e il ripristino dopo un arresto di emergenza richiedono l'intervento manuale. Lo slider di velocità sul pendant agisce come filtro analogico superiore rispetto ai comandi software.

Nel regime **Remote**, il PC può interfacciarsi con il *Dashboard Server* (porta 29999) per gestire autonomamente l'intero ciclo di vita del robot, eseguendo operazioni quali l'accensione, lo sblocco dei freni e il caricamento dei programmi senza ausilio umano. Questa modalità è fondamentale per l'implementazione di logiche avanzate di robotica collaborativa basate sul monitoraggio spaziale degli operatori.

4.2.4 Architettura gerarchica e limiti del driver

Il driver opera esclusivamente nello **spazio dei giunti**, senza informazioni sulla morfologia geometrica del sistema. Per superare questa limitazione, l'architettura adotta una struttura gerarchica piramidale:

1. **Flusso ascendente:** il driver pubblica gli angoli dei giunti su `/joint_states`; MoveIt2 li riceve e, tramite la cinematica diretta, calcola la posizione e l'orientamento del TCP nello spazio cartesiano;
2. **Elaborazione e collision detection:** MoveIt2 verifica che la traiettoria richiesta non intersechi ostacoli e, in caso di conflitto, calcola una traiettoria alternativa nello spazio libero;
3. **Flusso discendente:** la traiettoria validata viene tradotta in sequenze temporizzate di angoli giunto tramite cinematica inversa e inviata al driver per l'attuazione elettromeccanica.

4.3 MoveIt2

MoveIt2 è il framework di motion planning utilizzato per gestire ed eseguire i movimenti del manipolatore nello spazio. Opera definendo e mettendo in relazione diversi sistemi di riferimento: il **sistema di riferimento fisso** ancorato alla base

del robot e il **sistema di riferimento mobile** solidale con la flangia `tool0`, visibile in Figura 4.1. Lo stato del `tool0` è descritto dalla **Posa**, un vettore a 6 dimensioni:

$$X = [x, y, z, \phi, \theta, \psi]^T$$

dove (x, y, z) costituiscono il vettore di traslazione e (ϕ, θ, ψ) gli angoli di Eulero.

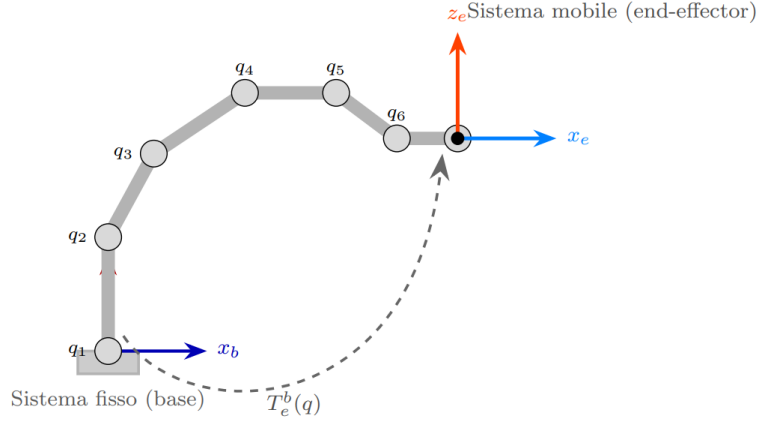


Figura 4.1: Rappresentazione schematica del manipolatore UR5e a 6 giunti: sistema di riferimento fisso $\{b\}$ ancorato alla base e sistema di riferimento mobile $\{e\}$ solidale con l'end-effector (la flangia), collegati dalla matrice di trasformazione omogenea $T_e^b(q)$.

4.3.1 Cinematica diretta (FK)

La cinematica diretta consente di determinare la posa dell'end-effector (identificato come `tool0`) nello spazio cartesiano a partire dalla configurazione angolare dei giunti $q \in \mathbb{R}^6$. Il numero di gradi di libertà (n_{gl}) del manipolatore è regolato dall'equazione di Kutzbach, che descrive la mobilità di una catena cinematica:

$$n_{gl} = 6n_{cm} - \sum_{j=1}^5 (6 - j)c_j$$

Per un manipolatore seriale come l'UR5e, composto da 6 giunti rotoidali di classe 5 ($c_5 = 1$), tale equazione si riduce al numero di giunti indipendenti, confermando che il sistema possiede 6 gradi di libertà. Questo giustifica la natura del vettore delle coordinate articolari q , che necessita di 6 componenti scalari per definire univocamente la configurazione del robot nello spazio.

Il manipolatore è modellato come una catena cinematica di corpi rigidi; la relazione tra due sistemi di riferimento solidali a giunti consecutivi ($i - 1$ e i) è descritta dalla matrice di trasformazione omogenea $A_i \in SE(3)$. Il gruppo $SE(3)$ (*Special Euclidean Group*) rappresenta l'insieme delle trasformazioni rigide (rotazioni e traslazioni) nello spazio tridimensionale. Nella sua forma generale, tale matrice si scrive come:

$$A_i = \begin{bmatrix} R_i & \mathbf{p}_i \\ \mathbf{0}^T & 1 \end{bmatrix}$$

dove $R_i \in SO(3)$ è la matrice di rotazione 3×3 che descrive l'orientamento relativo tra i frame $i - 1$ e i , e $\mathbf{p}_i \in \mathbb{R}^3$ è il vettore di traslazione corrispondente. Nel caso specifico del manipolatore UR5e, adottando la convenzione di **Denavit-Hartenberg** (DH), la matrice A_i assume la seguente forma espansa:

$$A_i = \begin{bmatrix} \cos(q_i) & -\sin(q_i) \cos(\alpha_i) & \sin(q_i) \sin(\alpha_i) & a_i \cos(q_i) \\ \sin(q_i) & \cos(q_i) \cos(\alpha_i) & -\cos(q_i) \sin(\alpha_i) & a_i \sin(q_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

La sottomatrice 3×3 superiore sinistra descrive l'orientamento relativo (rotazione), mentre la quarta colonna definisce la traslazione del frame i rispetto al frame $i - 1$. I parametri DH hanno il seguente significato fisico:

- q_i (o θ_i): angolo di rotazione attorno all'asse Z_{i-1} , che definisce il moto dinamico del giunto;
- α_i : angolo di torsione (*twist*) attorno all'asse X_i , che definisce l'orientamento spaziale relativo tra due assi di rotazione consecutivi;
- a_i : lunghezza del link, ovvero la distanza tra gli assi Z_{i-1} e Z_i misurata lungo X_i ;
- d_i : scostamento (*offset*) assiale lungo Z_{i-1} , necessario poiché i giunti consecutivi non operano generalmente sulla medesima quota assiale.

La posa globale dell'organo terminale rispetto al sistema di riferimento base $\{b\}$ è ottenuta mediante il prodotto a cascata delle singole trasformazioni parziali:

$$T_e^b(q) = A_1^b(q_1) \cdot A_2^1(q_2) \cdot A_3^2(q_3) \cdot A_4^3(q_4) \cdot A_5^4(q_5) \cdot A_6^5(q_6)$$

La posizione cartesiana X dell'organo terminale nel sistema di riferimento fisso si ricava applicando la trasformazione globale al vettore posizione P_e espresso nel frame dell'end-effector:

$$P_b = T_e^b(q) \cdot P_e \quad \Rightarrow \quad X = f(q)$$

4.3.2 Server Python

Il collegamento socket è stato implementato in Python per la disponibilità di librerie dedicate alla gestione del protocollo Modbus, come `pymodbus` e lo script ufficiale Robotiq. In Python è possibile stabilire la connessione con poche righe di

codice, mentre in C++ la gestione del Modbus TCP grezzo avrebbe richiesto librerie esterne complesse e la manipolazione manuale dei bit a basso livello.

Il codice è stato ottenuto dal repository pubblico `ros2_RobotiqGripper` del gruppo IFRA-Cranfield [8]. Il server viene avviato con il seguente comando:

```
ros2 run ros2_robotiqgripper server.py \
  --ros-args -p IPAddress:="169.254.0.4"
```

Una volta avviato, il server espone il servizio ROS2 `/Robotiq_Gripper`, che accetta richieste di tipo `RobotiqGripper` con il campo `action` impostato su `"OPEN"` o `"CLOSE"`.

4.4 Nodo `pick_place_completo` (C++)

4.4.1 Subscriber e coda comandi

Il nodo `pick_place_completo` centralizza la ricezione delle direttive operative mediante un oggetto **Subscriber** abbonato al topic `/pick_place_cmd`, configurato con una coda di ricezione pari a 10 messaggi. L'interfaccia attende messaggi di tipo `std_msgs::msg::Int32MultiArray`, che trasportano una coppia di interi: il primo definisce la stazione di prelievo (Pick, 1–10) e il secondo la stazione di rilascio (Place, 1–10).

Il nodo opera in regime di programmazione concorrente, gestendo l'interazione tra due flussi di esecuzione distinti. Un thread asincrono secondario intercetta i messaggi in ingresso tramite la funzione di callback `commandCallback`, inserendoli in una coda **FIFO**¹ (*First In, First Out*). Parallelamente, il thread principale estrae ciclicamente il comando in testa alla coda per avviare la pianificazione cinematica.

Poiché entrambi i thread condividono la medesima risorsa (`std::queue`), è stato integrato un meccanismo di mutua esclusione tramite `std::mutex` e `std::lock_guard`. Tale accorgimento garantisce che l'accesso alla coda sia atomico, impedendo conflitti di lettura/scrittura (*race conditions*)² che porterebbero all'instabilità del sistema. In questo modo, l'invio dei comandi ai driver avviene solo dopo che l'operazione di estrazione e validazione del task è stata completata con successo, garantendo la coerenza del flusso di controllo.

4.4.2 Sequenza `pick and place`

Una volta estratto un comando dalla coda FIFO, il ciclo principale avvia la sequenza deterministica di manipolazione. La sequenza coordina movimenti **MoveJ** e **MoveL** calcolati dinamicamente tramite i solutori di `moveit::core::RobotState`.

¹*First In, First Out*: struttura dati in cui il primo elemento inserito è il primo ad essere estratto, garantendo l'ordine cronologico di esecuzione dei comandi.

²Una *race condition* è una condizione di errore che si verifica quando due o più thread accedono contemporaneamente a una risorsa condivisa senza adeguata sincronizzazione, producendo risultati non deterministici e potenzialmente causando instabilità o crash del processo.

La fase di prelievo si articola nei seguenti passaggi:

1. **Approccio al Pick (MoveJ):** Il software calcola la posa cartesiana della stazione di prelievo tramite cinematica diretta e applica un offset di +13 cm sull'asse Z, generando il punto alto di approccio. Il vettore cartesiano viene convertito in angoli di giunto tramite il solutore IK (`setFromIK`). Il robot esegue una traiettoria articolare rapida verso il punto alto.

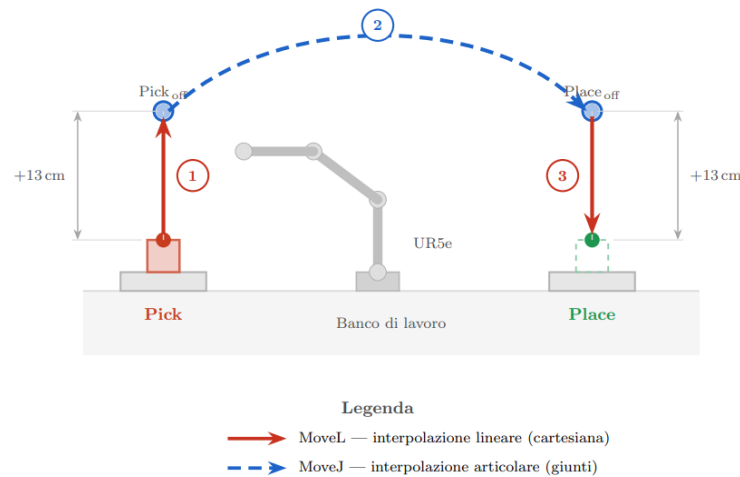


Figura 4.2: Schema della traiettoria completa pick-and-place: trasferimento MoveJ tra i punti di offset delle due stazioni.

2. **Discesa e Presa (MoveL):** Il nodo invoca la funzione `moveLinear`, che utilizza `computeCartesianPath` per imporre una discesa verticale perfetta. Raggiunto il fondo, viene inviata la richiesta di servizio "CLOSE" alla pinza.

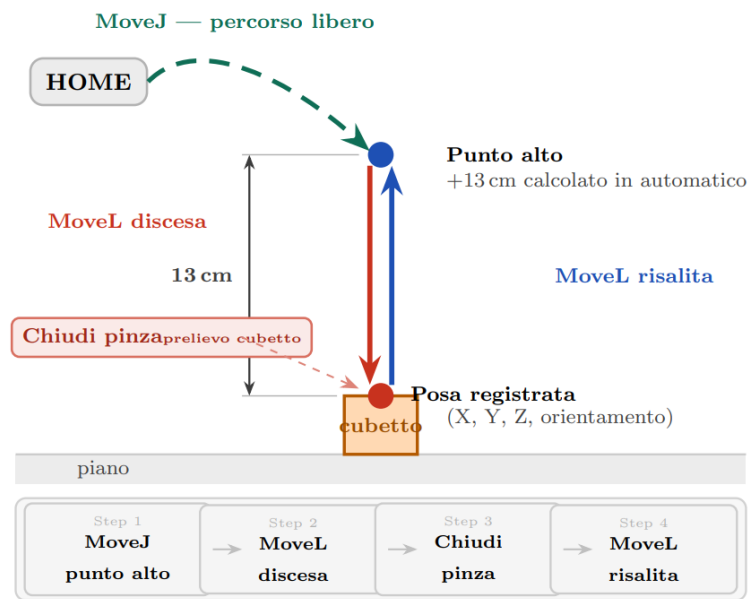


Figura 4.3: Sequenza di approccio alla stazione di pick: MoveJ verso il punto alto (+13 cm), MoveL di discesa e chiusura pinza.

3. **Risalita (MoveL):** Il manipolatore esegue la traiettoria cartesiana inversa per risalire di +13 cm.

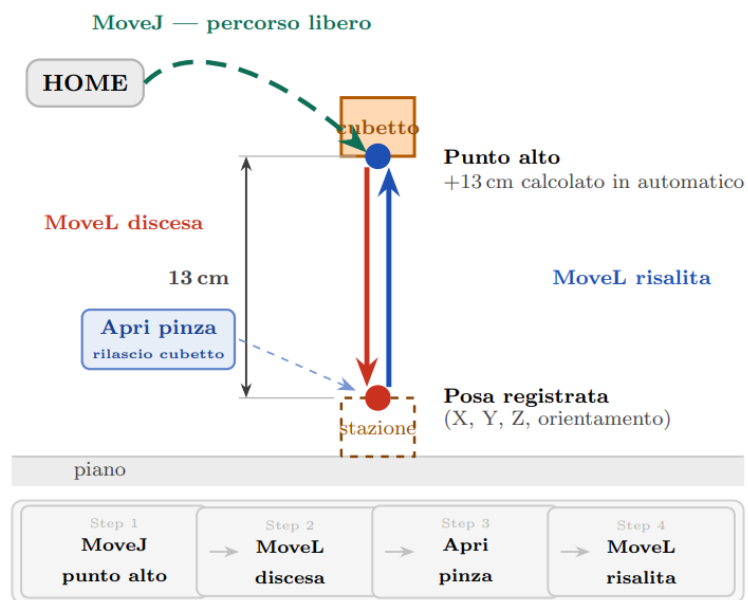


Figura 4.4: Sequenza di rilascio alla stazione di place: MoveL di discesa, apertura pinza e MoveL di risalita.

La medesima logica viene replicata per la fase di deposito: approccio MoveJ al

punto alto della stazione place, discesa MoveL, apertura pinza con "OPEN" e risalita MoveL.

4.4.3 Registrazione coordinate stazioni

La mappatura spaziale della cella è stata eseguita tramite una procedura di apprendimento manuale in modalità *Freedrive* sul Teach Pendant. La registrazione è stata effettuata con il cubetto reale inserito nell'alloggiamento e la pinza in posizione di presa, per campionare l'esatta postura dell'organo terminale nel momento cinematico più critico.

Le 10 configurazioni articolari (5 Pick e 5 Place) sono state registrate in gradi sessagesimali tramite PolyScope e successivamente convertite in radianti. I vettori risultanti sono stati inseriti direttamente nel codice sorgente C++ come matrice statica `target_joints`:

```
std::vector<std::vector<double>> target_joints = {
    {-3.6043, -2.0921, -1.3306, 4.9964, 1.5790, 2.7101}, //
    1 PICK
    {-3.2526, -1.8373, -1.7022, 5.1152, 1.5776, 3.0643}, //
    2 PICK
    {-2.8442, -1.7565, -1.8075, 5.1409, 1.5752, 3.4364}, //
    3 PICK
    {-2.4724, -1.8970, -1.6228, 5.0958, 1.5727, 3.8378}, //
    4 PICK
    {-2.2349, -2.1694, -1.2041, 4.9491, 1.5717, 4.0746}, //
    5 PICK
    { 0.8048, -2.2201, -1.1181, 4.9086, 1.5759, 3.9561}, //
    6 PLACE
    { 0.5583, -1.9799, -1.5041, 5.0541, 1.5757, 3.7104}, //
    7 PLACE
    { 0.2105, -1.8818, -1.6410, 5.0929, 1.5755, 3.3626}, //
    8 PLACE
    {-0.1630, -1.9771, -1.5038, 5.0510, 1.5753, 2.9889}, //
    9 PLACE
    {-0.4749, -2.2338, -1.0931, 4.8974, 1.5753, 2.7016}, //
    10 PLACE
};
```

La memorizzazione nello **spazio dei giunti** anziché nello spazio cartesiano offre due vantaggi fondamentali: la prevenzione delle singolarità matematiche, poiché la configurazione è stata sperimentata fisicamente sul banco, e l'eliminazione delle ambiguità cinematiche, garantendo che il robot si approcci sempre con la medesima postura dei link.

5 Struttura software - MATLAB

5.1 Introduzione a MATLAB

Il presente capitolo illustra il ruolo svolto da MATLAB nell'architettura di controllo sviluppata per la cella robotica collaborativa. L'ambiente MATLAB non si è limitato alla gestione periferica dei componenti hardware — quali i LED di segnalazione e i microinterruttori di finecorsa — ma ha operato come *supervisore logico* dell'intero sistema, coordinando l'interazione tra i dispositivi elettronici e il livello di controllo del braccio robotico.

Il flusso informativo implementato prevede che MATLAB elabori la strategia ottimale di task allocation e trasmetta i comandi al nodo di controllo `pick_place_completo` (descritto nella Sezione 4.4). Tale nodo interpreta le direttive ricevute e delega la pianificazione cinematica delle traiettorie al framework MoveIt2. La traiettoria validata viene quindi trasmessa al driver del robot tramite il protocollo RTDE, garantendo l'esecuzione sincronizzata dei movimenti sul manipolatore fisico. In questo schema, MATLAB centralizza le decisioni strategiche di alto livello, delegando a ROS2 e MoveIt2 la gestione delle operazioni cinematiche.

Il flusso completo — dalla pianificazione MATLAB fino all'esecuzione sul robot fisico — è schematizzato in Figura 5.11, discussa in dettaglio nella Sezione 5.6.

5.2 Toolbox installati

L'architettura software è stata realizzata integrando due ambienti distinti: il framework ROS2, dedicato alla gestione in tempo reale del manipolatore UR5e e alla pianificazione cinematica delle traiettorie, e l'ambiente MATLAB, che funge da supervisore logico dell'intero banco di lavoro. Tale integrazione consente la sincronizzazione tra la segnalazione luminosa dei LED, l'acquisizione degli eventi dai finecorsa e l'esecuzione delle traiettorie robotiche.

L'interoperabilità tra i diversi domini è stata resa possibile dall'impiego dei seguenti toolbox specializzati.

MATLAB Support Package for Arduino Hardware. Questo pacchetto ha consentito l'interfacciamento diretto con le schede Arduino UNO R4 WiFi e Arduino Mega 2560. Attraverso le librerie messe a disposizione, il sistema acquisisce lo stato logico dei microinterruttori di finecorsa — configurati in modalità *pull-up* — e comanda l'azionamento dei LED di segnalazione — configurati in modalità *active-low* — realizzando un monitoraggio in tempo reale degli eventi fisici sul banco di lavoro.

ROS Toolbox. Questo strumento abilita la comunicazione bidirezionale tra l'ambiente MATLAB e il framework ROS2, consentendo l'istanziamento di nodi ROS2 direttamente all'interno di MATLAB. In particolare, è stata sfruttata la funzione `ros2publisher` per la configurazione di un nodo Publisher, responsabile della trasmissione dei comandi di movimento al topic `/pick_place_cmd`:

```
pub = ros2publisher(node, '/pick_place_cmd', ...  
    'std_msgs/Int32MultiArray');
```

In questo schema, MATLAB opera come client supervisor, inviando le direttive di task allocation che il nodo `pick_place_completo` recepisce e traduce in traiettorie eseguite sul manipolatore fisico tramite MoveIt2 e il driver `ur_robot_driver`.

5.3 Interfacciamento hardware Arduino

L'architettura del banco di lavoro è stata interfacciata con due unità di controllo distinte, delegate rispettivamente alla gestione dei segnali di uscita digitale per la segnalazione luminosa e all'acquisizione degli eventi fisici provenienti dai microinteruttori di finecorsa. La scelta di adottare due microcontrollori separati — Arduino UNO R4 WiFi e Arduino Mega 2560 — è stata dettata dalla disponibilità dell'hardware consolidato nel laboratorio, nonché dalla necessità di preservare l'integrità del cablaggio già realizzato nella fase di allestimento del banco. Il circuito interno del modulo di controllo, comprendente entrambe le schede e il relativo cablaggio, è illustrato in Figura 5.1.

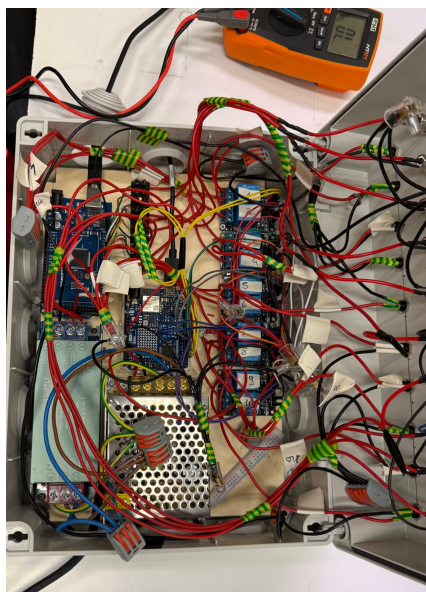


Figura 5.1: Circuito interno del modulo di controllo: schede Arduino Mega 2560 e UNO R4 WiFi, alimentatore switching, busbar e cablaggio dei LED e finecorsa.

5.3.1 Sistema di segnalazione luminosa (Arduino UNO R4 WiFi)

La gestione dell'interfaccia visiva è affidata alla scheda Arduino UNO R4 WiFi, configurata in modalità *Digital Output*. Il sistema è deputato a guidare l'operatore umano nelle attività di pick-and-place mediante una codifica luminosa: l'accensione fissa dei LED segnala i task assegnati all'operatore, mentre la modalità lampeggiante identifica le operazioni in corso di esecuzione da parte del robot.

Il pilotaggio diretto dei LED tramite i pin digitali dell'Arduino non è stato ritenuto sufficiente dal punto di vista elettrico, poiché la corrente massima erogabile da un singolo pin I/O è limitata a circa 40 mA. Per garantire l'alimentazione stabile dei dispositivi luminosi, si è pertanto interposto un modulo relè tra il pin di controllo e il circuito di potenza del LED. Il modulo impiegato è il **FC-31**, equipaggiato con il relè elettromeccanico Songle **SRD-05VDC-SL-C**, visibile in Figura 5.2.

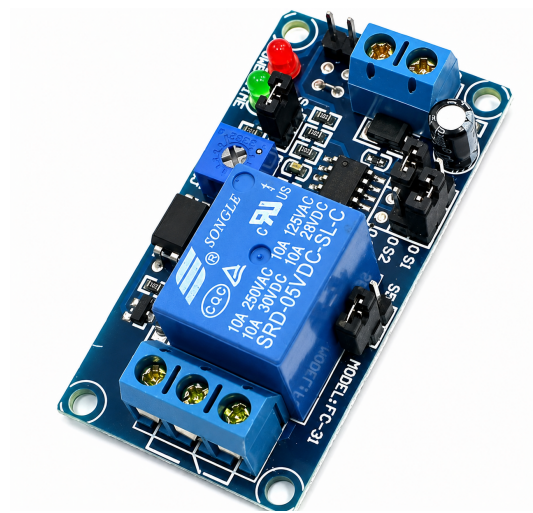


Figura 5.2: Modulo relè FC-31 con relè elettromeccanico Songle SRD-05VDC-SL-C utilizzato per il pilotaggio dei LED di segnalazione.

Il funzionamento del circuito è il seguente.

Stato ALTO (HIGH). Il pin digitale dell'Arduino porta la propria uscita a 5 V, eccitando la bobina del relè. Il contatto normalmente aperto (NO) si chiude, completando il circuito di potenza e consentendo l'accensione del LED.

Stato BASSO (LOW). Il pin è portato a potenziale zero (GND). La bobina risulta diseccitata, il contatto rimane aperto e il LED è spento.

Questa architettura garantisce il disaccoppiamento galvanico tra il circuito di controllo a bassa tensione (5 V logici dell'Arduino) e il circuito di alimentazione dei LED, proteggendo il microcontrollore da eventuali sovracorrenti e transitori induttivi generati dalla bobina del relè in fase di commutazione.

Si osserva che l'architettura attuale prevede l'utilizzo di due schede distinte per la segnalazione e l'acquisizione. Un'analisi del sistema evidenzia tuttavia la possibilità di unificare tali funzioni su un'unica scheda Arduino Mega 2560, il cui numero di

pin I/O disponibili sarebbe ampiamente sufficiente. Tale consolidamento non è stato implementato durante la fase di tesi per preservare l'integrità del cablaggio già consolidato sul banco, ma costituisce un potenziale sviluppo futuro: la centralizzazione su un'unica unità eliminerebbe la doppia istanza di comunicazione seriale verso il calcolatore, riducendo le latenze di I/O derivanti dalla gestione parallela di due distinte interfacce hardware e garantendo una maggiore stabilità del sistema in fase operativa.

In alternativa all'approccio basato su microcontrollori Arduino, la gestione dei segnali digitali di ingresso e uscita avrebbe potuto essere affidata a un **PLC** (*Programmable Logic Controller*), dispositivo industriale specificatamente progettato per operare in ambienti produttivi. I PLC offrono moduli I/O standardizzati con isolamento galvanico integrato, cicli di scansione deterministici nell'ordine del milisecondo e certificazioni di sicurezza funzionale (IEC 61508) che li rendono idonei per applicazioni collaborative conformi alla norma ISO/TS 15066. Nel contesto del presente lavoro, l'adozione della piattaforma Arduino è stata motivata dall'immediata disponibilità dell'hardware, dalla semplicità di integrazione con l'ambiente MATLAB tramite il Support Package dedicato e dall'assenza di requisiti di sicurezza funzionale certificata, trattandosi di un banco sperimentale a scopo di ricerca.

5.3.2 Sistema di acquisizione eventi (Arduino Mega 2560)

Il monitoraggio degli eventi fisici all'interno della cella di lavoro — ovvero il rilevamento dell'avvenuto prelievo o posizionamento dei cubetti nelle stazioni — è affidato alla scheda Arduino Mega 2560. Questa unità è stata selezionata in virtù dell'elevato numero di pin di ingresso digitale disponibili (54 pin I/O totali), che ha consentito di monitorare capillarmente le dieci stazioni del banco di lavoro senza ricorrere a moduli di espansione esterni.

La logica di lettura dei microinterruttori di finecorsa è implementata tramite la modalità *pull-up* interna del microcontrollore. Tale configurazione attiva via software una resistenza di pull-up (circa 50 k Ω) tra il pin e la tensione di alimentazione (+5 V), garantendo un potenziale elettrico definito in ogni condizione operativa. Il comportamento del circuito nei due stati possibili è il seguente.

Stato di riposo (finecorsa non premuto). Il circuito verso massa è aperto; la resistenza di pull-up mantiene il pin al potenziale di +5 V. Il software rileva il valore logico **HIGH**, indicando che la stazione è vuota e in attesa di attività.

Stato di attivazione (finecorsa premuto). La chiusura del contatto meccanico collega il pin direttamente al terminale GND (0 V). La tensione sul pin si azzerava e il software rileva il valore logico **LOW**. Questa transizione, che avviene con precisione istantanea al momento dell'interazione meccanica tra il cubetto e la linguetta del finecorsa, funge da trigger per l'ambiente MATLAB.

Ogni variazione di stato viene utilizzata per acquisire un *timestamp* di sistema tramite la funzione `posixtime`, consentendo di calcolare con accuratezza deterministica

la durata dei singoli task di manipolazione. La logica di temporizzazione adottata prevede che la transizione **HIGH** $\rightarrow$ **LOW** nella stazione di pick avvii il cronometro, mentre la medesima transizione nella stazione di place ne determini l'arresto. Il tempo di ciclo misurato costituisce pertanto il dato grezzo da cui viene derivata la cella corrispondente nelle matrici dei tempi T_{op} e T_{rob} .

Questo metodo di acquisizione evento-dipendente è fondamentale per alimentare l'algoritmo di ottimizzazione della task allocation: garantisce che le matrici dei tempi siano basate su misurazioni empiriche reali, prive di incertezze derivanti da stime o modelli cinematici approssimati, come discusso nella Sezione 5.4.

5.4 Algoritmo di task allocation (TSP)

L'obiettivo del sistema di supervisione è massimizzare l'efficienza della cella robotizzata attraverso un'allocazione intelligente dei task tra il manipolatore UR5e e l'operatore umano. Tale problema è stato modellato come un **Traveling Salesman Problem** (TSP) multi-agente, finalizzato alla minimizzazione del *makespan* complessivo, ovvero del tempo che intercorre tra l'inizio del primo task e il completamento dell'ultimo da parte di entrambe le risorse.

Il sistema riceve in ingresso tre parametri fondamentali che definiscono i costi di transizione tra i nodi del grafo di ottimizzazione:

- T_{rob} : matrice dei tempi di percorrenza misurati sperimentalmente dal manipolatore UR5e tra tutte le coppie di stazioni (5 pick e 5 place);
- T_{op} : matrice equivalente per l'operatore umano, che incorpora la variabilità fisiologica intrinseca nei movimenti manuali;
- T_{P_op} : parametro scalare di penalità temporale che modella il tempo di spostamento dell'operatore dal *depot* (posizione di riposo) alla prima stazione operativa.

È stata operata una distinzione nella gestione dei tempi iniziali: per il robot il tempo di *setup* è stato posto pari a zero ($T_{P_rob} = 0$), assumendo che il ciclo di lavoro inizi istantaneamente con il primo pick; per l'operatore, sebbene la variabile T_{P_op} consenta di modellare il tempo di ingresso nell'area di lavoro, essa è stata anch'essa posta pari a zero in fase sperimentale, al fine di garantire la massima omogeneità nelle metriche di confronto tra le due risorse.

5.4.1 Matrici dei tempi T_{rob} e T_{op}

Le matrici T_{rob} e T_{op} costituiscono la base informativa su cui il risolutore opera per determinare l'allocazione ottimale dei task. Ciascuna matrice è organizzata come una struttura di costo 10×10 , in cui ogni elemento (i, j) rappresenta il tempo, espresso in

secondi, necessario alla risorsa considerata per effettuare la transizione dalla stazione i alla stazione j .

La struttura della matrice è suddivisa in quattro quadranti logici che riflettono la geometria fisica del processo di pick-and-place, come illustrato in Figura 5.3.

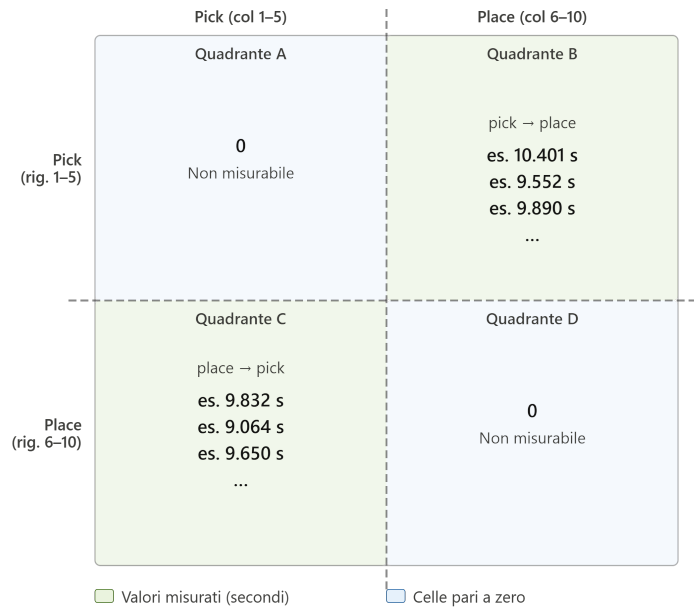


Figura 5.3: Struttura della matrice dei tempi 10×10 : Quadrante A e D (celle nulle, transizioni non operative), Quadrante B (tempi pick $\rightarrow$ place) e Quadrante C (tempi place $\rightarrow$ pick).

Quadranti A e D (transizioni interne). Questi quadranti, corrispondenti rispettivamente alle transizioni pick $\rightarrow$ pick e place $\rightarrow$ place, contengono valori nulli. Tali movimenti sono strutturalmente esclusi dal flusso di lavoro, in quanto ogni task richiede necessariamente il prelievo da una stazione di pick e il deposito in una stazione di place distinta.

Quadrante B (transizioni pick $\rightarrow$ place). Rappresenta il nucleo operativo del sistema e contiene i tempi misurati sperimentalmente per il prelievo di un cubetto da una stazione di pick e il suo successivo posizionamento in una stazione di place. Questi valori costituiscono il costo primario ottimizzato dall'algoritmo e ricadono nell'intervallo 3–7 s per il robot e 8–11 s per l'operatore.

Quadrante C (transizioni place $\rightarrow$ pick). Modella la fase di riposizionamento della risorsa al termine di un task di deposito verso una nuova stazione di prelievo. Il calcolo di questi tempi è essenziale per garantire la continuità del flusso produttivo nella sequenza multi-task.

5.4.2 Funzione `function_allocation_T_P`

La funzione `function_allocation_T_P` costituisce il nucleo computazionale dell'algoritmo di allocazione. Essa elabora le matrici dei tempi mediante un risolutore TSP, determinando la sequenza operativa che minimizza il makespan complessivo. Il processo decisionale si articola in tre fasi fondamentali.

1. Modellazione del grafo di sistema. Il problema fisico viene tradotto in un grafo pesato a 22 nodi: 10 nodi operativi per ciascuna risorsa (5 pick e 5 place) e 2 nodi *depot*, uno per risorsa, che rappresentano i punti di partenza e di sincronizzazione dallo stato HOME. L'unità logica fondamentale è il **box**: ogni box corrisponde a un singolo task fisico — un cubetto specifico gestito da una risorsa specifica — composto dal nodo di pick e dal relativo nodo di place. Il sistema conta 10 box totali (5 cubetti $\times$ 2 risorse). Le matrici dei costi vengono convertite in un vettore lineare di pesi `distt`, nel formato richiesto dalla libreria TSP3¹, che formula il problema come un problema di programmazione lineare intera.

2. Applicazione dei vincoli operativi. Per garantire la coerenza fisica della soluzione, l'algoritmo impone quattro categorie di vincoli, tradotti in matrici di adiacenza (A_{eq} , A):

- **Vincolo di univocità (Vincolo 1):** impedisce la sovrapposizione tra le risorse, vietando che il percorso di un agente si “mescoli” con quello dell'altro;
- **Vincolo di validità strutturale (Vincolo 2):** inibisce i collegamenti *mirror* non fisicamente validi, come le transizioni pick $\rightarrow$ pick o place $\rightarrow$ place;
- **Vincolo di copertura minima (Vincolo 3):** obbliga ogni risorsa a eseguire almeno un task per ciclo, evitando configurazioni in cui un agente rimane inattivo;
- **Vincolo di linearità geometrica (Vincolo 4):** con `layout = 1` (due linee parallele di stazioni), vieta salti geometricamente non ammissibili tra le due linee del banco di lavoro, penalizzando percorsi a zig-zag che degraderebbero l'efficienza dei movimenti.

Un vincolo aggiuntivo sul *depot* garantisce che ciascuna risorsa parta e ritorni esattamente una volta al proprio nodo di partenza, chiudendo il ciclo hamiltoniano².

¹La libreria TSP3 è una funzione MATLAB compilata (`.mex`), fornita nell'ambito di una collaborazione con l'Università degli Studi di Padova e integrata all'interno della funzione `function_allocation_T_P`. Il codice sorgente della libreria non è stato condiviso e il suo funzionamento interno è pertanto trattato come una *black box*: la funzione riceve in ingresso il vettore dei costi e le matrici di vincolo, restituendo in output la soluzione del problema di ottimizzazione combinatoria.

²Un *ciclo hamiltoniano* è un percorso su un grafo che visita ciascun nodo esattamente una volta e ritorna al nodo di partenza. Nel contesto del TSP, la soluzione ottimale corrisponde al ciclo hamiltoniano di costo minimo, ovvero alla sequenza di task che minimizza il tempo totale percorrendo tutti i nodi esattamente una volta prima di tornare al *depot*.

3. Ottimizzazione del makespan. Il risolutore calcola il percorso che minimizza la funzione obiettivo, definita come il massimo tra il tempo totale del robot e quello dell'operatore. L'algoritmo effettua un bilanciamento dinamico del carico (*load balancing*), assegnando i task in modo da minimizzare i tempi di inattività (*idle time*) di ciascuna risorsa. La funzione restituisce tre valori: **riss**, la sequenza ottimale di task per ciascuna risorsa; **cost**, il makespan complessivo della soluzione; e **Time**, il tempo individuale impiegato da ciascun agente, calcolato sommando i costi degli archi percorsi nella sequenza ottimale.

5.5 Sviluppo incrementale del sistema

Il sistema software è stato sviluppato secondo un approccio incrementale, articolato in una sequenza di script MATLAB denominati Step 1–9. Ogni step ha introdotto funzionalità aggiuntive rispetto al precedente, consentendo la validazione progressiva dell'hardware e della logica di controllo prima di procedere alle fasi successive.

Una scelta progettuale riguarda il vincolo geometrico imposto al flusso operativo: sia il robot che l'operatore umano transitano esclusivamente sul lato frontale del banco di lavoro per spostarsi tra le stazioni di pick e di place. Nel caso del manipolatore UR5e, tale comportamento è una conseguenza diretta della procedura di registrazione delle coordinate: le pose articolari delle stazioni sono state acquisite manualmente in modalità *Freedrive* scegliendo configurazioni che posizionano il braccio sul lato frontale, e MoveIt2 pianifica autonomamente le traiettorie di trasferimento tra tali punti seguendo il percorso cinematicamente più efficiente, che risulta naturalmente frontale. Per l'operatore umano il vincolo è invece di natura comportamentale: è stato richiesto di transitare sempre sul lato frontale del tavolo, al fine di garantire condizioni operative omogenee tra le due risorse e rendere i tempi di ciclo misurati direttamente comparabili.

5.5.1 Fase di validazione hardware e logica di base (Step 1–5)

Nella fase iniziale di sviluppo sono stati realizzati cinque script incrementali con l'obiettivo di validare progressivamente il funzionamento dell'hardware e della logica di controllo.

Step 1 — Validazione del sistema LED. Il primo script ha verificato la corretta mappatura dei pin sull'Arduino UNO R4 WiFi, accendendo i LED a coppie pick-place. In questa fase è stato riscontrato un problema di dispersione di corrente dovuto alla presenza di due schede Arduino nel medesimo circuito, che causava malfunzionamenti in alcuni LED. Il problema è stato risolto ridistribuendo i pin sull'Arduino UNO R4 WiFi, introducendo un pin non utilizzato ogni 3–4 pin contigui.

Step 2 — Validazione dei finecorsa. Il secondo script ha esteso la validazione ai microinterruttori di finecorsa, coinvolgendo l'operatore nel prelievo e nel deposito dei

cubetti. È emerso un pin difettoso su un finecorsa, risolto spostando il collegamento su un pin alternativo della scheda Arduino Mega 2560.

Step 3 — Esecuzione sequenziale con matrici simulate. Il terzo script ha introdotto l'esecuzione dei task da parte di entrambe le risorse in modalità sequenziale, con matrici dei tempi popolate da valori casuali in assenza delle misurazioni sperimentali reali.

Step 4 — Parallelismo robot-operatore. Il quarto script ha introdotto il parallelismo reale tra robot e operatore mediante una macchina a stati, consentendo alle due risorse di operare contemporaneamente e in modo indipendente. In questa fase vengono misurati i tempi di esecuzione dei singoli task (pick $\rightarrow$ place), ma non ancora i tempi di riposizionamento tra task consecutivi (place $\rightarrow$ pick).

Step 5 — Acquisizione completa dei timestamp. Il quinto script rappresenta la versione più completa della fase di validazione: mantiene il ciclo parallelo dello Step 4 e aggiunge la registrazione dei timestamp assoluti sia del pick che del place per ogni task. Ciò consente di calcolare sia il tempo pick $\rightarrow$ place (Quadrante B) sia il tempo place $\rightarrow$ pick tra task consecutivi (Quadrante C), fornendo i dati necessari per alimentare la funzione `function_allocation_T_P` con misurazioni empiriche reali.

5.5.2 Acquisizione tempi operatore (Step 7)

Lo Step 7 costituisce il modulo software dedicato alla mappatura sperimentale dei tempi di esecuzione dell'operatore umano. Per ciascuna delle 5 stazioni di pick, l'operatore esegue un ciclo completo articolato in due fasi.

Fase di andata (pick $\rightarrow$ place). L'operatore preleva il cubetto dalla stazione di pick e lo posiziona nella corrispondente stazione di place, come illustrato in Figura 5.4 e Figura 5.5.



Figura 5.4: Operatore in fase di prelievo (pick) alla stazione 1.



Figura 5.5: Operatore in fase di deposito (place) alla stazione 6.

Fase di ritorno (place $\rightarrow$ pick). L'operatore preleva il cubetto dalla stazione di place e lo riporta nella stazione di pick originaria.

Il sistema guida l'operatore tramite l'interfaccia luminosa del banco: i LED si accendono a coppie, indicando visivamente la stazione di pick di origine e la stazione di place di destinazione per ogni task. La variazione di stato dei finecorsa, intercettata

dall'Arduino Mega 2560, permette di registrare i timestamp di inizio e fine per ciascuna transizione.

Al termine del ciclo di acquisizione, il sistema genera una matrice dei tempi 10×10 contenente 50 misurazioni distinte: 25 valori nel Quadrante B, relativi alla fase di andata, e 25 valori nel Quadrante C, relativi alla fase di ritorno. Il file viene esportato nella directory `step7` con nomenclatura `operatore_YYYY_MM_DD_HH:MM:SS.txt`, dove il suffisso temporale consente di tracciare univocamente ogni sessione di acquisizione.

Questa procedura automatizzata di raccolta dati è fondamentale per garantire che l'algoritmo di allocazione (Sezione 5.4) operi su un insieme informativo realistico e aggiornato, eliminando le incertezze legate a stime teoriche e adattando il modello di ottimizzazione alle reali dinamiche esecutive dell'operatore. Un esempio di matrice T_{op} prodotta dallo Step 7 è riportato in Tabella 5.1.

Tabella 5.1: Esempio di matrice T_{op} acquisita il 09/06/2026. I valori in verde corrispondono ai Quadranti B e C (tempi in secondi); i valori nulli in blu ai Quadranti A e D (transizioni non operative).

	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	8.198	8.711	7.700	11.171	7.382
2	0	0	0	0	0	7.304	9.199	11.034	10.836	8.918
3	0	0	0	0	0	8.692	8.349	8.632	8.693	8.140
4	0	0	0	0	0	7.582	9.051	8.272	8.729	10.784
5	0	0	0	0	0	7.417	6.853	9.387	9.598	7.586
6	7.748	7.147	9.650	6.619	9.512	0	0	0	0	0
7	9.779	11.014	8.320	11.317	8.099	0	0	0	0	0
8	7.891	9.843	9.399	8.609	7.682	0	0	0	0	0
9	8.880	8.935	7.373	8.839	9.062	0	0	0	0	0
10	8.108	8.304	7.752	9.143	7.895	0	0	0	0	0

5.5.3 Acquisizione tempi robot (Step 8)

Lo Step 8 replica la logica di acquisizione dello Step 7, trasladola sul manipolatore UR5e. Mentre il modulo precedente era volto a mappare le prestazioni umane, questo script ha lo scopo di generare una mappatura precisa e deterministica dei tempi di esecuzione del robot. Per ciascuna stazione, il manipolatore esegue autonomamente il ciclo completo di prelievo e deposito, coprendo sia la fase di andata (pick $\rightarrow$ place) che quella di ritorno (place $\rightarrow$ pick), come illustrato in Figura 5.6 e Figura 5.7.

Il sistema di monitoraggio registra i timestamp in corrispondenza di ogni evento di commutazione dei finecorsa, popolando una matrice dei tempi 10×10 strutturata in modo identico a quella prodotta dallo Step 7: 25 valori nel Quadrante B, relativi ai tempi di trasporto del cubetto (pick $\rightarrow$ place), e 25 valori nel Quadrante C, relativi ai tempi di riposizionamento verso la stazione di prelievo successiva (place $\rightarrow$ pick).

Una differenza rispetto al modulo precedente riguarda l'interfaccia di feedback visivo: durante l'esecuzione, i LED lampeggiano in sincronia con il movimento del robot, fornendo un riscontro visivo immediato della stazione attiva in entrambi i sensi

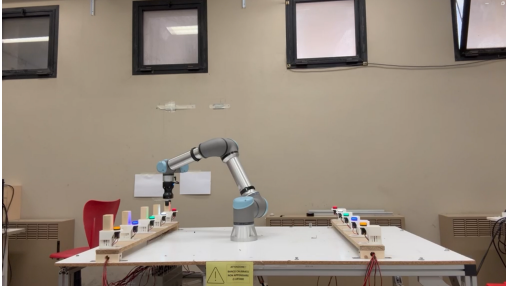


Figura 5.6: Manipolatore UR5e in fase di prelievo (pick) alla stazione 1.

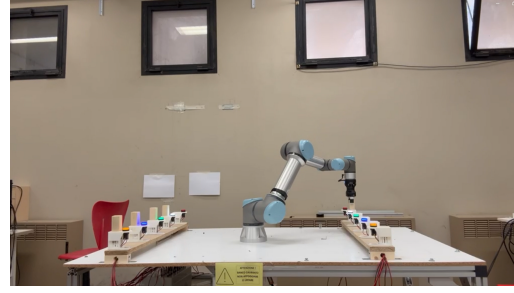


Figura 5.7: Manipolatore UR5e in fase di deposito (place) alla stazione 6.

di marcia. Il file di output viene esportato nella directory `step8` con nomenclatura `robot_YYYY_MM_DD_HH:MM:SS.txt`.

La coerenza tra le metodologie di acquisizione degli Step 7 e 8 — identiche per hardware, logica di trigger e formato di output — garantisce l'omogeneità dei dati di ingresso per il risolutore TSP, consentendo un confronto diretto tra le efficienze delle due risorse in fase di allocazione dei task. Un esempio di matrice T_{rob} prodotta dallo Step 8 è riportato in Tabella 5.2.

Tabella 5.2: Esempio di matrice T_{rob} acquisita il 09/06/2026. I valori in verde corrispondono ai Quadranti B e C (tempi in secondi); i valori nulli in blu ai Quadranti A e D (transizioni non operative).

	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	5.704	4.171	4.852	6.842	4.565
2	0	0	0	0	0	4.839	4.828	4.478	4.840	3.809
3	0	0	0	0	0	4.860	4.847	4.144	3.114	3.109
4	0	0	0	0	0	4.058	4.111	3.194	3.203	3.195
5	0	0	0	0	0	3.453	3.023	3.036	3.071	3.020
6	5.167	4.838	4.863	4.060	4.138	0	0	0	0	0
7	4.172	4.828	4.157	4.110	3.714	0	0	0	0	0
8	5.194	4.819	4.148	3.198	3.038	0	0	0	0	0
9	4.429	4.839	3.113	3.201	3.071	0	0	0	0	0
10	4.910	3.810	3.111	3.194	3.025	0	0	0	0	0

5.5.4 Elaborazione dei dati: media matrici (Step 9)

Lo Step 9 conclude il ciclo di acquisizione, integrando i dati grezzi raccolti negli step precedenti in un modello strutturato. La necessità di questa fase di elaborazione deriva dalla natura intrinsecamente differente delle due risorse.

Risorsa umana (stocasticità). L'operatore è soggetto a variabili fisiologiche quali l'affaticamento, il calo di prestazioni nel tempo o, al contrario, il miglioramento dovuto all'effetto apprendimento (*learning curve*). Per questo motivo vengono con-

dotte molteplici sessioni sperimentali, i cui dati richiedono una mediazione statistica per ottenere un valore rappresentativo da fornire al risolutore.

Risorsa robotica (determinismo). Fissata la velocità nominale del manipolatore al 100%, i tempi di esecuzione presentano un'elevata ripetibilità. La mediazione su più cicli consente tuttavia di filtrare eventuali micro-latenze introdotte dal controller o dalle comunicazioni di rete.

Lo Step 9 legge tutte le matrici prodotte dagli Step 7 e 8, ne calcola la media elemento per elemento e genera i file `T_op_new.txt` e `T_rob_new.txt` nella cartella condivisa con lo Step 6, come illustrato in Figura 5.8.

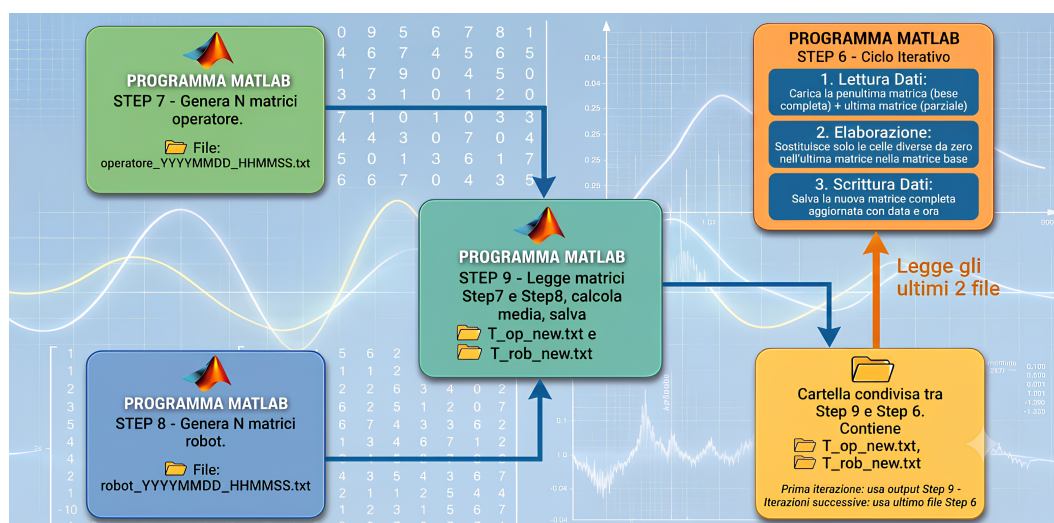


Figura 5.8: Flusso di elaborazione dati tra gli Step 7, 8, 9 e 6: acquisizione, mediazione e aggiornamento iterativo delle matrici dei tempi.

5.5.5 Task allocation e ottimizzazione (Step 6)

Lo Step 6 rappresenta la fase finale e decisionale dell'architettura software. In questo modulo il sistema integra le matrici di costo generate dallo Step 9 per eseguire la task allocation tramite la funzione `function_allocation_T_P` descritta nella Sezione 5.4.2.

Logica di aggiornamento adattivo. All'avvio, lo Step 6 legge i due file più recenti presenti nella cartella condivisa: la penultima matrice (base completa) e l'ultima matrice (parziale, prodotta dall'iterazione precedente). Le celle dell'ultima matrice con valore diverso da zero sostituiscono le corrispondenti celle della matrice base, generando una matrice aggiornata che viene salvata con un nuovo timestamp. Alla prima iterazione, il sistema utilizza direttamente l'output dello Step 9; nelle iterazioni successive, utilizza l'ultimo file prodotto dallo Step 6 stesso, realizzando un ciclo di miglioramento continuo.

Aggiornamento dei quadranti. La procedura di aggiornamento delle matrici T_{rob} e T_{op} è completa e copre entrambi i quadranti operativi: il Quadrante B (tempi

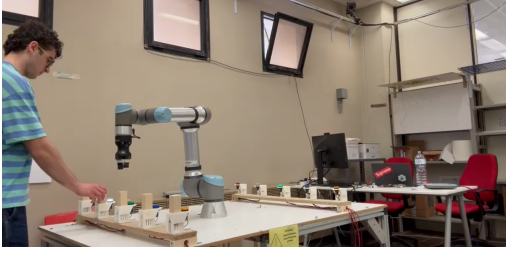


Figura 5.9: Operatore e robot operano in parallelo sulle rispettive stazioni durante l'esecuzione dello Step 6.

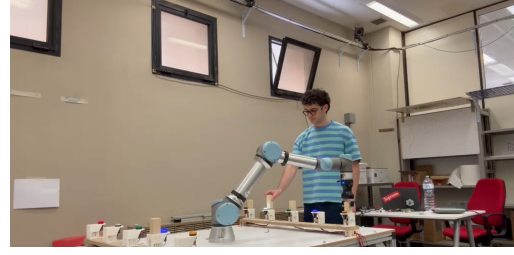


Figura 5.10: Dettaglio della collaborazione uomo-robot: entrambe le risorse eseguono task simultaneamente sulle stazioni assegnate.

pick $\rightarrow$ place), che definisce il costo netto dell'attività produttiva, e il Quadrante C (tempi place $\rightarrow$ pick), che modella i tempi di riposizionamento necessari alla risorsa per avviare il task successivo. Quest'ultimo aggiornamento è cruciale: un'allocatione che minimizza esclusivamente il tempo di esecuzione del singolo task senza considerare i tempi di riposizionamento potrebbe risultare subottimale a livello di makespan complessivo.

Convergenza del modello. Con il progredire delle iterazioni, le celle dei Quadranti B e C vengono aggiornate con misurazioni sempre più recenti, consentendo al modello di costo di convergere verso una rappresentazione fedele delle dinamiche reali della cella. Il risolutore TSP opera pertanto su dati aggiornati a ogni ciclo, garantendo che l'allocatione sia sempre ottimizzata rispetto all'ultima configurazione sperimentale disponibile.

Tabella 5.3: Aggiornamento iterativo della matrice T_{op} nello Step 6: confronto tra due iterazioni consecutive. Le celle evidenziate in giallo indicano i valori aggiornati con le misurazioni più recenti.

Prima iterazione (10:00:00)										
	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	9.323	8.711	8.134	11.171	9.406
2	0	0	0	0	0	9.259	9.199	11.034	10.836	11.623
3	0	0	0	0	0	8.692	8.349	8.632	8.693	8.140
4	0	0	0	0	0	7.582	9.051	8.272	8.729	10.784
5	0	0	0	0	0	7.417	6.853	11.479	9.598	7.586
6	8.813	9.064	9.650	6.619	9.512	0	0	0	0	0
7	9.779	11.014	8.320	11.317	8.099	0	0	0	0	0
8	8.336	9.843	9.399	8.609	9.361	0	0	0	0	0
9	8.880	8.935	7.373	8.839	9.062	0	0	0	0	0
10	10.307	10.857	7.752	9.143	7.895	0	0	0	0	0

Iterazione successiva (10:20:00)										
	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	9.323	8.711	8.134	11.171	7.382
2	0	0	0	0	0	7.304	9.199	11.034	10.836	11.623
3	0	0	0	0	0	8.692	8.349	8.632	8.693	8.140
4	0	0	0	0	0	7.582	9.051	8.272	8.729	10.784
5	0	0	0	0	0	7.417	6.853	9.387	9.598	7.586
6	8.813	7.147	9.650	6.619	9.512	0	0	0	0	0
7	9.779	11.014	8.320	11.317	8.099	0	0	0	0	0
8	8.336	9.843	9.399	8.609	7.682	0	0	0	0	0
9	8.880	8.935	7.373	8.839	9.062	0	0	0	0	0
10	8.108	10.857	7.752	9.143	7.895	0	0	0	0	0

5.6 Integrazione MATLAB-ROS2

L'integrazione tra MATLAB e ROS2 costituisce l'infrastruttura di comunicazione dell'intera cella automatizzata. Il flusso dati tra i moduli software è illustrato in Figura 5.11, che riporta l'architettura suddivisa nei quattro terminali logici T_1 - T_4 già descritti nel Capitolo 4.

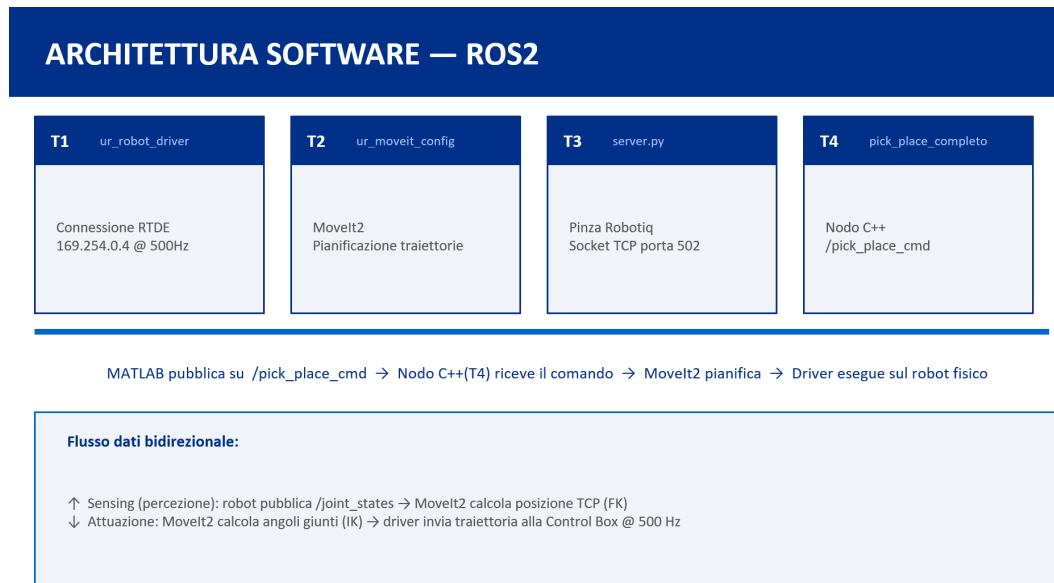


Figura 5.11: Architettura software ROS2: flusso dati tra MATLAB (client supervisor), il nodo `pick_place_completo` (T_4), MoveIt2 (T_2) e il driver UR5e (T_1).

I quattro moduli cooperano per garantire il movimento sincronizzato del manipolatore: T_1 gestisce la connessione RTDE con la Control Box, T_2 si occupa della pianificazione delle traiettorie tramite MoveIt2, T_3 interfaccia la pinza Robotiq via socket TCP e T_4 riceve i comandi di alto livello dal supervisor MATLAB.

5.6.1 Publisher sul topic /pick_place_cmd

MATLAB opera come client supervisor, interagendo con l'architettura ROS2 attraverso il paradigma Publisher/Subscriber. Il sistema inizializza un nodo ROS2 e configura un Publisher sul topic `/pick_place_cmd`. Il comando trasmesso è un messaggio di tipo `std_msgs/Int32MultiArray` contenente una coppia di interi che identificano rispettivamente la stazione di pick e quella di place assegnate al robot dall'algoritmo di task allocation:

```
% Inizializzazione nodo e publisher
node = ros2node('/matlab_supervisor');
pub = ros2publisher(node, '/pick_place_cmd', ...
    'std_msgs/Int32MultiArray');

% Composizione e invio del comando
msg_vuoto.data = int32([pick_r, place_r]);
send(pub, msg_vuoto);

% Registrazione timestamp di invio
t_pick_r(k_r) = posixtime(datetime('now'));
```


Questo invio non è unilaterale: nel momento in cui il messaggio viene pubblicato, il sistema attende la conferma di esecuzione prima di procedere, secondo il meccanismo di sincronizzazione descritto nella Sezione 5.6.2.

5.6.2 Comunicazione con `pick_place_completo`

Il meccanismo di sincronizzazione si completa attraverso un canale di feedback tra il nodo `pick_place_completo` e MATLAB. Una volta ricevuto il comando sul topic `/pick_place_cmd`, il nodo T_4 avvia la sequenza di pick-and-place: MoveIt2 pianifica la traiettoria, il driver T_1 la esegue sul manipolatore fisico e la pinza T_3 effettua le operazioni di presa e rilascio. Al termine dell'esecuzione, il nodo pubblica un messaggio di conferma su un topic di feedback; MATLAB, in ascolto su tale canale, riceve il segnale e dichiara il task come completato, consentendo l'aggiornamento della matrice dei costi e l'avvio del task successivo.

Questa comunicazione a chiusura di anello garantisce la sicurezza del sistema: MATLAB rimane costantemente informato sullo stato del robot, evitando la sovrapposizione tra movimenti consecutivi e assicurando che il conteggio dei tempi sia basato su eventi fisici reali piuttosto che su stime teoriche.

L'integrazione è completata dalla gestione hardware tramite le due schede Arduino, inizializzate con le rispettive modalità operative: `DigitalOutput` per i pin di controllo dei LED e `pullup` per i pin di lettura dei finecorsa. Tale configurazione garantisce che ogni evento manuale dell'operatore — prelievo o deposito di un cubetto — generi un segnale deterministico acquisito da MATLAB in modo sincrono rispetto alle azioni del manipolatore, mantenendo la coerenza temporale dell'intero ciclo di collaborazione uomo-robot.

6 Pipeline del sistema completo e risultati

Il presente capitolo analizza la sequenza operativa della cella robotica collaborativa, illustrando l'interazione tra i moduli hardware e software descritti nei capitoli precedenti. La pipeline di lavoro è concepita per garantire che ogni comando — dal livello di supervisione (MATLAB) a quello di attuazione (robot fisico) — sia validato e sincronizzato prima dell'esecuzione.

6.1 Architettura generale e sequenza di avvio

L'architettura del sistema si basa su una sequenza di attivazione gerarchica, necessaria a garantire l'integrità dei flussi di dati tra i moduli software. L'avvio dell'infrastruttura ROS2 richiede l'attivazione coordinata di quattro terminali, ciascuno con una funzione specifica, da eseguire nell'ordine indicato, come schematizzato in Figura 5.11.

1. **Terminale 1 — Driver RTDE (`ur_robot_driver`):** stabilisce la connessione con la Control Box del robot a 500 Hz, garantendo la bidirezionalità del flusso dati tra il PC e il manipolatore fisico.
2. **Terminale 2 — Pianificatore (`ur_moveit_config`):** carica la configurazione cinematica del manipolatore e inizializza il framework MoveIt2 per la gestione della pianificazione delle traiettorie.
3. **Terminale 3 — Server pinza (`server.py`):** avvia il server Python che gestisce l'interfacciamento socket sulla porta 502 per il controllo della pinza Robotiq Hand-E.
4. **Terminale 4 — Nodo C++ (`pick_place_completo`):** agisce come orchestratore dei movimenti atomici di pick e place, ricevendo i comandi dal supervisore MATLAB tramite il topic `/pick_place_cmd` e delegando la pianificazione a MoveIt2.

Una volta inizializzata l'infrastruttura ROS2, si avvia il supervisore MATLAB. La comunicazione segue un modello a cascata: ogni istruzione di movimento emessa da MATLAB attraversa l'intera catena di nodi, garantendo che ogni vincolo — cinematico o di sicurezza — sia verificato prima dell'attuazione sul robot fisico.

6.2 Protocollo operativo del supervisore

La pipeline logica del supervisore MATLAB si articola in tre fasi sequenziali, corrispondenti agli step di analisi e controllo descritti nel Capitolo 5.

Fase di caratterizzazione (Step 7 e Step 8). Il sistema esegue l'acquisizione sperimentale dei tempi di esecuzione dell'operatore e del robot. Questa fase è propedeutica alla costruzione del modello informativo su cui si baserà l'ottimizzazione: le matrici T_{op} e T_{rob} vengono popolate con misurazioni empiriche reali, acquisite tramite il sistema di rilevamento a finecorsa descritto nella Sezione 5.3.2.

Fase di elaborazione (Step 9). Questo modulo opera in modo indipendente da ROS2, eseguendo la mediazione statistica delle matrici temporali raccolte nelle sessioni precedenti. La normalizzazione filtra le variabilità stocastiche intrinseche all'operatore umano, fornendo al risolutore TSP un insieme di dati rappresentativo e privo di outlier. I file risultanti `T_op_new.txt` e `T_rob_new.txt` vengono depositati nella cartella condivisa con lo Step 6.

Fase di esecuzione collaborativa (Step 6). Rappresenta il culmine della pipeline operativa. In questa fase il sistema avvia il ciclo di collaborazione uomo-robot: MATLAB legge le matrici aggiornate, esegue l'allocazione ottimale dei task tramite la funzione `function_allocation_T_P` e trasmette i comandi al Terminale 4. Ogni task viene validato, attuato sul manipolatore fisico e confermato tramite il canale di feedback, chiudendo il ciclo di lavoro in modalità parallela tra robot e operatore.

6.3 Risultati sperimentali

Al fine di validare l'efficacia dell'approccio di collaborazione uomo-robot proposto e valutarne la scalabilità in un contesto industriale, sono state condotte 21 sessioni di test sperimentali. L'analisi si è focalizzata sull'ottimizzazione dei processi, monitorando le prestazioni del sistema in relazione alle configurazioni definite nel capitolo precedente (Step 6). I risultati ottenuti sono stati sintetizzati attraverso un'analisi grafica, suddivisa nelle sezioni seguenti, che illustra il comportamento del sistema e l'efficienza degli algoritmi di task allocation implementati.

6.3.1 Stazioni di pick più utilizzate: Robot vs Operatore

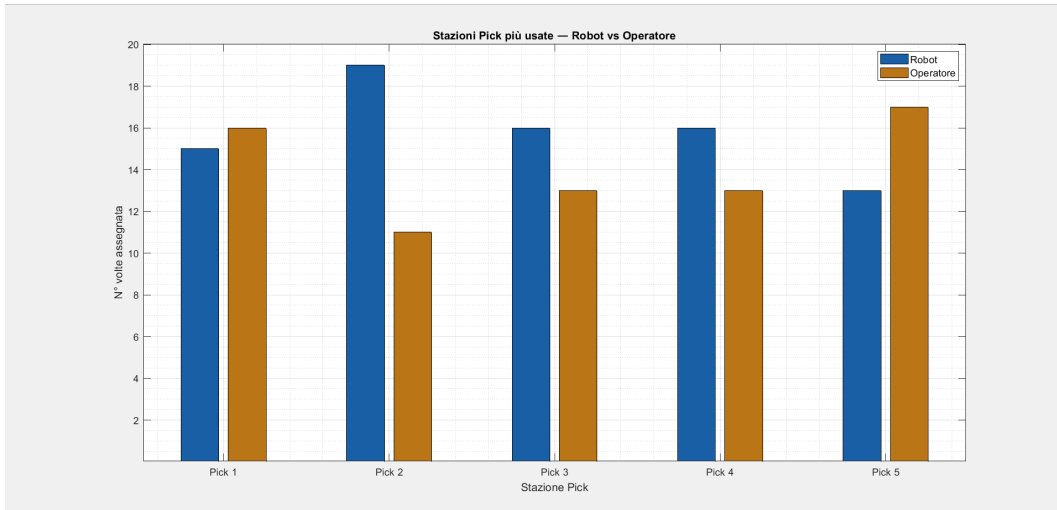


Figura 6.1: Distribuzione delle assegnazioni per le stazioni di pick tra robot e operatore, aggregata sulle 21 sessioni sperimentali.

Tabella 6.1: Numero di assegnazioni per ciascuna stazione di pick, robot vs operatore, aggregate sulle 21 sessioni sperimentali.

Risorsa	Pick 1	Pick 2	Pick 3	Pick 4	Pick 5
Robot	15	19	16	16	13
Operatore	16	11	13	13	17

Come riportato in Figura 6.1 e Tabella 6.1, si osserva che la stazione Pick 2 risulta significativamente più utilizzata dal robot (19 volte) rispetto all'operatore (11 volte). Questo comportamento è attribuibile alla posizione centrale della stazione nel layout del banco di lavoro, descritto nella Sezione 3.3.1: in assenza di sensori di rilevamento della presenza dell'operatore, quest'ultimo era costretto ad attendere il passaggio del robot prima di accedere alle stazioni centrali, aumentando il proprio tempo di esecuzione su quelle posizioni. L'algoritmo, apprendendo questa inefficienza attraverso l'aggiornamento iterativo delle matrici dei tempi, ha progressivamente assegnato tali task al robot.

Al contrario, le stazioni Pick 1 e Pick 5, posizionate agli spigoli del banco di lavoro (cfr. Figura 3.6), risultano più frequentemente assegnate all'operatore (rispettivamente 16 e 17 volte, contro 15 e 13 del robot). Questo è spiegabile sia dalla maggiore accessibilità di tali posizioni per l'operatore — che può raggiungerle senza interferire con le traiettorie del robot — sia dal vincolo imposto inizialmente che prevedeva un unico corridoio di transito condiviso tra robot e operatore, descritto nella Sezione 5.5. In particolare, la stazione Pick 5 risulta più conveniente per l'operatore in quanto riduce il percorso complessivo verso le stazioni di place.

Le stazioni Pick 3 e Pick 4 presentano invece una distribuzione più bilanciata verso il robot (16 assegnazioni ciascuna, contro 13 per l'operatore), posizione intermedia coerente con la loro collocazione centrale nel layout del banco.

6.3.2 Stazioni di place più utilizzate: Robot vs Operatore

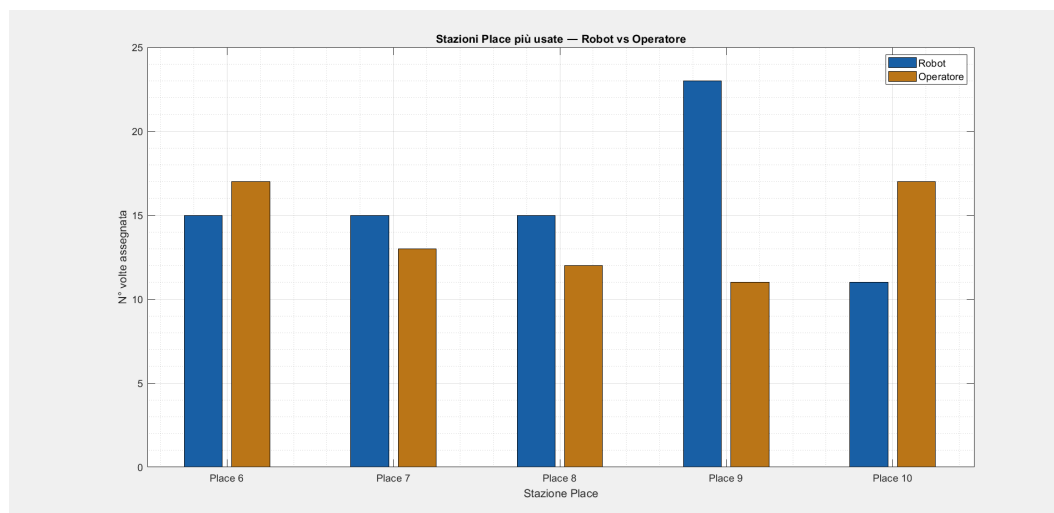


Figura 6.2: Distribuzione delle assegnazioni per le stazioni di place tra robot e operatore, aggregata sulle 21 sessioni sperimentali.

Tabella 6.2: Numero di assegnazioni per ciascuna stazione di place, robot vs operatore, aggregate sulle 21 sessioni sperimentali.

Risorsa	Place 6	Place 7	Place 8	Place 9	Place 10
Robot	15	15	15	23	11
Operatore	17	13	12	11	17

Per quanto riguarda le stazioni place, come riportato in Figura 6.2 e Tabella 6.2, si osserva un andamento speculare rispetto alle stazioni pick. La stazione Place 9 presenta la differenza più marcata, con il robot che deposita 23 volte contro le sole 11 dell'operatore. Questo risultato è coerente con quanto osservato per le stazioni pick centrali: la posizione di Place 9 nel layout (cfr. Sezione 3.3.1, Figura 3.6) risulta più facilmente raggiungibile dal robot, mentre l'operatore — dovendo rispettare il vincolo del corridoio di transito condiviso — impiegava tempi maggiori per raggiungerla, rendendo tale stazione sistematicamente più vantaggiosa per il robot agli occhi dell'algoritmo di allocazione.

Al contrario, la stazione Place 10 mostra un comportamento opposto, con l'operatore che deposita 17 volte contro le 11 del robot. Ciò è spiegabile con il fatto che, pur transitando comunque dal lato frontale del banco, l'operatore raggiunge

Place 10 mantenendosi il più possibile lontano dalla zona centrale di lavoro del robot, riducendo il rischio di interferenza e potendo quindi operare con maggiore continuità. Questo comportamento è confermato dall'andamento inversamente proporzionale tra Place 9 e Place 10: le stazioni che il robot predilige per il deposito sono esattamente quelle che l'operatore tende ad evitare, e viceversa. Tale complementarietà dimostra che l'algoritmo TSP ha appreso efficacemente le preferenze e i vincoli fisici dei due agenti, convergendo verso una distribuzione dei task che minimizza le interferenze spaziali e il carico temporale complessivo della cella.

6.3.3 Heatmap delle transizioni

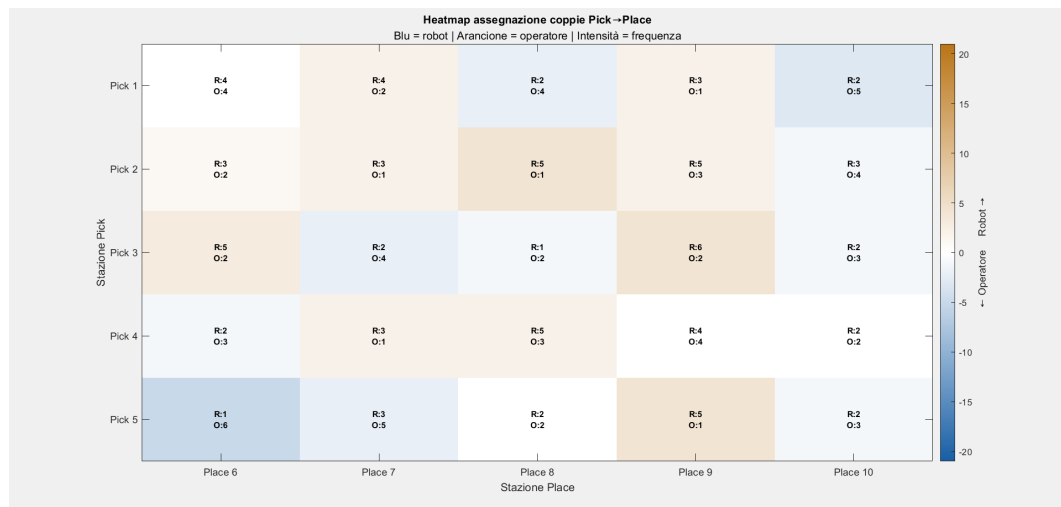


Figura 6.3: Heatmap della distribuzione delle coppie pick → place assegnate alle due risorse nel corso delle 21 sessioni sperimentali. Le tonalità arancioni indicano una maggiore assegnazione al robot (R), le tonalità blu una maggiore assegnazione all'operatore (O).

La heatmap riportata in Figura 6.3 mostra la distribuzione delle coppie pick → place assegnate alle due risorse nel corso delle 21 sessioni sperimentali. Il colore di ciascuna cella indica la preferenza dell'algoritmo: tonalità arancioni indicano una maggiore assegnazione al robot, mentre tonalità blu indicano una preferenza per l'operatore.

La coppia Pick 5 → Place 6 risulta la più frequentemente assegnata all'operatore (R:1, O:6). Questo comportamento è spiegabile considerando il layout del banco di lavoro, descritto nella Sezione 3.3.1: la stazione Pick 5, posizionata allo spigolo del banco, consente all'operatore di raggiungere Place 6 percorrendo un tragitto breve e privo di interferenze con le traiettorie del robot. Place 6, essendo la stazione di deposito più lontana dalla zona di lavoro centrale del robot, riduce al minimo il rischio di collisione, risultando quindi la destinazione più sicura e conveniente per l'operatore.

Al contrario, le coppie centrali — come Pick 2 → Place 8, Pick 2 → Place 9, Pick 3 → Place 6 e Pick 4 → Place 8 — risultano prevalentemente assegnate al robot,

6 Pipeline del sistema completo e risultati

confermando quanto osservato nei grafici precedenti (Sezioni 6.3.1 e 6.3.2): le posizioni centrali del banco comportano tempi più elevati per l'operatore a causa della necessità di attendere il passaggio del robot.

Si segnala tuttavia la presenza di alcune eccezioni, come la coppia Pick 3 → Place 7 (R:2, O:4), preferita dall'operatore nonostante la posizione relativamente centrale. Questo suggerisce che l'algoritmo non adotta una strategia rigidamente geometrica, ma apprende le preferenze reali misurate sperimentalmente, tenendo conto di fattori difficilmente modellabili a priori.

Infine, la coppia Pick 1 → Place 10 presenta una situazione speculare rispetto a Pick 5 → Place 6: l'operatore preleva da Pick 1 — stazione allo spigolo opposto — e deposita in Place 10, percorrendo comunque il lato frontale del banco ma mantenendosi il più possibile lontano dalla zona centrale di lavoro del robot. Anche in questo caso l'algoritmo ha appreso e consolidato questa preferenza nel corso delle sessioni.

6.3.4 Distribuzione dei task per sessione

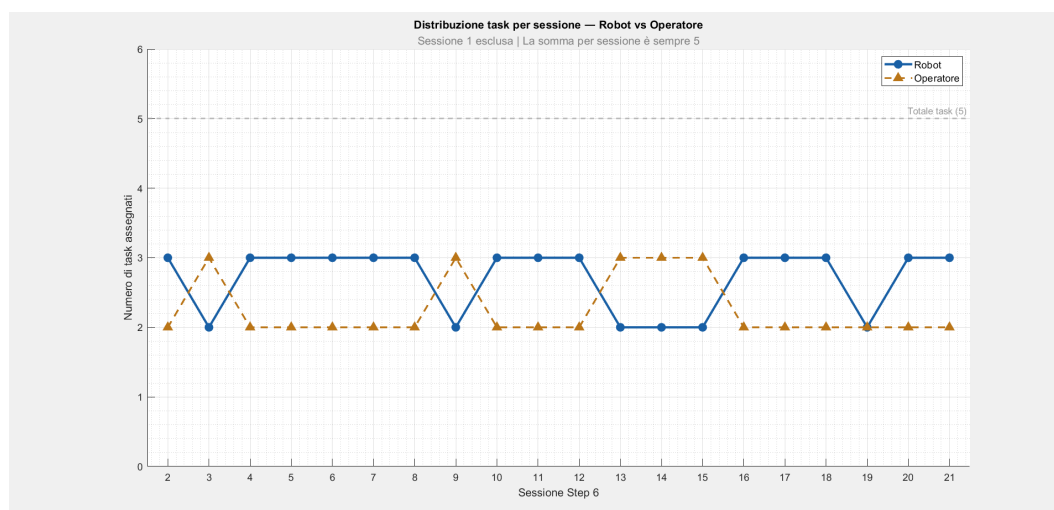


Figura 6.4: Distribuzione del numero di task rilevati per robot e operatore per ciascuna sessione sperimentale (Sessione 1 esclusa), basata sul conteggio delle celle variare nel Quadrante B della matrice dei tempi.

La Figura 6.4 mostra la distribuzione dei task tra robot e operatore nelle 21 sessioni sperimentali. Si osserva che nella maggior parte delle sessioni il robot esegue 3 task mentre l'operatore ne esegue 2, per un totale costante di 5 task per sessione. Il sistema impone infatti, come descritto nella Sezione 5.4.2, che entrambe le risorse eseguano almeno un task per ciclo (Vincolo di copertura minima, Vincolo 3): la distribuzione 5-0 risulta pertanto strutturalmente esclusa a priori dall'algoritmo. La distribuzione 4-1, pur essendo formalmente ammissibile dal vincolo di copertura minima, non viene mai osservata nelle sessioni sperimentali: un simile sbilanciamento

comporterebbe l'insorgere di tempi di inattività (*idle time*) sulla risorsa meno carica, costretta ad attendere il completamento dell'altra prima di poter ricevere nuovi task, con un conseguente aumento del tempo complessivo della sessione. L'algoritmo TSP, minimizzando la funzione obiettivo, converge pertanto verso distribuzioni più bilanciate come 3-2 o 2-3.

Si segnalano tuttavia le sessioni 3, 9, 13, 14 e 15, in cui la distribuzione si inverte: l'operatore esegue 3 task e il robot 2. Queste sessioni corrispondono ai picchi osservabili nel grafico dell'evoluzione del tempo cumulativo riportato nella Sezione 6.3.5, confermando che un maggiore carico di lavoro sull'operatore — risorsa più lenta — produce un incremento del tempo totale della sessione.

Si segnala infine un caso particolare nella sessione 19, in cui il grafico indica 2 task per il robot e 2 per l'operatore, per un totale apparente di 4 task anziché 5. Tale discrepanza non è riconducibile a un errore di esecuzione — tutti e 5 i task sono stati completati correttamente — bensì a una limitazione del metodo di conteggio adottato: il grafico rileva i task contando le celle variate nel Quadrante B della matrice dei tempi rispetto alla sessione precedente. Nella sessione 19, uno dei task eseguiti ha prodotto un tempo di esecuzione identico al valore già presente nella matrice, non generando alcuna variazione nella corrispondente cella e risultando pertanto non conteggiato. Uno sviluppo futuro del sistema prevede la registrazione esplicita del numero di task eseguiti per ciascuna sessione direttamente nello Step 6, indipendentemente dalle variazioni delle matrici dei tempi, come discusso nella Sezione 7.2.

6.3.5 Evoluzione del tempo cumulativo per sessione

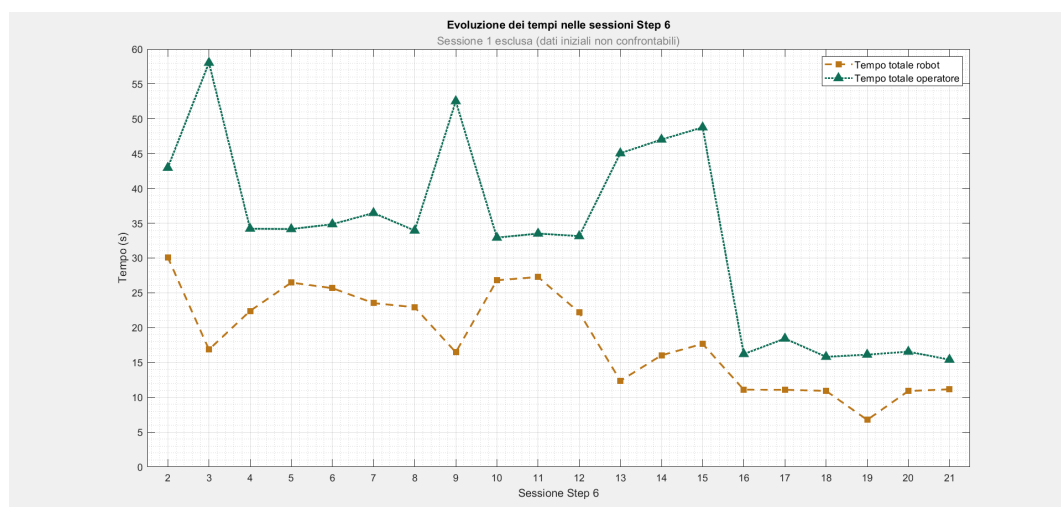


Figura 6.5: Evoluzione dei tempi cumulativi per sessione del robot (arancione) e dell'operatore (verde) nelle sessioni dello Step 6 (Sessione 1 esclusa).

La Figura 6.5 mostra l'evoluzione del tempo cumulativo per sessione del robot e dell'operatore, calcolato come somma delle celle variate rispetto alla sessione precedente — includendo sia i tempi di andata (pick $\rightarrow$ place, Quadrante B di Figura 5.3) che i tempi di ritorno (place $\rightarrow$ pick, Quadrante C di Figura 5.3). Si precisa che tale metrica non corrisponde al makespan effettivo della sessione — che richiederebbe la misurazione del tempo assoluto trascorso dal primo pick eseguito da una delle due risorse fino all'ultimo place completato — bensì rappresenta una stima indiretta del carico di lavoro assegnato a ciascuna risorsa, utile a evidenziarne l'andamento relativo nel corso delle sessioni.

Analogamente al grafico precedente (Sezione 6.3.4), la sessione 1 viene esclusa dalla visualizzazione: in tale sessione la variazione sarebbe calcolata rispetto a una matrice iniziale composta interamente da zeri, con 50 celle che passerebbero da zero a valori non nulli, producendo una somma artificialmente elevata non confrontabile con le sessioni successive e priva di significato ai fini dell'analisi.

Si osserva che il tempo cumulativo dell'operatore risulta sistematicamente superiore a quello del robot in tutte le sessioni analizzate, confermando che l'operatore rappresenta la risorsa critica che condiziona maggiormente la durata complessiva della sessione. I picchi nelle sessioni 3, 9, 13, 14 e 15 sono direttamente riconducibili alla distribuzione dei task evidenziata nella Sezione 6.3.4: in tali sessioni l'operatore esegue 3 task anziché 2, comportando la somma di 5 celle — 3 nella zona pick $\rightarrow$ place e 2 nella zona place $\rightarrow$ pick — a fronte delle 4 celle sommate nelle sessioni standard in cui l'operatore esegue 2 task.

Infine, si nota che con il progredire delle sessioni le due curve tendono progressivamente a convergere verso un medesimo valore asintotico: il tempo cumulativo dell'operatore si riduce sensibilmente, avvicinandosi a quello del robot, che invece rimane più stabile lungo l'intero arco delle sessioni. Questo fenomeno indica che l'algoritmo sta ottimizzando la distribuzione dei task in modo da bilanciare progressivamente il carico tra le due risorse, riducendo i tempi di inattività (*idle time*) della risorsa meno impegnata. Una misurazione rigorosa del makespan effettivo della cella, basata sul timestamp assoluto tra il primo pick e l'ultimo place, costituisce uno sviluppo necessario per la validazione quantitativa completa di questo comportamento, ed è discussa nella Sezione 7.2.

6.3.6 Evoluzione della somma totale delle celle della matrice dei tempi

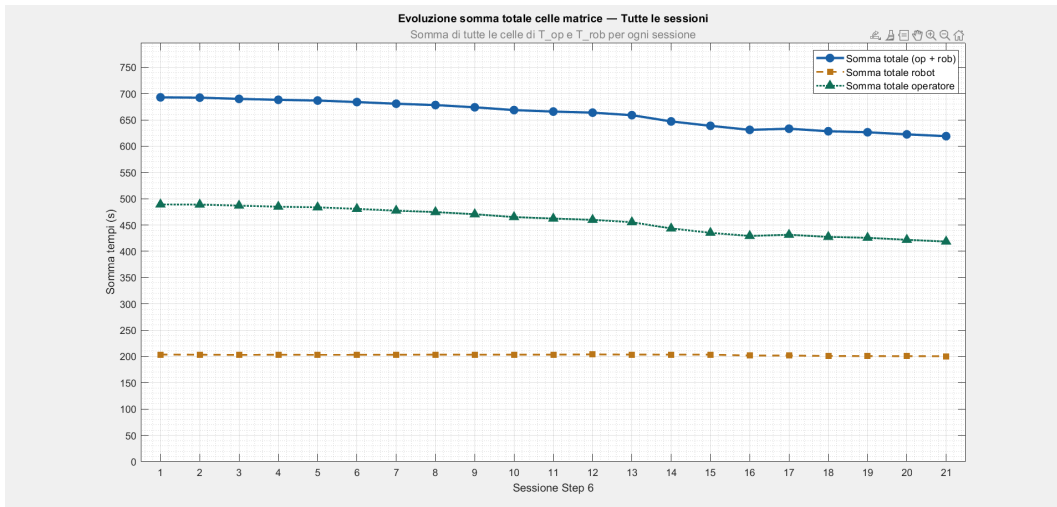


Figura 6.6: Evoluzione della somma totale delle celle delle matrici T_{op} e T_{rob} per ciascuna delle 21 sessioni sperimentali: somma totale combinata (blu), somma del robot (arancione) e somma dell'operatore (verde).

La Figura 6.6 mostra l'evoluzione della somma totale di tutte le celle delle matrici T_{op} e T_{rob} per ogni sessione, includendo sia le celle variate che quelle rimaste invariate rispetto alla sessione precedente. A differenza del grafico precedente (Sezione 6.3.5), che considera solo le variazioni sessione per sessione, questo grafico offre una visione globale dello stato della conoscenza accumulata dal sistema nel corso dell'intero processo di apprendimento.

Si osservano tre andamenti distinti. La somma totale dell'operatore presenta una riduzione progressiva e significativa nel corso delle 21 sessioni, passando da circa 490s nella sessione 1 a circa 420s nella sessione 21. Questo decremento riflette il miglioramento continuo delle stime dei tempi dell'operatore: con l'accumulo delle misurazioni reali, i valori inizialmente sovrastimati vengono progressivamente sostituiti da tempi effettivi più bassi, aggiornati tramite media mobile, come descritto nello Step 9 (Sezione 5.5.4).

La somma totale del robot risulta invece sostanzialmente costante per tutta la durata delle sessioni, attestandosi intorno a 200s. Questo comportamento è coerente con la natura deterministica del robot, che opera a velocità nominale fissa e produce tempi di esecuzione stabili, con variazioni dell'ordine di pochi centesimi di secondo.

La somma totale combinata operatore-robot presenta un andamento decrescente analogo a quello dell'operatore, poiché la componente del robot — essendo quasi costante — non contribuisce in modo significativo alla variazione complessiva. L'andamento della curva totale è pertanto dominato dall'evoluzione dei tempi dell'operatore, confermando ulteriormente quanto osservato nella Sezione 6.3.5: l'operatore rappresenta la variabile determinante nel processo di ottimizzazione del sistema.

6.4 Matrici dei tempi aggiornate

Il confronto tra la matrice acquisita nella prima sessione sperimentale e quella prodotta alla ventunesima sessione consente di valutare quantitativamente l'evoluzione del modello di costo nel corso delle iterazioni. Le celle evidenziate in giallo nelle tabelle di destra indicano i valori aggiornati rispetto alla sessione iniziale.

Matrice T_{op} — confronto tra sessione 1 e sessione 21. Come riportato in Tabella 6.3, la matrice dei tempi dell'operatore presenta variazioni significative in quasi tutte le celle operative. Le riduzioni più marcate si osservano nelle celle (5, 6), (10, 5) e (4, 8), con decrementi rispettivamente del $-46,3\%$, $-48,8\%$ e $-41,4\%$ rispetto alla prima sessione. Tale fenomeno è riconducibile all'effetto *learning curve*: con il progredire delle sessioni, l'operatore acquisisce familiarità con il layout del banco di lavoro, riducendo i tempi di esecuzione. Il modello di costo converge quindi verso una rappresentazione più fedele delle reali prestazioni dell'operatore, eliminando la sovrastima iniziale dovuta alla scarsa confidenza con il sistema.

Tabella 6.3: Confronto tra T_{op} alla sessione 1 (08/06/2026) e alla sessione 21 (12/06/2026). Le celle in giallo indicano i valori aggiornati; le celle in verde i Quadranti B e C; le celle in blu i Quadranti A e D (zeri).

Sessione 1											Sessione 21 celle variate										
	1	2	3	4	5	6	7	8	9	10		1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	10.401	9.552	9.890	11.171	9.406	1	0	0	0	0	0	8.198	8.711	7.700	11.171	6.872
2	0	0	0	0	0	9.259	9.199	11.034	11.377	11.597	2	0	0	0	0	0	7.304	9.199	11.034	10.836	6.398
3	0	0	0	0	0	8.895	9.776	9.835	9.503	10.071	3	0	0	0	0	0	8.692	8.349	8.632	8.693	8.140
4	0	0	0	0	0	9.584	9.051	8.781	9.123	11.244	4	0	0	0	0	0	7.582	9.051	5.145	6.101	10.784
5	0	0	0	0	0	8.158	8.540	11.479	9.598	8.750	5	0	0	0	0	0	4.379	6.016	9.387	9.598	7.586
6	9.832	9.064	9.882	8.409	10.538	0	0	0	0	0	6	7.748	7.147	9.650	5.881	9.512	0	0	0	0	0
7	10.726	11.014	9.742	11.317	10.059	0	0	0	0	0	7	9.779	11.014	8.320	11.317	8.099	0	0	0	0	0
8	10.132	9.843	10.738	9.133	9.361	0	0	0	0	0	8	7.891	9.843	9.399	8.609	7.682	0	0	0	0	0
9	8.880	9.417	8.076	9.240	9.062	0	0	0	0	0	9	8.880	8.935	7.373	8.839	9.062	0	0	0	0	0
10	10.307	10.819	9.598	9.552	9.113	0	0	0	0	0	10	8.108	8.304	7.752	9.143	4.669	0	0	0	0	0

Matrice T_{rob} — confronto tra sessione 1 e sessione 21. La matrice dei tempi del robot, riportata in Tabella 6.4, mostra invece variazioni notevolmente più contenute, generalmente nell'ordine dell'1–5%, con poche eccezioni che raggiungono il 15–23% (celle (2, 9) e (3, 6)). Questo comportamento è coerente con il carattere deterministico del manipolatore: a parità di velocità nominale, i tempi di esecuzione del robot sono intrinsecamente ripetibili e non soggetti a variabilità fisiologica. Le piccole fluttuazioni residue sono attribuibili a micro-latenze di rete nella comunicazione RTDE e a variazioni nei tempi di pianificazione di MoveIt2.

Il confronto tra le due matrici evidenzia pertanto un risultato significativo: mentre il modello del robot converge rapidamente già dalle prime sessioni, il modello dell'operatore richiede un numero maggiore di iterazioni per stabilizzarsi, a causa della variabilità comportamentale intrinseca alla risorsa umana. Questo giustifica la necessità del ciclo di aggiornamento iterativo implementato nello Step 6, descritto nella Sezione 5.5.5.

6.4 Matrici dei tempi aggiornate

Tabella 6.4: Confronto tra T_{rob} alla sessione 1 (08/06/2026) e alla sessione 21 (12/06/2026). Le celle in giallo indicano i valori aggiornati; le variazioni sono contenute nell'ordine dell'1–5%, coerentemente con il comportamento deterministico del manipolatore.

Sessione 1											Sessione 21 celle variare										
	1	2	3	4	5	6	7	8	9	10		1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	5.754	4.143	4.838	6.907	4.492	1	0	0	0	0	0	5.704	4.382	4.852	6.842	4.565
2	0	0	0	0	0	4.835	4.838	4.492	4.836	3.798	2	0	0	0	0	0	4.839	4.828	3.862	3.726	3.228
3	0	0	0	0	0	4.833	4.816	4.144	3.107	3.109	3	0	0	0	0	0	3.912	4.847	4.144	3.540	3.070
4	0	0	0	0	0	4.141	4.145	3.105	3.107	3.107	4	0	0	0	0	0	4.058	3.611	3.315	3.203	3.195
5	0	0	0	0	0	3.453	3.107	3.103	3.111	3.105	5	0	0	0	0	0	3.453	3.023	3.036	3.071	3.020
6	5.217	4.834	4.836	4.143	4.138	0	0	0	0	0	6	5.167	4.838	4.863	4.060	4.138	0	0	0	0	0
7	4.144	4.838	4.126	4.144	3.798	0	0	0	0	0	7	4.172	4.828	4.157	4.110	3.714	0	0	0	0	0
8	5.180	4.833	4.148	3.109	3.105	0	0	0	0	0	8	5.194	4.819	4.148	3.198	3.038	0	0	0	0	0
9	4.494	4.835	3.106	3.105	3.111	0	0	0	0	0	9	4.429	4.839	3.113	3.201	3.071	0	0	0	0	0
10	4.837	3.799	3.111	3.106	3.110	0	0	0	0	0	10	4.910	3.810	3.111	3.194	3.025	0	0	0	0	0

7 Conclusioni e sviluppi futuri

Il lavoro svolto ha consentito la progettazione e l'implementazione di un sistema di collaborazione uomo-robot (*Human-Robot Collaboration* — HRC) in un ambiente di pick-and-place industriale. Il sistema ha dimostrato di costituire una soluzione robusta e adattiva per l'ottimizzazione dei processi produttivi in celle collaborative.

7.1 Sintesi dei risultati raggiunti

Architettura robotica integrata. È stata realizzata una cella di lavoro governata dal framework ROS2/MoveIt2, capace di interpretare comandi di alto livello con precisione millimetrica e di gestire la pianificazione delle traiettorie in tempo reale, garantendo l'assenza di collisioni e configurazioni singolari. L'architettura gerarchica adottata — con MATLAB come supervisore logico e i quattro nodi ROS2 come strato di attuazione — ha dimostrato robustezza operativa nel corso delle sessioni sperimentali condotte presso il MIRLab.

Ottimizzazione dinamica tramite TSP. L'algoritmo di task allocation non si è limitato a un'assegnazione statica, ma ha dimostrato capacità di auto-apprendimento iterativo. Il sistema affina costantemente la propria conoscenza dei tempi di ciclo delle risorse (robot e operatore), aggiornando le matrici di costo (T_{rob} e T_{op}) per minimizzare progressivamente il carico temporale complessivo della cella. Come evidenziato dai grafici dell'evoluzione dei tempi cumulativi (Sezione 6.3.5) e della somma totale delle celle (Sezione 6.3.6), dopo numerose sessioni sperimentali si osserva un netto miglioramento dei risultati, con una progressiva convergenza dei tempi dell'operatore verso quelli, più stabili, del robot.

Supervisione tramite pipeline MATLAB. L'integrazione hardware e software è risultata completa e funzionale. La pipeline MATLAB funge da supervisore intelligente, coordinando non solo il robot tramite il protocollo ROS2, ma anche l'interazione umana tramite segnali luminosi (LED) e feedback fisici (finecorsa), confermando l'efficacia dell'architettura proposta per applicazioni di HRC (*Human-Robot Collaboration*) in contesti di ricerca.

7.2 Sviluppi futuri

Nonostante i risultati conseguiti dal prototipo, l'architettura presenta ampi margini di evoluzione per un'applicazione in contesti industriali reali.

Misurazione rigorosa del makespan. L'analisi sperimentale condotta nel Capitolo 6 si è basata su una metrica indiretta, calcolata come somma delle celle aggiornate nelle matrici dei tempi a ogni sessione. Tale approccio, sebbene utile per evidenziare le tendenze relative tra robot e operatore, non costituisce una misurazione rigorosa del makespan effettivo della cella. Uno sviluppo necessario consiste pertanto nell'implementazione di un cronometraggio basato sul timestamp assoluto dell'intera sessione: l'avvio del timer al momento del primo pick eseguito da una qualsiasi delle due risorse, e l'arresto al momento dell'ultimo place completato. Questa misurazione fornirebbe un dato direttamente confrontabile con la definizione formale di makespan adottata nell'algoritmo di task allocation (Sezione 5.4.2), consentendo una validazione quantitativa più solida dell'effettiva riduzione del tempo totale di esecuzione nel corso delle sessioni sperimentali.

Registrazione esplicita dei task per sessione. Il metodo attuale di conteggio dei task si basa sul rilevamento delle celle variate nel Quadrante B della matrice dei tempi rispetto alla sessione precedente. Tale approccio introduce una limitazione metodologica: qualora un task venga eseguito con un tempo identico al valore già presente nella matrice, la cella corrispondente non risulta variata e il task non viene conteggiato, producendo una sottostima apparente del numero di task eseguiti — come osservato nella sessione 19 (Sezione 6.3.4).

Uno sviluppo necessario consiste nell'introduzione di un contatore esplicito all'interno dello Step 6, indipendente dalle variazioni delle matrici dei tempi: ad ogni task completato, il sistema incrementerebbe un contatore dedicato per ciascuna risorsa, salvando tale informazione insieme alle matrici aggiornate. Questo consentirebbe una tracciabilità precisa del numero di task eseguiti per sessione, disaccoppiando il conteggio dall'aggiornamento delle matrici e rendendo l'analisi sperimentale più robusta e affidabile.

Conformità normativa e sicurezza (ISO/TS 15066). L'evoluzione naturale del sistema prevede l'integrazione delle strategie di sicurezza definite dalla norma ISO/TS 15066, in particolare il SSM (*Speed and Separation Monitoring*). L'adozione di sensori di sicurezza certificati permetterebbe al robot di modulare dinamicamente la propria velocità in funzione della distanza dall'operatore, garantendo l'incolumità umana senza arrestare il ciclo produttivo.

Si osserva che il vincolo del corridoio di transito condiviso adottato nel presente lavoro (Sezione 5.5) non è stato introdotto per sopperire all'assenza di sensori di sicurezza, bensì per garantire l'omogeneità delle condizioni sperimentali tra le due risorse. In assenza di tale vincolo, l'algoritmo di task allocation avrebbe potuto apprendere a distribuire i task separando spazialmente robot e operatore — indirizzandoli verso aree di lavoro o percorsi differenti per ridurre le interferenze reciproche — rendendo i tempi misurati per le due risorse meno direttamente confrontabili. L'introduzione di un sistema SSM costituirebbe pertanto uno sviluppo distinto e complementare: una volta validata la metodologia di task allocation in condizioni omogenee, l'integrazione del monitoraggio dinamico della sicurezza consentirebbe di rilassare il vincolo di

transito, permettendo all'algoritmo di sfruttare anche la separazione spaziale come ulteriore leva di ottimizzazione, in aggiunta al bilanciamento dei tempi di esecuzione.

Tracciamento dinamico tramite sistemi di motion capture. Per supportare le strategie di sicurezza, si prevede l'integrazione di sistemi di *motion capture* come OptiTrack. Tale tecnologia consentirebbe un monitoraggio continuo della posizione dell'operatore nello spazio tridimensionale, fornendo al controllore dati granulari per la pianificazione reattiva delle traiettorie.

Predittività tramite filtro di Kalman. L'implementazione di un filtro di Kalman permetterebbe di stimare le intenzioni e le traiettorie future dell'operatore, trasformando il robot da risorsa reattiva a predittiva. Qualora il sistema rilevasse che l'operatore si sta dirigendo verso una stazione assegnata al manipolatore, quest'ultimo potrebbe riallocare autonomamente il proprio target, aggiornando le matrici dei tempi in tempo reale per preservare l'efficienza complessiva della cella.

Migrazione del protocollo di comunicazione MATLAB–ROS2. Nel corso delle sessioni sperimentali è stato riscontrato un comportamento anomalo ricorrente: all'avvio del programma MATLAB, i LED di segnalazione e i finecorsa risultavano correttamente operativi, ma il manipolatore non riceveva i comandi di movimento. Tale problema è riconducibile alla natura asincrona del paradigma Publisher/Subscriber adottato: il messaggio inviato da MATLAB sul topic `/pick_place_cmd` veniva pubblicato prima che il nodo `pick_place_completo` completasse l'inizializzazione, andando quindi perso. Una soluzione strutturale consisterebbe nella migrazione verso il paradigma **Service** o, preferibilmente, verso il paradigma **Action** di ROS2. Il Service garantisce la sincronizzazione bloccante tra client e server — MATLAB attenderebbe la conferma di esecuzione prima di procedere al task successivo — eliminando la possibilità di messaggi non recepiti. Il paradigma Action estende ulteriormente questo modello, aggiungendo la possibilità di ricevere feedback intermedi durante l'esecuzione del movimento e di annullare un comando in corso, caratteristiche particolarmente utili in un sistema collaborativo in cui le condizioni operative possono variare dinamicamente.

Ottimizzazione dell'infrastruttura elettronica. A livello di prototipazione hardware, è possibile semplificare il sottosistema di controllo consolidando le funzioni attualmente distribuite tra le due schede Arduino su un'unica unità Arduino Mega 2560. La densità di pin I/O di quest'ultima è ampiamente sufficiente a gestire sia le linee di controllo dei LED sia quelle di lettura dei finecorsa, senza ricorrere alla modulazione PWM — non richiesta per la segnalazione luminosa a intensità fissa. Il risultato sarebbe un sistema più compatto, economico e facilmente manutenibile, con un'unica istanza di comunicazione seriale verso il calcolatore.

Bibliografia e Sitografia

- [1] GM Insights. Collaborative manufacturing robots market size. <https://www.gminsights.com/it/industry-analysis/collaborative-manufacturing-robots-market>, 2024. Accessed: 2025.
- [2] Claudio Taesi, Francesco Aggogeri, and Nicola Pellegrini. COBOT applications — recent advances and challenges. *Robotics*, 12(3):79, 2023.
- [3] International Organization for Standardization. ISO/TS 15066:2016 — Robots and robotic devices: Collaborative robots. Technical report, ISO, 2016.
- [4] Alessio Baratta, Antonio Cimino, Francesco Longo, Giovanni Mirabelli, and Letizia Nicoletti. Task allocation in human-robot collaboration: A simulation-based approach to optimize operator’s productivity and ergonomics. *Procedia Computer Science*, 232:688–697, 2024.
- [5] Maurizio Faccio, Irene Granata, and Raffaele Minto. Task allocation model for human-robot collaboration with variable cobot speed. *Journal of Intelligent Manufacturing*, 35:793–806, 2024.
- [6] Gaetano Lettera, Daniele Costa, and Massimo Callegari. A hybrid architecture for safe human-robot industrial tasks. *Applied Sciences*, 2024. Submitted manuscript.
- [7] Natanael Magno Gomes, Felipe Nascimento Martins, José Lima, and Heinrich Wörtche. Reinforcement learning for collaborative robots pick-and-place applications: A case study. *Automation*, 3(1):223–241, 2022.
- [8] David Cruz García. Aplicación de un cobot para pick-and-place inteligente en ros. Technical report, Universidad de Salamanca, 2023.
- [9] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.
- [10] David Coleman, Ioan Şucan, Sachin Chitta, and 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, 2014.

- [11] Hyunjong Choi, Yecheng Xiang, and Hyoseung Kim. Priority-driven real-time scheduling in ros 2: Potential and challenges. In *2022 59th ACM/IEEE Design Automation Conference (DAC)*, pages 1351–1354. IEEE, 2022.
- [12] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.
- [13] Jia Pan Pan, Sachin Chitta, and Dinesh Manocha. Fcl: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, pages 3859–3866. IEEE, 2012.