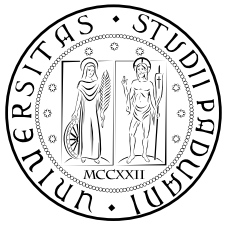


UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI TECNICA E GESTIONE DEI SISTEMI INDUSTRIALI
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA MECCATRONICA

TESI DI LAUREA MAGISTRALE

STUDIO DI UN MODELLO DI BILANCIAMENTO IN UN SISTEMA COLLABORATIVO UOMO-ROBOT

Relatore: Maurizio Faccio

Correlatore: Giovanni Boschetti

Laureando: Mattia Milanese
1159846-IMC

ANNO ACCADEMICO: 2019-20

SOMMARIO

In questo elaborato viene studiato un metodo di allocazione dei task per un sistema collaborativo uomo-robot. La soluzione proposta si basa sulla definizione di un modello matematico la cui risoluzione conduca ad una schedulazione che minimizzi il makespan del processo produttivo considerato. Vengono quindi ripresi i paradigmi della programmazione lineare intera e applicati per definire il problema di ottimizzazione considerato. In seguito sono stati definiti degli indici per caratterizzare gli input, ovvero diagramma delle precedenze e durata dei task, e per valutare gli output, cioè makespan e tempo di collaborazione. Tramite le simulazioni e le prove sperimentali è stato validato il modello proposto ed è stata esaminata la relazione tra gli dati in ingresso e in uscita dal problema.

INDICE

ELENCO DELLE FIGURE

ELENCO DELLE TABELLE

1 INTRODUZIONE

Nell'ambito della robotica collaborativa una delle sfide attualmente affrontate consiste nell'assegnazione dei task tra le risorse in gioco. In letteratura esistono varie soluzioni a questo problema che si differenziano tra di loro sulla base dell'obiettivo stesso dell'allocazione: in alcuni casi si cerca di minimizzare i tempi di produzione, in altri di garantire un sistema flessibile e dinamico; alcuni autori invece pongono la propria attenzione sulla sicurezza degli operatori umani e sulla possibilità di diminuire lo sforzo fisico, mentre c'è chi associa il problema di assegnazione a quello di design del layout. Nella maggior parte dei casi le soluzioni proposte si basano su un framework procedurale, mentre l'obiettivo di questa dissertazione è proporre un metodo di risoluzione esatto per fornire una schedulazione dei task che minimizzi il makespan.
Di conseguenza è stato sviluppato un problema di assegnazione basato sulla programmazione lineare intera: a partire dai dati relativi al processo produttivo, ovvero il diagramma delle precedenze, il numero di task e la loro durata, è stato costruito un modello matematico la cui risoluzione permette di minimizzare il tempo di produzione. A seguire sono state effettuate una serie di simulazioni tramite il software Matlab, così da analizzare il modello e visualizzare le relazioni che intercorrono tra gli indici introdotti per caratterizzare gli input e gli output del problema. In fine sono presentate i test sperimentali effettuati per la validazione del modello.
La seguente dissertazione sarà di conseguenza organizzata come segue: nel capitolo 2 si analizzerà lo stato dell'arte delle tecniche di allocazione dei task in ambito collaborativo. Nel capitolo 3 dopo aver introdotto la programmazione lineare di interi saranno proposti due modelli matematici, il primo basato sul ben noto SALBP, il secondo sviluppato appositamente per un sistema uomo-robot collaborativo. Nel capitolo 4 vengono introdotti degli indici che caratterizzano gli input e gli output del problema, i quali poi saranno valutati attraverso le simulazioni presentate nel capitolo 5. In questo verrà inoltre presentato il software utilizzato, oltre che al codice sviluppato; sarà trattato anche il problema della scelta del risolutore adeguato al problema di ottimizzazione presentato. Nell'ultimo capitolo si presenteranno i risultati dei test sperimentali e sarà eseguito un confronto con le prove simulative al fine di valutare il modello proposto.

2 LA ROBOTICA COLLABORATIVA

2.1 INTRODUZIONE

La robotica collaborativa rappresenta uno dei maggiori cambiamenti nell'automazione industriale. Essa combina le potenzialità e le capacità di uomo e robot per automatizzare funzioni ed azioni operative che non era possibile automatizzare prima. Protagonisti di questa rivoluzione sono i cosiddetti "Cobot", termine che nasce dalle parole inglesi "collaborative robot" e la cui definizione è la seguente: *"robot antropomorfi con movimenti su sei assi progettati per rispettare criteri di sicurezza, flessibilità e compattezza e studiati per lavorare a stretto contatto con l'operatore anche senza barriere protettive all'intorno"*, [14]. Il loro impiego è incrementato notevolmente negli ultimi 10 anni a causa delle richieste del mercato, il quale pretende dall'industria moderna la capacità di produrre una larga varietà di prodotti in piccoli lotti. Per questo motivo si è reso necessario sviluppare un sistema di produzione che sia abbastanza flessibile da gestire cambiamenti nei volumi e nei processi di produzione. In un primo momento questi risultati sono stati raggiunti implementando un sistema di assemblaggio manuale (Manual Assembly System o MAS), il quale semplicemente prevede di assegnare all'operatore umano tutti i task da eseguire, realizzando in questo modo il massimo grado di flessibilità. Questa soluzione però presenta evidenti limiti associati agli elevati costi della manodopera, a cui corrisponde inoltre un decremento della produttività stessa. A questo si aggiunge l'eventuale occorrenza dell'errore umano, il quale comporta una serie di conseguenze negative come ad esempio rallentamenti, ritardi e la realizzazione di prodotti non conformi. Per questo motivo l'industria odierna si appoggia principalmente su sistemi di assemblaggio automatici: questi permettono di ovviare ai problemi presenti nei sistemi MAS incrementando notevolmente la produzione, riducendo i costi e migliorando la qualità dei prodotti finali. Però la volontà di passare da una produzione di massa alla possibilità di customizzare il prodotto finale richiede l'ottimizzazione del processo di assemblaggio, così da essere sia competitivi a livello di costo ma soprattutto assicurando allo stesso tempo la variabilità richiesta. Un particolare ramo per lo sviluppo di sistemi di produzione flessibili prevede l'utilizzo di Cobot: il principale vantaggio di questa soluzione deriva dalla combinazione della duttilità tipica dell'uomo con la precisione e l'accuratezza del robot. Questo tipo di sistema quindi oltre a garantire una buona variabilità nella produzione, permette di ridurre i costi e il tempo di ciclo totale, sfruttando una delle più im-

portanti novità introdotte dalla robotica collaborativa: la capacità per un robot di poter condividere lo spazio di lavoro con degli operatori in sicurezza. Questa è la prima differenza che distingue un robot "tradizionale" dal cobot: questo permette l'interazione diretta tra le risorse, superando la classica divisione del lavoro, la quale richiede che i robot siano confinati all'interno di celle di sicurezza, distanti dall'operatore stessi. Oltre a questa fondamentale caratteristica, sono molteplici i fattori che distinguono i robot collaborativi rispetto ai classici robot industriali e che permettono di aumentare in modo significativo la flessibilità nella produzione. Tra questi possiamo ricordare che mentre i robot tradizionali sono tipicamente in grado di svolgere una sola specifica operazione, in modo veloce e ripetitivo, i cobot si prestano ad eseguire una più ampia varietà di task differenti. Questo è dovuto al fatto che la programmazione di un robot tradizionale è di norma più complessa, richiede un tempo maggiore e un alto livello di specializzazione. Nel caso dei cobot al contrario sono disponibili software specifici di facile utilizzo: per esempio essi possono essere basati sulla realtà aumentata, dove le interazioni tra uomo e macchina possono avvenire attraverso tocchi o la voce stessa; oppure attraverso una programmazione per dimostrazione, con la quale è possibile insegnare al robot stesso un determinato percorso nello spazio semplicemente spostandolo da una posizione all'altra. Un altro limite dei robot tradizionali è legato a questioni di sicurezza: i sistemi automatizzati attualmente presenti separano completamente l'operatore dallo spazio di lavoro del robot, attraverso barriere fisiche o barriere virtuali realizzate tramite appositi sensori. Con l'utilizzo dei cobot invece, questi limiti sono completamente eliminati dato che questi hanno diversi meccanismi che impediscono il verificarsi di situazioni pericoloso per l'uomo e l'ambiente. Un minor pericolo è dovuto anche alle dimensioni del robot stesso, il quale di solito è compatto e di conseguenza leggero. Questa caratteristica inoltre permette di poterlo spostare più volte su più mansioni nell'arco della singola giornata lavorativa, il che si traduce con un aumento della flessibilità nella produzione stessa. Al contrario i robot tradizionali non possono né esser spostati né riprogrammati, se non dopo una serie di interventi che richiedono tempi di fermo piuttosto lunghi e di conseguenza costi elevati.

In conclusione, osservando la tabella 1, possiamo constatare che sono molteplici i motivi che suggeriscono come sia preferibile un cobot ad un sistema tradizionale, soprattutto se si intende implementare una linea di produzione e assemblaggio collaborativa che rispecchi i canoni di una industria moderna e flessibile.

Robot industriali tradizionali	Robot industriali collaborativi
Installazione fissa	Flessibilità nella ricollocazione
Task ripetitivi	Frequenti cambiamenti di tasks
Programmazione offline complicata	Programmazione offline e online supportata da diverse modalità di iterazione
Rare interazioni con l'operatore	Frequenti interazioni con l'operatore
Spazio di lavoro separato	Condivisione dello spazio di lavoro
Non può interagire con l'operatore in sicurezza	Può interagire con l'operatore in sicurezza
Profittevole solo con produzioni di media e larga scala	Profittevole anche con produzioni a lotto singolo
Piccolo o grande, molto veloce	Piccolo, leggero, facile da utilizzare e muovere

Tabella 1: Principali differenze tra robot e cobot.

Figura 1: Campi di ricerca nell'ambito della robotica collaborativa [22].

Riprendendo le considerazioni effettuate da Villani et al. in [22], si può affermare che i principali campi di ricerca affrontati nell'ambito della collaborazione tra uomo e robot (detta brevemente HRC, ovvero Human robot collaboration), e rappresentai in Fig. 1 siano i seguenti:

- Sicurezza: è la principale sfida affrontata da ogni sistema collaborativo che preveda un'iterazione tra uomo e robot, in quanto l'obiettivo intrinseco della HRC è eliminare barriere e celle, permettendo un contatto diretto tra le risorse in modo sicuro. La ricerca in questo ambito riguarda principalmente gli standard di sicurezza introdotti dalle normative internazionali (ISO 10218-1, ISO 10218-2, ISO TS 15066), e le modalità con cui può avvenire la collaborazione. Questo argomento verrà trattato più nel dettaglio successivamente durante la spiegazione della prova sperimentale, dove verrà presentato il cobot utilizzato e i relativi standard di sicurezza adottati.

- Interfacce per la programmazione robot: si concentra sullo studio di nuovi approcci per la programmazione del robot che vadano oltre la classica scrittura di codice offline. Infatti, si stanno cercando modalità innovative per sviluppare l'interazione tra uomo robot come ad esempio comandi vocali, e programmazione per dimostrazione, ma anche tecniche avanzate che utilizzino la realtà aumentata. L'obiettivo finale è quello di creare interfacce intuitive e facili da utilizzare, rendendo il lavoro dell'operatore più semplice e accessibile anche ad utenti poco qualificati. Il problema relativo a questo campo di studio risiede nel fatto che le soluzioni sviluppate siano per ora abbastanza limitate in termini di possibili operazioni e la maggior parte di questi scenari siano stati validati soltanto a livello sperimentale nei laboratori di ricerca.

- Metodi di progettazione: comprendono lo studio delle leggi di controllo, la scelta di sensori e dei task da eseguire per garantire che l'operatore lavori in sicurezza fianco a fianco del robot, condividendo attivamente lo spazio di lavoro e i compiti da portare a termine.

Come anticipato la corrente trattazione ricade nell'ultimo campo di ricerca analizzato e in particolare si colloca nel problema di assegnazione ottima dei task tra le risorse in gioco, ovvero robot e operatore. In modo da contestualizzare il lavoro effettuato, in questo capitolo verranno riproposte alcune soluzioni di task allocation studiate in letteratura. Un percorso simile è già stato svolto da Tsarouchi et al. in [20], dove gli autori oltre ad aver analizzato i vari metodi di interazione tra robot e uomo (HRI, human robot interaction) offrendo una

precisa descrizione delle varie tecnologie utilizzate ad oggi per realizzarlo, hanno anche riportato alcune soluzioni adottate per risolvere il problema dell'assegnazione dei task. A differenza di quest'ultimo però in questo capitolo viene dato maggior risalto all'analisi dei metodi utilizzati per la risoluzione di tale problema, analizzando gli obiettivi dei vari autori e le soluzioni a cui sono pervenuti. Un riepilogo sui risultati di tale studio è rappresentato in tabella 2.
Si può innanzi tutto notare che nella maggior parte delle trattazioni in letteratura gli autori hanno proposto un metodo per la risoluzione del problema del task allocation basato su un framework: la soluzione viene quindi trovata seguendo un modello operativo, formato da diversi step. Questi a loro volta possono richiedere sia di effettuare operazioni matematiche, come per esempio la valutazione di un determinato indice, ma anche di prendere decisioni oggettive o soggettive sulla base dei dati a disposizione di chi svolge l'analisi. Johannsmeier et al. in [7] per esempio ha proposto un framework che comprende tre livelli:

- Nel primo livello, detto anche team level, il processo di assemblaggio viene pianificato offline a partire dalle informazioni sui task da svolgere, sulle parti da assemblare, sulle risorse disponibili. Inoltre, viene valutato un grafico AND/OR caratteristico del prodotto, il quale suddivide i task e li ordina in modo gerarchico considerando in questo modo implicitamente la possibilità di poter parallelizzare le attività. L'allocazione ottima viene ricercata attraverso un algoritmo di ricerca, il quale minimizza delle funzioni di costo diverse in base al fatto che la risorsa considerata sia un operatore o un robot. Queste tengono conto attraverso alcuni coefficienti di diversi parametri, quali:
  - il tempo di esecuzione dei task,
  - il costo della risorsa, come ad esempio il consumo di energia elettrica nel caso del robot,
  - fattori di rischio per l'operatore, come l'ampiezza e l'ergonomia del carico,
  - livello di attenzione ed esperienza dell'operatore.
- Nel secondo livello la schedulazione ottenuta al passo precedente viene modificata tenendo in considerazione il comportamento del robot. In particolare, si controlla che questo sia effettivamente capace di portare a termine l'esecuzione dei task assegnati e che nel fare questo, l'ambiente di lavoro rimanga comunque sicuro per l'operatore. In altre parole, viene verificato che non vi siano i presupposti per eventuali collisioni, le quali oltre a influire negativamente nell'aspetto relativo alla sicurezza sono anche sinonimo di rallentamenti e interruzioni.

- Il terzo livello del framework è quello direttamente responsabile del planning delle traiettorie del robot. In questo caso la pianificazione avviene online: attraverso l'utilizzo di sensori, sistemi di visione e tecniche di machine learning vengono modificate in real time i movimenti e le traiettorie del robot in modo da evitare situazioni di pericolo e far fronte a eventi imprevisti.

Come si può notare osservando i vari step proposti, l'obiettivo dell'autore era quello di offrire attraverso un framework operativo non solo un metodo per l'allocazione dei task ma anche, con l'introduzione del secondo e terzo livello, di suggerire una serie di considerazioni che possono esser effettuate in modo da garantire una collaborazione sicura e flessibile.
Un'altra soluzione basata su un framework è stata proposta da Tsarouchi et al. in [21], con una sostanziale differenza rispetto al caso appena discusso. Infatti, mentre nella trattazione precedente i tre livelli introdotti andavano a identificare un'unica soluzione di per sé già ottima, con il seguente metodo l'algoritmo decisionale presentato dall'autore permette di trovare una serie di soluzioni, le quali dovranno essere poi valute a posteriori con uno dei criteri presenti in letteratura. In particolare, i vari step che devono essere svolti sono:

- valutare la capacità della risorsa, sia essa l'operatore o il robot, di svolgere un determinato task;
- nel caso la risorsa possa eseguire il task, bisogna valutare se essa è libera o occupata;
- dopo aver fatto le precedenti valutazioni per tutte le risorse, il task deve essere assegnato all'operatore o robot che lo esegue nel minor tempo possibile;

Queste valutazioni devono essere effettuate per ogni task, partendo da quelli al livello più basso del modello gerarchico che rappresenta il processo di assemblaggio del prodotto preso in considerazione. Come anticipato avremo una serie di possibili task allocation, le quali nel caso dell'articolo analizzato sono state valutate attraverso due criteri: il primo tiene conto dell'utilizzo medio di una risorsa (ARU Average resource utilisation), il secondo del makespan medio.
Un ulteriore framework da utilizzare per raggiungere una task allocation in ambito collaborativo è stato proposto da Tan et al. in [19]. A differenza di quelli precedentemente esposti in questo caso l'autore propone un metodo che risolva il problema non solo valutando la produttività del processo di assemblaggio, ma studiando altri campi di interesse come la qualità, la fatica dell'operatore e la sua sicurezza. Inoltre, quest'analisi viene effettuata per un particolare tipo di interazione uomo robot diversa dalle precedenti. Infatti, come raccontato da Tsarouchi et al. in [20] la HRI può essere classificata in tre principali categorie:

- task e workspace condivisi
- task e workspace comuni
- task comuni e workspace separati

Mentre nelle trattazioni [7] e [21] rispettivamente di Johannsmeier e Tsarouchi, il robot e l'operatore nei momenti di collaborazione avevano compiti e spazio di lavoro comuni, nel caso studiato da Tan et al. viene presa in considerazione la situazione in cui essi siano condivisi, ovvero quando cobot e uomo lavorano contemporaneamente allo stesso task nello stesso workspace. Il framework proposto in questo articolo ha quindi come obiettivo principale quello di stabilire se ogni task dove essere eseguito da una singola risorsa o da entrambe in modo collaborativo. Il procedimento prevede due fasi: la prima di tipo qualitativo in cui si decide, sulla base delle informazioni disponibili sia in termini di tempo che di qualità di esecuzione, se l'azione deve essere compiuta da una sola delle due risorse. Nella seconda parte i task non ancora allocati vengono valutati quantitativamente attraverso degli indici relativi ai parametri citati inizialmente: produttività, qualità, fatica dell'operatore e sicurezza. In base a questi si può finalmente capire se sia meglio assegnare tale compito ad una risorsa o all'altra o magari ad entrambe. L'autore attraverso la prova sperimentale effettuata ha infine dimostrato che utilizzando il metodo collaborativo da esso proposto vi è un netto miglioramento delle prestazioni rispetto al caso in cui tutti i task vengano eseguiti dal solo operatore umano.
In letteratura troviamo inoltre ricercatori come Michalos et al. e Fechter et al. che rispettivamente in [10] e [4] consigliano di risolvere il problema del task allocation andando a considerare parallelamente la necessità di progettare un layout adeguato a un sistema collaborativo. In particolare anche Michalos et al. basano la loro trattazione su un'analisi di più criteri, confrontando le diverse soluzioni in termini di produttività, ergonomia e qualità del processo con i possibili layout che permettono di ottenere un uso efficiente del piano di lavoro.
Tra le altre dissertazioni presenti in letteratura troviamo anche quella di Ranz et al., i quali in [13] cercano di determinare un scheduling ottimale tenendo in considerazione la capacità (intesa come abilità) delle risorse, ponendo di conseguenza maggior importanza sulla qualità finale del prodotto. Anche questo metodo è formato da una serie di step esecutivi:

- inizialmente a partire dall'analisi delle caratteristiche del prodotto e del processo produttivo vengono individuati i diversi task che devono essere eseguiti e il loro ordine di esecuzione.
- In seguito vengono individuati i task definiti "invariabili", ovvero quelli che possono essere eseguiti da una sola delle due risorse in modo qualitativamente ottimale. Per far questo gli

autori hanno eseguito una approfondita ricerca in letteratura e realizzato un primo catalogo, il quale contiene dodici criteri utili per stabilire se l'operatore umano ha la capacità di eseguire quel determinato compito, o se questo debba per forza esser assegnato al robot.

- i task che non appartengono a questa categoria sono detti "variabili" e possono essere assegnati ad entrambe le risorse. Gli autori per rendere anche in questo caso obiettiva la scelta della risorsa ottimale hanno introdotto un secondo catalogo. Questo lista una serie di oggetti tipici negli assemblaggi di elementi di dimensioni piccole e compatte. Per ognuno di essi è presente un indice di capacità, che indica se è preferibile che tale oggetto sia assemblato da un operatore umano o un robot, il quale è stato precedentemente calcolato a partire da considerazioni sperimentali relative ai tempi di esecuzione, qualità del processo di costruzione e presenza di investimenti addizionali.

L'obiettivo di questo articolo è quindi quello di fornire un approccio "esatto" e oggettivo sulla base di osservazioni empiriche per l'allocazione di quelle azioni che non possono essere assegnate a priori ad una risorsa o all'altra. I risultati ottenuti confermano l'esistenza di alcuni task che date le capacità della risorsa devono essere preferibilmente assegnati ad un robot, altri che richiedono maggior destrezza, abilità e di conseguenza devono essere assegnati all'operatore, e infine task che richiedono un criterio che valuti come effettuare l'assegnazione.
Takata et al. in [17] si sono invece posti il problema di ricercare una task allocation che prenda in considerazione l'eventualità di cambiamenti di scenario nel modello del prodotto e nei volumi della produzione. In realtà in questo caso non viene proposto una vera e proprio metodo per determinare l'assegnazione dei task in ambito collaborativo, ma invece viene presentato ed esaminato un procedimento per valutare a posteriori quale scheduling utilizzato sia migliore nel caso di frequenti cambiamenti nella produzione. Il tema in questo articolo è di fatto l'influenza della variabilità in un processo produttivo e l'obiettivo è valutare la flessibilità del metodo di assegnazione dei task. Per questo motivo le considerazioni in questo articolo seppur non completamente affini all'oggetto trattato in questa tesi vengono brevemente riportate in quanto l'argomento rimane comunque centrale nell'abito della robotica collaborativa. Per ottenere il risultato desiderato, l'autore suppone innanzitutto di aver un numero preciso di scenari noti e possibili e consiglia di procedere come segue: per prima cosa scegliere uno degli n scenari iniziali e calcolare la task allocation che minimizzi il makespan. In seguito con il scheduling trovato l'autore suggerisce di determinare il tempo di produzione per tutti gli altri possibili scenari che si possono verificare e moltiplicare il risultato ottenuto con la probabilità che tale cambiamento si verifichi. Il

passo finale consiste di calcolare il costo totale di produzione sommando i valori trovati precedentemente: i risultati ottenuti da questa esperienza dimostrano che la task allocation che porta ad avere il risultato minimo è anche quella che più è adatta ad essere utilizzata per minimizzare il makespan nelle situazioni in cui i cambiamenti di scenario sono frequenti.
Chen et al. in [2], a differenza dei casi presentati fino ad ora, propone un modello matematico che descriva il problema del task balancing e la cui risoluzione porti ad ottenere una schedulazione che minimizza il tempo totale di assemblaggio dell'oggetto in questione. La particolarità di questa trattazione risiede nella modalità con cui sono definiti i vincoli progettuali tra i diversi task: l'autore infatti assume che l'intero processo di assemblaggio in ambito collaborativo possa esser suddiviso in due parti: assemblaggio parallelo o sequenziale. Nel primo caso operatore e robot lavorano contemporaneamente condividendo lo spazio di lavoro e anche il tempo; nell'assemblaggio sequenziale invece, operatore e robot eseguono i task uno dopo l'altro. Questa assunzione ha effetto sull'organizzazione dei task stessi. Essi sono suddivisi in gruppi: possono essere eseguiti in parallelo solo se all'interno di uno stesso gruppo, mentre questi ultimi devono avere tra di loro un rapporto sequenziale. Tale suddivisione risulta molto importante perché in questo modo l'ottimizzazione del modello matematico proposto dall'autore, il quale si presenta come un problema di programmazione di numeri interi (ILP Integer Linear Programmation), viene suddiviso in un numero di sotto problemi pari al numero di gruppi individuati precedentemente, portando ad una significativa semplificazione nella ricerca della soluzione. Nelle prove sperimentali effettuate, gli autori hanno valutato diversi scenari in cui a variare erano il numero di robot che collaboravano insieme all'operatore umano; i risultati elaborati pongono particolare attenzione alle prestazioni dell'algoritmo genetico implementato, e da questi si osserva come un problema di natura complessa possa esser risolto in tempi relativamente corti utilizzando il metodo proposto.
Come Chen et al. anche Bogner et al. in [1] sviluppa un modello matematico da ottimizzare per la risoluzione del problema del task balancing, introducendo una formulazione basata sulla programmazione lineare di interi. A differenza del caso precedente però i vincoli progettuali sono rappresentati attraverso un diagramma delle precedenze, portando di conseguenza ad una sostanziale variazione del modello stesso. Anche in questa trattazione l'autore ha l'obbiettivo di valutare le prestazioni del processo di ottimizzazione: i risultati ottenuti dimostrano che, nel caso di problemi con una grande quantità di dati, la risoluzione del ILP sia molto più efficace a livello di tempo di elaborazione utilizzando un approccio euristico. Questo criterio però può generare soluzioni che non coincidono esattamente con l'ottimo, eventualità che al contrario non si presenta nel caso in cui vengano

utilizzati metodi di risoluzione esatti.
In [12] gli autori Pearce et al. hanno l'obiettivo di trovare un'allocazione ottima che minimizzi contemporaneamente sia il tempo di esecuzione dell'insieme di task che compongono l'assemblaggio, sia lo sforzo fisico dell'operatore. Quest'ultimo è stato valutato attraverso un indice denominato SI (Strain Index), originariamente introdotto da Moore e Garg in [16], il quale va a valutare quanto sia rischioso per l'operatore una determinata azione. In questo caso il problema di task allocation è stato formulato utilizzando una variante del ILP, ovvero attraverso una programmazione lineare di interni mista (MILP, mixed integer linear Programmation) la quale verrà introdotta nel capitolo seguente, in quanto è un argomento centrale ai fini dell'analisi effettuata in questa tesi. L'obiettivo finale dello scheduler è quello di allocare i task, organizzati attraverso un modello gerarchico, ad una delle risorse e determinare la sequenza che minimizzi elementi come il tempo di produzione, lo stress fisico dell'operatore o entrambi.
In conclusione, possiamo affermare che il problema del task allocation è una delle sfide di maggior importanza nella robotica collaborativa e come tale è stato ampiamente trattato in letteratura. Come si è cercato di evidenziare da questa ricerca, nonostante la maggior parte delle volte il problema sia stato affrontato con la definizione di un framework, le soluzioni proposte sono tra di loro differenti e varie. In particolare diversi sono gli obiettivi che i vari autori si sono assegnati: in alcuni casi come in [21],[12] l'analisi si è concentrata sul trovare la task allocation che ottimizzi la produttività, in altri come in [4] e [10] gli autori hanno tenuto in considerazione anche il problema di design del layout. In [7] e [19] i metodi proposti sono stati sviluppati per garantire la presenza di un sistema flessibile e dinamico, mentre in [13], [12] e [17] il problema del task allocation è stato valutato in relazione ad altri parametri come la capacità delle risorse, lo forzo fisico dell'operatore e la variabilità dei scenari di produzione. Per quanto riguarda il lavoro che verrà presentato in questa dissertazione le trattazioni più interessanti e che verranno in seguito richiamate sono quelle di Bogner et al. e Chen et al. ovvero [1] e [2], in quanto come sarà spiegato nel prossimo capitolo, il problema del task balancing verrà affrontato in modo simile attraverso la definizione di un modello matematico ILP, la cui ottimizzazione porti ad una schedulazione che minimizzi il makespan.

Tabella 2: Stato dell'arte.

References	Authors	Year	Task Allocation			Input data				Method			Result
			Optimization of productivity and cycling time	Desing workspace considering scheduling	Dynamic allocation and flexibility	Task data, graph ,resources	CAD Data	Product Charateristic	Other/none	Framework	Genetic Algorhitm	Heuristic	
[1]	Bogner et al.	2018	X			X						X	Developed a mathematicam model for task allocation and resolved with Heuristic
[2]	Chen et al.	2014	X			X					X		Developed a mathematicam model for task allocation and resolved with GA
[4]	Fechter et al.	2018		X		X	X					X	Developed a planning system that consideres layout
[7]	Johannsmeier et al.	2017			X	X				X			Developed a planning system that results dynamic and flexible
[10]	Michalos et al.	2018		X			X	X		X			Developed a planning system that consideres layout
[11]	Nikolakis et al.	2018			X	X				X			Reduces time spended on a specific job
[12]	Pearce et al.	2018	X			X					X		Reduce makespan considering phisical strain
[13]	Ranz et al.	2017			X	X		X		X			Developed a method for task allocation based on resources capability
[17]	Takata et al.	2011			X				X	X			Developed a method to find best task allocation in manufacturing process with multiple scenarios
[19]	Tan et al.	2010			X	X			X	X			Shorter assembly time using collaboration
[21]	Tsarouchi et al.	2016	X			X				X			Developed a planning system that minimize makespan and avarage resource utilization

3

IL MODELLO MATEMATICO

In questo capitolo verrà presentato il modello matematico implementato per affrontare il problema del task balancing in ambito collaborativo. Nella prima sezione verrà innanzitutto analizzata la formulazione teorica da cui deriva. In seguito, verrà introdotto il problema del task balancing utilizzato nelle linee di assemblaggio sequenziali. Questo sarà il punto di partenza da cui verrà derivato il modello di allocazione proposto e servirà in seguito anche come termine di paragone per valutare la bontà dell'assegnazione.

3.1 PROGRAMMAZIONE LINEARE INTERA

La Programmazione lineare intera (PLI o ILP, ovvero Integer Linear Programming) tratta il problema della minimizzazione (o massimizzazione) di una funzione obiettivo di tipo lineare, la quale è definita da più variabili soggette a vincoli di uguaglianza o disuguaglianza lineari. Quest'ultime possono essere tutte di tipo intero (PLI pura) o solo in parte (PLI mista). Un problema di programmazione lineare in forma canonica è espresso nel modo seguente:

$$\begin{aligned} &\min c^T x \\ &Ax \geqslant b \\ &x \in \{0,1\}. \end{aligned} \tag{1}$$

Data la generalità del modello, possono essere rappresentati e risolti un'elevata varietà di problemi reali; in particolare la programmazione lineare intera viene utilizzata in quelle applicazioni dove è necessario scegliere tra un numero finito di alternative e in cui le risorse in gioco siano indivisibili. Alcuni esempi comprendono problemi operativi come il sequenziamento delle attività produttive, problemi di pianificazione come la gestione finanziaria di un portafoglio o problemi di progettazione quali il dimensionamento di un sistema automatico di produzione. Una particolare variante del PLI è un problema di ottimizzazione combinatoria in cui le variabili in gioco sono di tipo binario $\{0,1\}$: se ad esempio si vuole modellare il caso in cui un determinato evento possa verificarsi oppure no, la nature del problema suggerisce di utilizzare una variabile x che può assumere solo valori 0,1. Generalmente, si porrà $x = 1$ se l'evento si verifica e $x = 0$ altrimenti. Inoltre, è possibile formulare una serie di disequazioni che colleghino le variabili intere in modo da esprimere una serie di relazioni tra gli eventi del problema considerato.
Un classico esempio in letteratura è quello del *Kanpsack* binario: si

supponga di avere n progetti; per ognuno di essi è possibile definire un costo pari a a_j, e un valore pari a c_j. Ogni progetto deve essere portato a termine e non è ammissibile una realizzazione parziale; inoltre esiste un budget b disponibile per realizzare tali progetti. La richiesta di questo problema consiste nel determinare un sottoinsieme di progetti che possono essere realizzati senza superare il budget fissato estraendo il massimo valore da essi. È possibile modellare il problema appena introdotto con una variabile binaria:

$$x_j = \begin{cases} 1 \text{ se il progetto } j \text{ è selezionato} \\ 0 \text{ altrimenti,} \end{cases} \tag{2}$$

la quale è sottoposta al vincolo $\sum_{j=1}^{n} c_j x_j \leqslant b$. Il problema può essere quindi descritto come segue:

$$\begin{aligned} \max \quad & \textstyle\sum_{j=1}^{n} c_j x_j \\ & \textstyle\sum_{j=1}^{n} a_j x_j \leqslant b \\ x_j \in \{0,1\} \quad & j = 1, \ldots, n. \end{aligned} \tag{3}$$

Si osserva che in questo particolare esempio l'evento sintetizzato è l'attuazione o meno del singolo progetto il quale è sottoposto da un solo vincolo, anche se in realtà ne esistono molteplici e di natura diversa.
Un altro esempio in letteratura è il problema di assegnamento, il quale come il problema del *Knapsack* riprende alcuni concetti che saranno considerati anche per la formulazione del modello di bilanciamento. Esso cerca di determinare il modo ottimale per assegnare lavori a persone, o con una concezione più generica, assegnare mezzi (ad esempio cobot e operatore) ad attività. Supponiamo ad esempio che n persone debbano svolgere altrettanti lavori, in modo che esattamente una persona svolga un solo lavoro e di conseguenze che tutti siano occupati con un'unica attività. Inoltre si consideri che ad ognuno di essi è associata una variabile c_{ij} la quale rappresenta il costo necessario affinché la i-esima persona svolga il j-esimo lavoro. L'obiettivo del problema è assegnare ad ogni persona un lavoro minimizzando il costo totale. In questo caso la variabile binaria utilizzata dipende da due fattori, persone e lavori, e può essere scritta come segue:

$$x_{ij} = \begin{cases} 1 \text{ se la persona } i \text{ è assegnata al lavoro } j \\ 0 \text{ altrimenti.} \end{cases} \tag{4}$$

La formulazione completa è data da:

$$\begin{aligned} \min \quad & \textstyle\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\ & \textstyle\sum_{i=1}^{n} x_{ij} = 1 \\ & \textstyle\sum_{j=1}^{n} x_{ij} = 1 \\ x_{i,j} \in \{0,1\} \quad & i, j = 1, \ldots, n. \end{aligned} \tag{5}$$

Possiamo quindi valutare la presenza di una nuova tipologia di vincolo, detto *vincolo di assegnamento*, la cui definizione è fondamentale per il modello che verrà proposto.
Infine, nella definizione di un problema di programmazione lineare di interi può essere necessario utilizzare variabili booleane per modellare condizioni logiche. Un esempio classico sono le cosiddette *variabili indicatrici*, di tipo binario e che consentono di formulare relazioni di dipendenza tra i dati del problema. Si consideri, per esempio, il caso di una variabile x di cui si conosce il limite superiore M; se si introduce il vincolo:

$$x - My \leqslant 0 \tag{6}$$

la variabile $y \in \{0, 1\}$ è costretta ad assumere valore 1 se $x > 0$, e 0 se $x = 0$. Tale relazione può esser sfruttata per imporre un *vincolo di mutua esclusione*: si consideri il caso di due variabili x_1 e x_2, e si voglia attuare il seguente legame logico: se $x_1 > 0 \quad \Rightarrow \quad x_2 = 0$ e viceversa, è possibile scrivere i seguenti vincoli:

$$\begin{aligned} x_1 - My &\leqslant 0 \\ x_2 - M(1-y) &\leqslant 0, \end{aligned} \tag{7}$$

dove M è un numero positivo maggiore del più grande valore che possono assumere le variabili.
Si può concludere affermando che utilizzando le tipologie di formulazioni presentate in questa sezione, è possibile riprodurre una grande varietà di problemi reali tra cui anche il problema di allocazione dei task in un sistema collaborativo che verrà studiato.

3.2 SALBP - IL SIMPLE ASSEMBLY LINE BALANCING PROBLEM

Il simple assembly line balancing problem, o più brevemente SALBP è stato ampiamente studiato nelle ultime decadi, in quanto la sua risoluzione permette di ottenere una chiara schedulazione dei task all'interno di una linea di assemblaggio cadenzata. Le principali caratteristiche di questo modello sono le seguenti:

- produzione di una sola tipologia di prodotto attraverso n task estrapolati dal processo produttivo;
- layout della linea di assemblaggio seriale;
- produzione cadenzata con un determinato tempo di ciclo;
- tempi dei task deterministici o interi;
- nessun limite di assegnazione se non le relazioni di precedenza dovuti a vincoli costruttivi e rappresentati attraverso il diagramma delle precedenze.

Esistono in letteratura varie versioni di questo problema le quali si differenziano sulla base dei diversi obiettivi del problema stesso. L'attenzione di questa analisi si soffermerà in una particolare variante del problema di fattibilità: l'obiettivo sarà prima quello di verificare se data una determinata linea con m risorse esiste una soluzione che mi permette di eseguire l'assemblaggio rispettando un determinato tempo di ciclo c, per poi cercare di ottimizzare il tempo di ciclo stesso, ovvero massimizzando la produzione. Prima di presentare la formulazione matematica vera e propria, di seguito vengono riportate quelle che sono le notazioni utilizzate:

n	numero di task
j	indice del task
m	numero di risorse
k	indice della risorsa
G	grafico delle precedenze
c	tempo di ciclo
q	makespan
t_{jk}	tempo di esecuzione del task j per la risorsa k
P_j	insieme degli immediati predecessori del task j-esimo
S_j	l'insieme degli immediati successori del task j-esimo

I dati in ingresso al problema sono ovviamente il numero di task n e i relativi tempi di esecuzione t_{jk}, il numero di risorse m e il grafico delle precedenze G. Da quest'ultimo è possibile ricavare l'insieme di predecessori e successori immediati di ogni task, P_j e S_j che risultano fondamentali per la formulazione del modello matematico. Per quanto riguarda il tempo di ciclo c, esso è la variabile che il problema vuole minimizzare e di conseguenza può esser visto come la vera e propria funzione obiettivo del problema di ottimizzazione implementato.
Esistono diversi metodi per formulare matematicamente questo problema attraverso un modello di programmazione lineare di interi, introdotto nella sezione precedente. Quella utilizzata in questa dissertazione deriva dalla formulazione di Patterson e Albracht [15], poiché tra quelle proposte in letteratura contiene il minimo numero di vincoli e variabili.
Il modello di bilanciamento è il seguente:

$$x_{jk} = \begin{cases} 1 \text{ se il task } j \text{ è assegnato alla stazione } k \\ 0 \text{ altrimenti,} \end{cases} \tag{8}$$

con

$$x_{jk} = \{0, 1\}, \tag{9}$$

dove $x_j k$ è una variabile decisionale binaria e rappresenta completamente la soluzione, o schedulazione. Questa è soggetta ai seguenti vincoli e condizioni:

$$\sum_{k=1}^{m} x_j k = 1, \tag{10}$$

$$\sum_{k=1}^{m} k \cdot x_{jk} \leqslant \sum_{k=1}^{m} k \cdot x_{ik}, \qquad \text{con } (i,j)|i \in P_j, \tag{11}$$

$$\sum_{j=1}^{n} t_{jk} \cdot x_{jk} \leqslant c. \tag{12}$$

Ovviamente l'obiettivo sarà quello di minimizzare c

$$\min c. \tag{13}$$

Andiamo ora ad analizzare le condizioni introdotte:

- Il primo vincolo descritto in (10) è detto *vincolo di assegnazione* e insieme a (9) garantisce che ogni task sia affidato esattamente a una sola risorsa;
- la disequazione (11) è detta *vincolo di precedenza* e garantisce che un task sia eseguito dopo tutti i suoi predecessori;
- l'ultimo vincolo (12) garantisce che per ogni risorsa sia rispettato il tempo di ciclo minimo.

Questa formulazione come anticipato viene utilizzata nel caso di linee di assemblaggio sequenziali, ovvero quando le risorse m sono poste in serie in stazioni diverse, non condividendo lo spazio di lavoro. È quindi una soluzione ovviamente inadatta per la allocazione dei task in un sistema collaborativo. Però dato che è un sistema ancora ampiamente diffuso vista la facilità di implementazione e utilizzo, potrebbe essere comunque adattato per l'allocazione dei task in un sistema collaborativo con una semplice modifica del layout come in Fig. 2: in questo modo il cobot e l'operatore hanno la possibilità di lavorare contemporaneamente al processo di assemblaggio. Come vedremo, il risultato ottenuto non è dei migliori per quanto riguarda la collaborazione. Infatti, l'obbiettivo di questo problema è sempre e comunque minimizzare il tempo di ciclo e di conseguenza bilanciare il tempo di esecuzione delle due risorse. Invece il modello proposto nella sezione successiva, attraverso una serie di modifiche permette di minimizzare il makespan, quindi di massimizzare la produzione.

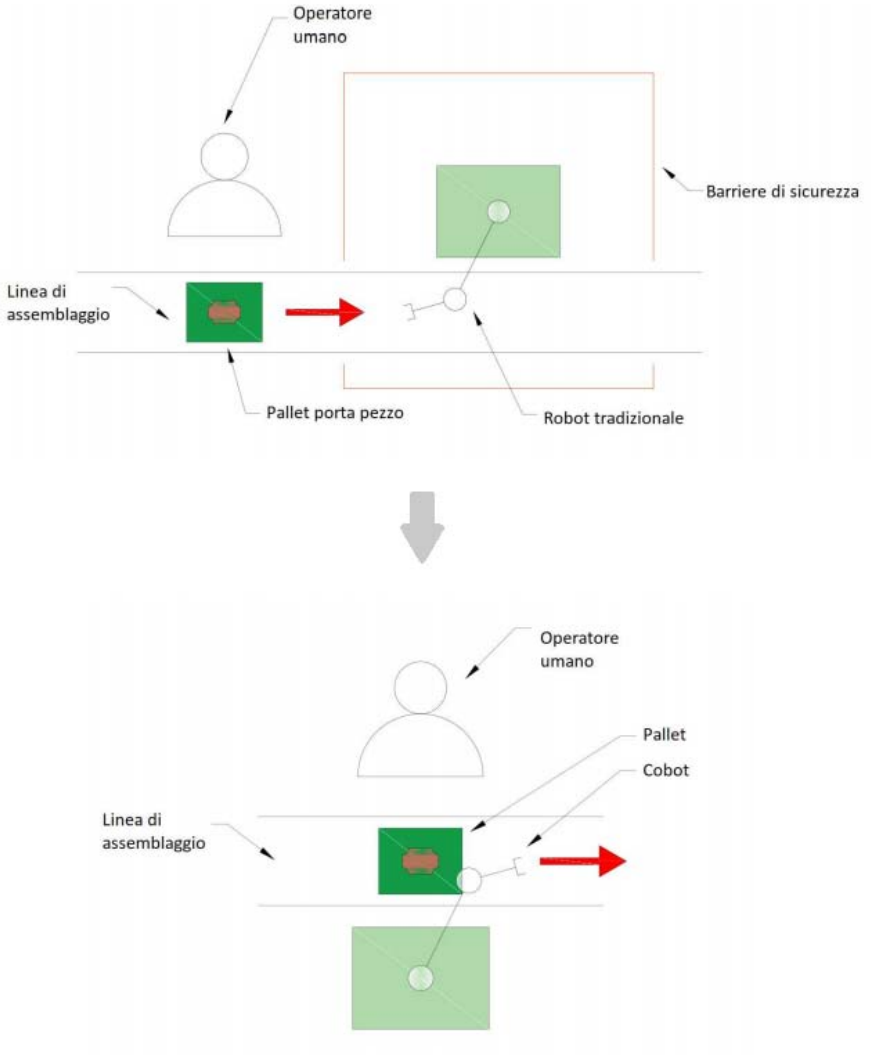


Figura 2: Layout tradizionale e layout collaborativo [3].

3.3 MODELLO DI BILANCIAMENTO IN AMBITO COLLABORATIVO

In questa sezione verrà spiegato nel dettaglio il modello di balancing utilizzato per l'allocazione ottima dei task nel caso collaborativo. Nella prima parte sono spiegate le caratteristiche, i dati e le assunzioni fatte per la definizione di tale problema. A seguire vengono riportati i vincoli e la funzione obiettivo così da completare la formulazione matematica.

3.3.1 *Descrizione del problema di allocazione*

I dati e le caratteristiche del problema di balancing preso in esame possono essere riassunti come segue:

- il layout preso in considerazione è formato da una sola stazione dove lavorano un operatore umano e un cobot;
- le risorse condividono lo spazio di lavoro e svolgono task comuni;
- produzione di una sola tipologia di prodotto attraverso n task estrapolati dal processo produttivo;
- i task sono indivisibili, il loro tempo è fisso e dipende dalla risorsa che lo esegue;
- nessun limite di assegnazione, se non le relazioni di precedenza dovuti a vincoli costruttivi e rappresentati attraverso il diagramma delle precedenze;
- l'obiettivo dell'assegnazione è minimizzare il makespan, ovvero il tempo che intercorre dall'inizio dell'esecuzione del primo task e la fine dell'ultimo.

Osserviamo che rispetto al problema introdotto precedentemente le differenze sostanziali che troviamo riguardano:

- come rappresentato in Fig.2 il layout in questo caso è studiato in modo da garantire una collaborazione tra robot e operatore;
- la funzione obiettivo che in questo caso diventa il makespan, poiché non ha più senso parlare di tempo di ciclo essendo presente un'unica stazione.

Date queste modifiche che permettono il passaggio da un assemblaggio classico a collaborativo, ne consegue che il modello matematico dovrà essere profondamente rivisitato rispetto a quello presentato nel capito precedente.

3.3.2 *Notazioni utilizzate*

Di seguito sono elencate le notazioni utilizzate per la formulazione del modello matematico:

n	numero di task
j	indice del task
V	insieme contenete tutti i task
m	numero di risorse
k	indice della risorsa
K	insieme delle risorse, ovvero $K = \{OP, R\}$ dove OP è l'operatore e R il robot
G	grafico delle precedenze
t_{jk}	tempo di esecuzione del task j per la risorsa k
P_j	insieme degli immediati predecessori del task j-esimo
S_j	l'insieme degli immediati successori del task j-esimo
x_{jkt}	variabile binaria da allocare, la quale indica a che istante t il task j viene iniziato dalla risorsa k
t	istante temporale in secondi
T	orizzonte temporale in secondi
y_{ji}	variabile binaria che indica se j è stato eseguito dopo di i
C	makespan in secondi

3.3.3 *Il modello matematico*

Il modello matematico è formato dalle seguenti variabili decisionali binarie:

$$x_{jkt} = \begin{cases} 1 & \text{se task j viene iniziato dalla risorsa k all'istante t} \\ 0 & \text{altrimenti,} \end{cases}$$

$$\forall j \in V, \forall k \in K, \forall t \in [0, T] \tag{14}$$

$$y_{ji} = \begin{cases} 1 & \text{se j è eseguito prima del task i} \\ 0 & \text{altrimenti,} \end{cases} \tag{15}$$

dove la variabile x_{jkt} è quella che indicherà la schedulazione risolto il problema di ottimizzazione. Si osserva che rispetto al caso analizzato nella sezione precedente è stata introdotta la variabile tempo. Questa è la principale causa delle modifiche effettuate, ma il suo utilizzo si è reso necessari per definire in modo corretto il makespan (21).

Per quanto riguarda y_{ji},descritta in (15), anch'essa è una variante del problema (*variabile indicatrice*), e deve sempre esser assegnata durante il processo di risoluzione. Il suo utilizzo è direttamente legato alla condizione descritta in (18), ovvero uno dei vincoli seguenti:

- *vincolo di assegnazione o occorrenza*:

$$\sum_{t\in T}\sum_{k\in K} x_{jkt} = 1, \quad \forall j \in V, \tag{16}$$

  il quale serve per garantire che tutti i task j inizino esattamente ad un istante $t \in T$ e siano eseguiti da una sola delle due risorse. Esso è la naturale evoluzione di quello introdotto con (10) per il SALBP e modificato tenendo conto del parametro tempo.

- *Vincolo di precedenza*:

$$\sum_{t\in T}\sum_{k\in K}(t + t_{ik})\cdot x_{jkt} \leqslant \sum_{t\in T}\sum_{k\in K} t\cdot x_{jkt}, \quad \forall j \in V, \quad \forall i \in P_j, \tag{17}$$

  il quale garantisce il rispetto del vincolo tecnologico. Anche in questo caso la precedenza deve tener conto anche della variabile tempo: il secondo membro di tale disequazione, dato il vincolo di occorrenza permette di individuare un istante temporale ben preciso in cui il task j avrà inizio. Il vincolo impone che tale t sia maggiore dell'istante finale (ovvero $(t + t_{ik})$) di tutti i task i direttamente precedenti j.

- *Vincolo di precedenza nei task paralleli*:

$$\sum_{j\in V}\sum_{k\in K} x_{jkt} \leqslant 1 \qquad \forall k \in K,$$

$$\sum_{t\in T}\sum_{k\in K}(t + t_{jk})\cdot x_{jkt} - M\cdot y_{ji} \leqslant \sum_{t\in T}\sum_{k\in K} t\cdot x_{ikt}, \quad \forall j, i \in V, \quad i \neq j,$$

$$\sum_{t\in T}\sum_{k\in K}(t + t_{jk})\cdot x_{jkt} - M\cdot(1 - y_{ji}) \leqslant \sum_{t\in T}\sum_{k\in K} t\cdot x_{ikt}, \quad \forall j, i \in V, \quad i \neq j. \tag{18}$$

  Queste disequazioni sono necessari perché il vincolo proposto precedentemente in (17), garantisce il rispetto della precedenza soltanto nei casi in cui vi sia un collegamento diretto tra i task in questione. Infatti osservando la Fig. 3, rappresentante un generico diagramma delle precedenze, si vede che dati un task j i rimanenti $n-1$ task possono essere suddivisi in:

  – P_j^*: insieme di tutti i task che devono essere eseguiti prima del task j, ovvero l'insieme di tutti i predecessori. In Fig.3 sono evidenziati con il rettangolo rosso ;

- S_j^*: insieme di tutti i task che devono essere eseguiti dopo il task j ,ovvero l'insieme di tutti i successori. In Fig.3 sono evidenziati con il rettangolo verde;
- $D_j = P_j^* + S_j^*$: insieme di tutti I task che hanno un collegamento diretto con il task j.

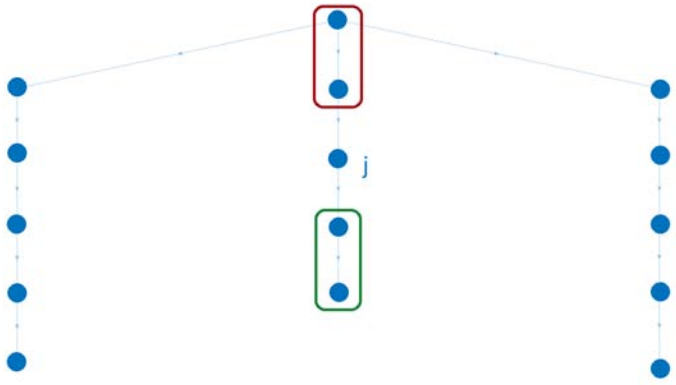


Figura 3: Esempio di P_j^* e S_j^* in un generico grafico delle precedenze.

Se questo vincolo non viene utilizzato può succedere che l'algoritmo di risoluzione assegni un task j ad una risorsa anche se questa in realtà è occupata: supponiamo per esempio di avere due task, 1 e 2, tra di loro paralleli e un'unica risorsa. Dati soltanto i vincoli proposti fino ad ora (in (16) e in (17)), non vi è nessuna condizione che vieti il verificarsi dal caso (a) di Fig. 4, ovvero un'assegnazione che obblighi la risorsa ad eseguire due task contemporaneamente. Per ovviare a questa situazione bisogna garantire che la fine del primo task avvenga prima dell'inizio del secondo o viceversa, come rappresentato nei casi (b) e (c) di Fig. 4:

$$start2 \geqslant end1 \quad \text{oppure} \quad start1 \geqslant end2. \tag{19}$$

Per implementare questa operazione logica di esclusione all'interno del modello, è stata introdotta la variabile binaria y_{ji}, definita in (15) e il numero intero M, scelto in modo da essere adeguatamente grande. Le relazioni precedente vengono quindi riscritte nel modo seguente:

$$\begin{aligned} start2 &\geqslant end1 \; - M \cdot y_{ji} \\ start1 &\geqslant end2 \; - M \cdot (1 - y_{ji}) \end{aligned} \tag{20}$$

Osserviamo che se il risolutore sceglie di far sì che il task j venga eseguito dopo di i, ovvero $y_{ji} = 1$, la prima delle due disequazioni è sempre vera, quindi ininfluente. La seconda invece realizza esattamente il vincolo desiderato. È possibile verificare che con $y_{ji} = 0$ accade esattamente il contrario. A questo punto è facile capire le ultime due formule del vincolo introdotto con

(18), basta semplicemente sostituire ai valori $\mathtt{start}$ e $\mathtt{end}$ le opportune sommatorie che grazie sempre al vincolo di occorrenza e alla prima disequazione di (18), mi permettono di individuare l'istante di tempo desiderato. L'operazione appena descritta deve essere ripetuta per ogni task j andando a valutare il vincolo con ogni $i \notin D_j$, ovvero per tutti i task che non hanno tra di loro un collegamento diretto.

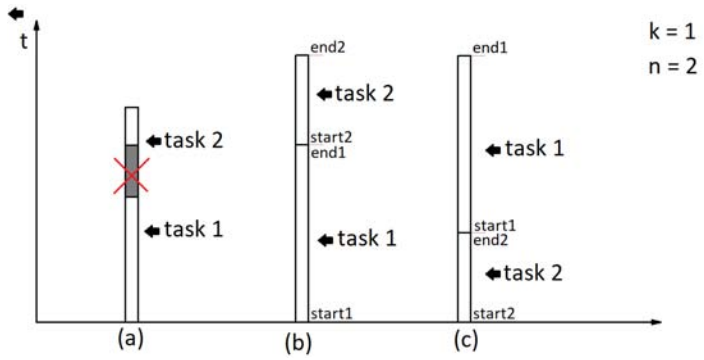


Figura 4: Spiegazione condizione di precedenza nei task paralleli.

- vincolo sul makespan:

$$C \geqslant \sum_{t \in T} \sum_{k \in K} (t + t_{jk}) \cdot x_{jkt}, \qquad \forall j \in V, \tag{21}$$

con il quale determiniamo il modo con cui deve essere calcolato il makespan a partire dall'assegnazione effettuata.

L'obbiettivo del risolutore sarà quindi quello di ricercare una task allocation che minimizzi il tempo necessario per l'assemblaggio del prodotto in questione:

$$\min C. \tag{22}$$

Il modello così definito oltre a soddisfare tutti i canoni della programmazione lineare con interi, rientra anche a far parte di una categoria di problemi di schedulazione nota in letteratura con il nome di *Resource-constrained project scheduling problem* o più brevemente RCPSP [18].

4

INIDICI DI VALUTAZIONE

In questo capitolo vengono presentati degli indici adimensionali per la valutazione del modello matematico implementato. Essi riguardano i dati caratteristici del problema sia in input che in output, ovvero: diagramma delle precedenze, durata dei task, makespan e collaborazione.

4.1 INDICE DI PARALLELIZZAZIONE

L' indice di parallelizzazione è un parametro introdotto per caratterizzare il diagramma delle precedenze. Viene indicato con il simbolo $p_{\%}$ e si calcola nel modo seguente:

$$p_{\%} = 1 - \frac{\sum_{j=1}^{n} \frac{D_j}{(n-1)}}{n}, \quad \text{con} \quad p_{\%} \in [1, 0] \tag{23}$$

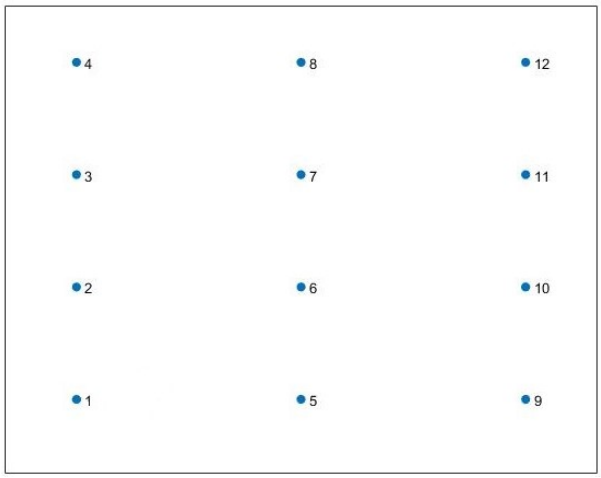
dove ricordiamo:

- n numero di task
- j indice del task
- V insieme di task j
- G grafico delle precedenze
- p_j numero di tutti i predecessori di j
- P_j^* insieme contenete tutti i predecessori di j
- s_j numero di tutti i successori di j
- S_j^* insieme contenete tutti i successori di j
- d_j numero di tutti i task che hanno un collegamento diretto con j.

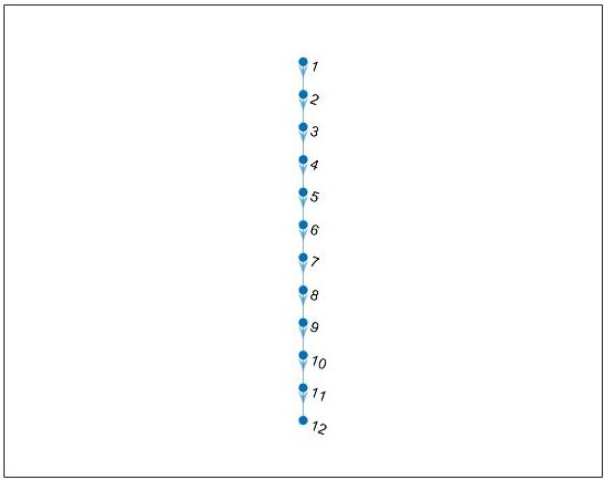
Come si osserva per il calcolo di $p_{\%}$ è necessario conoscere il numero totale dei task e il grafico delle precedenze, da cui è possibile ricavare il valore d_j nel modo seguente:

$$d_j = p_j + s_j. \tag{24}$$

Questo corrisponde al numero di task del problema che hanno un collegamento diretto con il task j-esimo preso in considerazione, ovvero come esplicita la formula è la somma del numero di predecessori e successori di j. Tale suddivisione è rappresentata in Fig.3 e riprende quella effettuata nel capitolo precedente quando è stato introdotto vincolo di precedenza tra task paralleli.
L'idea alla base della formulazione di questo indice può essere riassunta con i seguenti passi:



(a) $p_{\%} = 1$



(b) $p_{\%} = 0$

Figura 5: Caso con parallelizzazione massima e nulla.

- per ogni task j bisogna calcolare il numero di task $i \neq j$ che non possono essere eseguiti in parallelo con j, ovvero d_j;
- il risultato deve essere normalizzato per $(n-1)$, cioè il valore massimo di d_j.
- i valori trovati devono essere mediati sul numero n di task;
- il risultato deve essere scalato di un fattore 1.

In questo modo la formula descritta in (23) contiene l'informazione riguardante il grado di parallelizzazione del diagramma delle precedenze indipendentemente dal numero e dalla durata dei task. Il valore risultante sarà compreso tra 0 e 1, dove:

- il caso migliore dal punto di vista collaborativo viene raggiunto con $p_{\%}$ unitario, il quale corrisponde ad un diagramma del-

le precedenze formato da n task completamente indipendenti l'uno dall'altro, in cui si ha che:

$$\forall j \in V, \quad d_j = 0, \quad p_{\%} = 1; \tag{25}$$

- il caso peggiore invece è rappresentato da un indice di parallelizzazione con valore nullo, ovvero un diagramma delle precedenze in cui tutti i task devono essere eseguiti in modo sequenziale:

$$\forall j \in V, \quad d_j = n - 1, \quad p_{\%} = 0; \tag{26}$$

Questi due casi limite sono rappresentati in Fig. 5, mentre tutte le altre tipologie di diagramma delle precedenze ricadono all'interno di questo intervallo.

4.2 INDICE SUI TEMPI DEI TASK

Come l'indice di parallelizzazione anche il $t_{\%}$ va a caratterizzare uno dei parametri in ingresso: i tempi de esecuzione dei task. Viene calcolato nel modo seguente:

$$t_{\%} = \frac{\min\{\sum_{j=1}^{n} t_{j1}, \sum_{j=1}^{n} t_{j2}\}}{\max\{\sum_{j=1}^{n} t_{j1}, \sum_{j=1}^{n} t_{j2}\}}, \quad t_{\%} \in [0, 1], \tag{27}$$

dove ricordiamo:

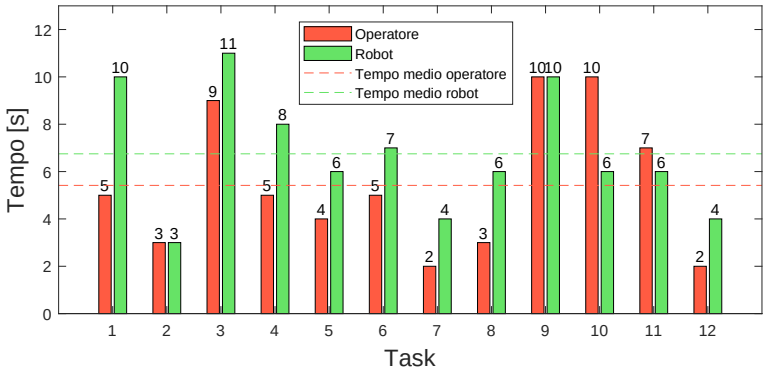
n	numero di task
j	indice del task
m	numero di risorse
k	indice della risorsa, $k \in \{1, 2\}$
t_{jk}	tempo di esecuzione task j per la risorsa k

L'obiettivo di tale indice è fornire un parametro per capire se è presente una disparità nelle velocità di esecuzione dei task tra le due risorse. Esso dipende esclusivamente dai valori t_{jk} e come nel caso precedente è compreso tra 0 e 1, dove:

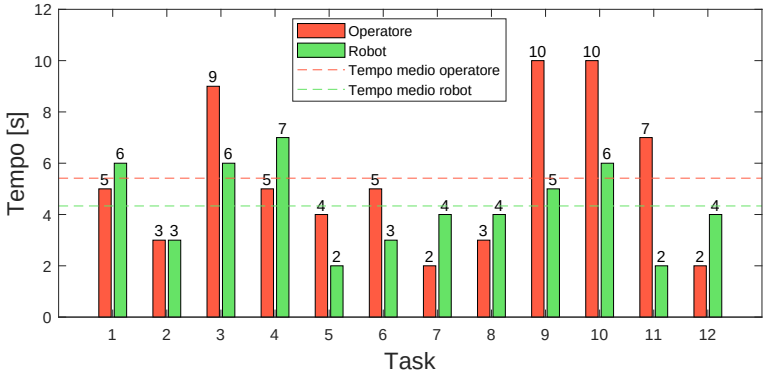
- un valore unitario indica che i tempi di esecuzione mediamente sono uguali tra le due risorse: questo implica che molto probabilmente robot e operatore porteranno a termine lo stesso task con tempistiche simili.

- un valore nullo non può esser mai raggiunto, ma al diminuire dell'indice aumenterà la differenza tra i tempi delle due risorse.

Inoltre è possibile osservare dalla Fig. 6 che il $t_{\%}$ non fornisce alcuna informazione a proposito di quale due risorse sia più lenta: osserviamo che i due casi hanno lo stesso indice ma mentre nel primo



(a) Caso in cui l'operatore è mediamente più veloce



(b) Caso in cui il robot è mediamente più veloce

Figura 6: Esempio dei tempi di esecuzione dei task con $t_{\%} = 0.8$.

l'operatore è mediamente più veloce, nel secondo è il robot ad aver tempi minori. Infatti la definizione data in (27) è stata formulata in modo da rendere l'indice indipendente oltre che dal numero di task e dal diagramma delle precedenze, anche dalla risorsa analizzata.

4.3 INDICE SUL MAKESPAN

Il makespan è definito come l'intervallo di tempo che intercorre tra l'inizio del primo task e la fine dell'ultimo. Esso è la variabile che il modello matematico deve minimizzare e come tale è anche un output del problema di allocazione. L'indice associato al makespan è il seguente:

$$m_{\%} = \frac{makespan[s]}{\min\{makespan^*_{|p_{\%}=0}\}[s]}. \tag{28}$$

Questa formulazione permette di riportare il dato calcolato durante la simulazione o misurato sperimentalmente, ad un valore indipendente dal numero di task e dal diagramma di precedenze utilizzato. Infatti il denominatore di (28) indica il makespan ottimo calcolato con parallelizzazione nulla, ovvero nella situazione rappresentata nella fi-

gura (b) di 5. Questo può esser calcolato facilmente prima della risoluzione del problema: esso può essere trovato allocando ogni task alla risorsa che ha il tempo di esecuzione minimo. La Fig. 7 da una rappresentazione di tale valore.

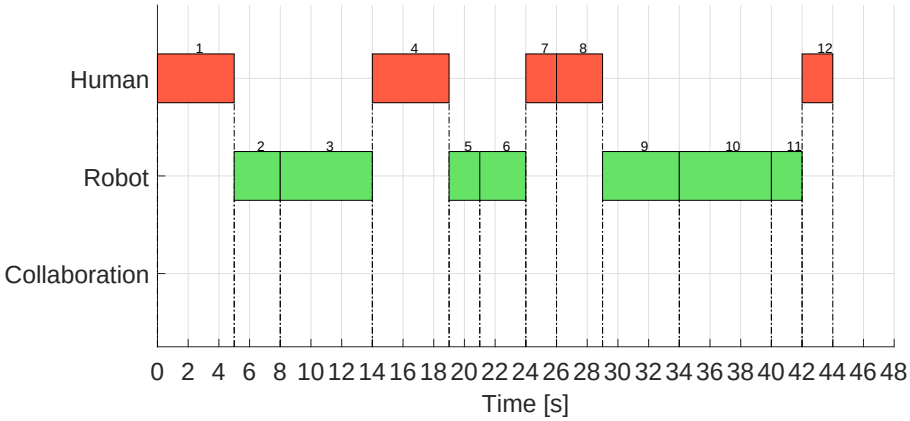


Figura 7: Esempio di makespan ottimo con $p_{\%} = 0$.

4.4 INDICE DI COLLABORAZIONE

Allo scopo di valutare i vantaggi nell'utilizzo dei cobot, viene riproposta la definizione dell'indice di collaborazione introdotta da Faccio et al. in [3]:

$$c_{\%} = \frac{T_{coll}[s]}{makespan[s]}, \quad c_{\%} \in [0, 1]. \tag{29}$$

Esso rappresenta in percentuale la frazione di tempo in cui il robot e l'operatore condividono lo spazio di lavoro (ovvero T_{coll}), rispetto al tempo di assemblaggio totale. È possibile ricavare tale dato a partire dalla schedulazione come mostrato in Fig. 8

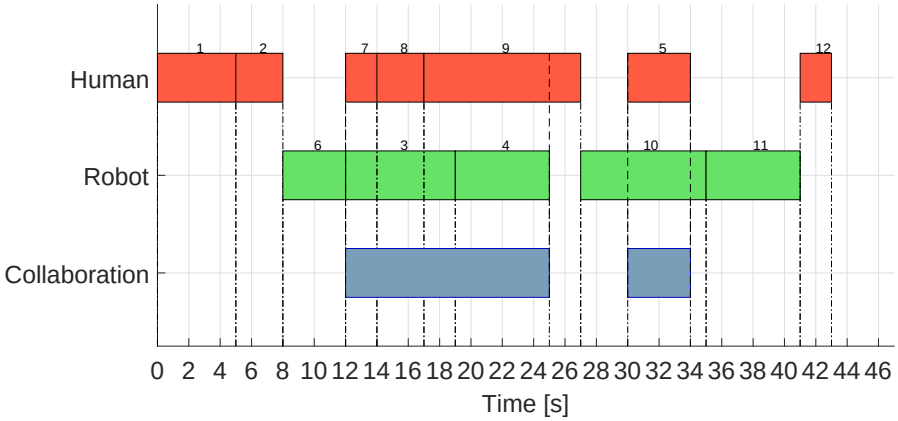


Figura 8: Esempio di scheduling con collaborazione tra le risorse.

5 SIMULAZIONI IN MATLAB

Elaborato il modello matematico, sono state effettuate diverse simulazioni mediante il software Matlab per la sua validazione e studio al variare degli indici di valutazione. Si descriverà quindi in un primo momento il software utilizzato, i modelli simulativi implementati e il metodo di risoluzione adoperato. Si procederà poi con il presentare le simulazioni effettuate e i risultati da esse ottenuti.

5.1 DESCRIZIONE DEL SOFTWARE DI SIMULAZIONE

Il software utilizzato per la simulazione è l'*Optimization Toolbox* di Matlab prodotto da *Mathworks* [9]. Esso offre funzioni per risolvere problemi di minimizzazione o massimizzazione degli obiettivi, siano essi definiti attraverso la programmazione lineare (LP), la programmazione non lineare (NLP), la programmazione lineare intera pura o mista (MILP), la programmazione quadratica (QP), i minimi quadrati lineari e non lineari. I solver di questo toolbox riescono a trovare soluzioni ottimali a problemi continui e discreti, a attività di stima dei parametri o applicazioni come ottimizzazione di portafoglio e ovviamente pianificazione della produzione.
Il software propone due metodi di approccio per la risoluzione del problema di ottimizzazione:

- ottimizzazione basata su solver: la funzione obiettivo e i vincoli del modello vengono riportati in Matlab usando funzioni matematiche e matrici di coefficiente, applicando un risolutore appropriato ai dati inseriti.
- ottimizzazione basato sui problemi: la funzione obiettivo e i vincoli vengono scritti usando espressioni che riflettono la forma matematica del problema. In questo caso si utilizzano quindi variabili di ottimizzazione, operatori matematici, operazioni con matrici e funzioni variabili. Il risolutore è standard e in automatico utilizzerà il processo più adatto a risolvere il problema implementato.

In questo elaborato per effettuare le simulazioni è stato utilizzato il secondo metodo, ovvero l'ottimizzazione basata sul problema. Per spiegare il funzionamento di tale tool di seguito verrà brevemente descritto il codice riportato in Appendice A. Esso implementa la formulazione matematica del problema descritta nel capitolo 3 ed è organizzato come segue:

- inizialmente vengono definiti i dati di input, ovvero come discusso durante la presentazione del modello:
  - n numero dei task;
  - m numero di risorse, il cui valore è 2 in quanto il layout considerato comprende solo un operatore umano e il cobot;
  - i tempi di esecuzione dei task per entrambe le risorse;
  - il diagramma delle precedenze, implementato attraverso un grafico a nodi;
  - il tempo di esecuzione massimo previsto, ovvero il valore del makespan ottimo con $p_{\%} = 0$, introdotto con la formulazione dell'indice $m_{\%}$ in (28).
- A seguire sono state definite attraverso la funzione `optimvar` la variabile da ottimizzare rappresentante il makespan e le variabili binarie x_{jkt} e y_{ji}. Queste inizialmente non hanno un valore definito, ma devono essere assegnate dal risolutore in modo che rispettino le condizioni imposte dal modello.
- La parte successiva prevede la scrittura di tutti i vincoli del modello introdotti nel terzo capitolo attraverso la funzione `optimconstr`.
- Utilizzando la funzione `optimproblem` viene creato un oggetto contenente i parametri del problema di ottimizzazione: deve essere dichiarato se è un necessario minimizzazione o massimizzazione la funzione obiettivo e specificare quali siano i vincoli a cui le variabili sono sottoposte.
- A questo punto è possibile risolvere il problema utilizzando il solver proposto da MATLAB. Però come si vedrà nella sezione successiva, tale strada non è percorribile poiché il risolutore standard data la complessità del problema non riesce a trovare un risultato in un tempo accettabile.
- Nella parte finale di codice è quindi stata introdotta una procedura che permette di ricavare a partire dall'oggetto creato precedentemente e contenente i parametri del problema, una struttura all'interno della quale troviamo le informazioni necessaria per avviare il risolutore utilizzato: la versione di solver sviluppata da CPLEX per i problemi di tipo MILP.

Questo script permette quindi di implementare e risolvere attraverso l'*Optimization Tool* un generico problema di task allocation. Esso però non è tutto il codice elaborato per effettuare le simulazioni; infatti sono state create varie versioni del file riportato, in base al tipo di prova che veniva svolta e all'analisi che doveva esser effettuata. Inoltre, sono state sviluppate una serie di funzioni utilizzate per l'elaborazione e la visualizzazione dei risultati ottenuti.

5.2.1 *Complessità di un problema di tipo PLI*

Il modello proposto formulato sulla base della programmazione lineare intera permette di trattare una grande varietà di problemi, ma la loro risoluzione nella maggior parte delle situazioni è molto complessa. Un ramo della teoria della computabilità che studia le risorse necessarie per la risoluzione di un problema è la *Teoria della complessità computazionale* [23]. Essa suddivide i problemi esistenti in varie categorie, dette classi di complessità, sulla base dell'efficienza del miglior algoritmo fino ad ora sviluppato per risolvere quello specifico problema.
Uno dei parametri di valutazione utilizzati da questi algoritmi è il tempo computazionale richiesto, il quale è misurato attraverso una *funzione di complessità propria*, la quale permette di effettuare una distinzione tra le tipologie di algoritmi presenti:

- algoritmi con tempo di risoluzione polinomiale: il tempo di computazione è limitato superiormente da una funzione polinomiale che dipende dalla lunghezza n degli input del problema;
- algoritmi con tempo di risoluzione esponenziale: il tempo di computazione aumenta esponenzialmente all'aumentare della grandezza del problema

È evidente che tra le due alternative presentate la prima è la migliore poiché si riesce ad arrivare ad una soluzione in un tempo finito: possono essere risolti in questo modo alcuni problemi di ottimizzazione e fattibilità, i quali rientrano nella classe NP-difficile (ovvero problemi difficili non deterministici a tempo polinomiale). Il caso trattato in questa dissertazione appartiene invece ad una sottocategoria di questa classe: esso è un problema NP-completo, che a differenza di quello precedente è risolvibile solo con algoritmi del tempo esponenziali. Questo è il motivo per cui una delle principali sfide affrontate durante l'implementazione del programma utilizzato per le simulazioni, è stato trovare un risolutore che fosse capace di trovare una soluzione in tempi accettabili.

5.2.2 *Metodo di risoluzione: Branch-and-Bound*

La maggior parte dei solveri utilizzano per la risoluzione dei problemi di programmazione lineare di interi il metodo *Branch-and-Bound*. Esso è un criterio applicabile a tutti i tipi di *problemi di ottimizzazione combinatoria*, i quali sono definiti come:

$$\min / \max f(x) \quad x \in X, \tag{30}$$

dove X è un insieme finito di soluzioni e $f(x)$ una generica funzione obiettivo. Per i problemi in forma combinatoria di tipo PLI esistono metodi risolutivi esatti di complessità esponenziale, oppure se è accettabile arrivare ad una soluzione buona ma che potrebbe non essere quella ottima, sono presenti algoritmi più efficienti basati su approcci euristici e meta-euristici.
Ritornando al *Branch-andBound*, esso fa parte dei metodi che permettono di arrivare ad una soluzione esatta e si basa su uno schema detto *Algoritmo universale per ottimizzazione combinatoria*, cosi definito:

1. per prima cosa bisogna generare tutte le possibili soluzioni di x;
2. per ogni soluzione è necessario verificare se è ammissibile;
3. in seguito viene valutata $f(x)$;
4. infine deve esser scelta la x che garantisce la $f(x)$ migliore.

Questo schema è molto semplice e facile da comprendere ma soffre di tre importanti problemi:

- potrebbe non essere facile valutare la $f(x)$;
- il numero di soluzioni x potrebbe essere molto elevato;
- anche la cardinalità di X potrebbe essere molto elevata.

In particolare, dall'ultima osservazione sorgono due temi non trascurabili: come deve essere generato lo spazio delle soluzioni, e come devo esplorarlo in modo efficiente. L'algoritmo di *branch-and bound* affronta principalmente proprio quest'ultimi due quesiti:

- la generazione delle soluzione avviene attraverso l'operazione di *Branch*. Essa si basa sulla seguente affermazione [5]:

  > *Dato un problema di ottimizzazione combinatoria* $z = \min / \max\{f(x) : x \in X\}$ *e data una suddivisione della regione ammissibile* X *in insiemi* $X_1, X_2, \ldots, X_n$ *si ha che* $\cup_{i=1}^{n} X_i = X$*, con* $z^{(k)} = \min / \max\{f(x) : x \in X_k\}$.
  > *Allora la soluzione del problema* $z = \min / \max_{k=1,\ldots,n} z^{(k)}$.

  È possibile di conseguenza dividere X in sottoinsiemi più piccoli e risolvere il problema per ognuno di essi. Questa operazione viene effettuata in modo ricorsivo, dividendo a loro volta le regioni dei sotto-problemi dove è ammissibile che si trovi una soluzione realizzabile. Questo processo dà luogo ad un albero delle soluzioni ammissibili: avremo un nodo radice che comprende l'insieme di tutte le soluzioni (ovvero X), dal quale si diramano due rami a cui corrispondo due nodi figli tali che:

$$X_i = \cup_{j \text{ figlio di } i} E_j, \tag{31}$$

ovvero , le soluzioni presenti nel padre devono essere presenti anche in almeno uno dei sui figli. Infine, questa operazione di suddivisione si ferma nel momento in cui ogni nodo contiene una sola soluzione.

- L'esplorazione efficiente avviene invece attraverso l'operazione di *Bound*. Generalmente l'operazione di branch produce un numero di soluzioni da analizzare che cresce in modo esponenziale, corrispondente all'enumerazione di tutte le soluzioni in X e pertanto non è analizzabile nella sua completezza. Viene quindi ricercato un metodo per esplorare solo le aree "buone" delle regioni ammissibili: per ogni nodo dell'albero viene calcolato un **Bound**, ovvero una valutazione ottimistica del valore che la $f(x)$ può assumere. Questo limite ci permette di eliminare sottoalberi che sicuramente non hanno al loro interno la soluzione ottima, ovvero quelli che hanno un buond peggiore di quello calcolato fino a quel momento,

A partire da queste considerazioni e dato un problema di ottimizzazione combinatoria $z = \min / \max\{f(x) : x \in X\}$, con:

- P_0:problema iniziale;
- L: insieme di nodi aperti per ogniuno dei quali si ha P_i e B_i, ovvero il sotto-problema e il realtivo buond;
- z^*: valore della migliore soluzione ammissibile;
- x^*: migliore soluzione corrente;

è possibile schematizzare il metodo *Branch and Bound* con i seguenti passi:

1. *Inizializzazione*: stima di B_0 della funzione obiettivo, e imposizione di $L = \{(P_0, B_0\}$
2. *Criterio di stop*: se L è vuoto allora x^* è soluzione ottima. Se vengono superati i limiti di tempo, nodi esplorati, nodi aperti ecc... x^* è una soluzione amissibile
3. *Selezione nodo*: Selezione di (P_i, B_i) per effettuare il branch
4. *Branching*: divisione di P_i in sotto-problemi P_{ij}.
5. *Bounding*: per ogni sotto-problema valuto il relativo B_{ij}
6. *Fathoming*: se P_{ij} non è ammissibile o B_{ij} non è migliore di z^* bisogna tornare al punto 1. Se la soluzione è ammissibile e è migliore di z^*, allora B_{ij} è il nuovo z^*, viene aggiornata pure x^* ed eliminati tutti i nodi con bound peggiore di questo. Altrimenti si deve aggiungere (P_{ji}, B_{ji}) a L e tornare al punto 1.

Questo sopra esposto è solo un'indicazione di come possa esser risolto un problema di ottimizzazione combinatoria, poi ogni risolutore utilizza criteri e operazioni diverse per effettuare tali operazioni, ed è proprio questo che li distingue quando si vanno a valutare le prestazioni che essi hanno.

5.2.3 *Risolutori utilizzati*

Inizialmente è stato utilizzato il risolutore standard presente nell'Optimization Tool chiamato `intlinprog`. La strategia che esso adotta si basa su un metodo procedurale formato da 6 step svolti consecutivamente:

- inizialmente viene studiato il problema lineare senza considerare le variabili intere e i vincoli che agiscono su di esse. Questa fase comprende:
  - un'operazione di pre-processing in cui vene ridotta la dimensione del problema ad esempio eliminando eventuali ridondanze e calcolando eventuali punti di infattibilità del modello.
  - la risoluzione di questo problema "rilassato" in modo da ricavare un eventuale limite superiore per la variabile da ottimizzare.
- In seguito il solver analizza anche i vincoli che agiscono sulle variabili intere per determinare se il problema è fattibile, se è possibile rendere i limiti più stringenti, se è possibile rimuovere i vincoli ridondanti e rafforzare i rimanenti. L'obiettivo principale di questa fase è semplificare il problema e far sì che negli step successivi sia possibile ricavare la soluzione ottima più facilmente.
- L'operazione successiva che esegue il solver è detta *"Cut generations"* e consiste nell'aggiunta di vincoli lineari sotto forma di disequazioni. Queste hanno come obiettivo la riduzione della regione all'interno della quale è possibile trovare la soluzione del problema.
- Vengono poi utilizzate delle procedure euristiche per la ricerca di possibili punti di fattibilità prima e durante la fase di branch e bound: questo passaggio permette di calcolare eventuali limiti superiori sulla funzione obiettivo.
- l'ultima parte di questo procedimento è anche il più importante: viene implementata la tecnica di *branch-and-bound* spiegata precedentemente, la quale costruisce una sequenza di sotto problemi con l'obiettivo di convergere alla soluzione del MILP. Questo avviene perché ogni sotto-problema permette di aggiungere un

limite inferiore o superiore alla regione di appartenenza della soluzione ricercata

Il risolutore proposto dall'*Optimization Tool* è stato utilizzato per effettuare le simulazioni nella prima parte di analisi del modello di bilanciamento. In seguito a causa dell'aumento del numero di variabili in gioco, come il numero di task e la loro durata, i tempi di esecuzione dell'algoritmo di ottimizzazione sono diventati troppo elevati. Si è deciso quindi di utilizzare il software *CPLEX* [6] sviluppato da *IBM*: questo oltre ad essere una delle migliori soluzioni presenti per la risoluzione di problemi di ottimizzazione, ha la possibilità di interfacciarsi facilmente a Matlab installando l'apposito pacchetto. Il punto di forza di tale software risiede nell'algoritmo utilizzato per la risoluzione dei problemi PLI, ovvero il *"Branch and Cut algorithm"*, il quale comprende una serie di sofisticate caratteristiche che gli permettono di incrementare drasticamente la velocità di risoluzione. Il procedimento risolutivo utilizzato e risulta molto simile a quello descritto nel caso del solver standard di Matlab, però dato che il software in questione è attualmente commercializzato per l'utilizzo aziendale oltre che a quello accademico non è possibile conoscere le reali tecniche implementate.
In ogni caso data la complessità del problema studiato, in cui il numero di variabili decisionali è estremamente alto, è stato necessario definire anche con il software *CPLEX* un tempo limite di un'ora per l'esecuzione dell'ottimizzazione. Questa decisione ha fatto si che nel caso di simulazioni con un elevato numero di task e tempi di esecuzione molto elevati, le soluzioni ricavate non erano necessariamente quelle ottime. Per risolvere questo problema in futuro potrebbe esser valutato un processo di risoluzione che faccia uso di metodi meta-euristici o algoritmi genetici.

5.3 INTRODUZIONE ALLE SIMULAZIONI EFFETTUATE

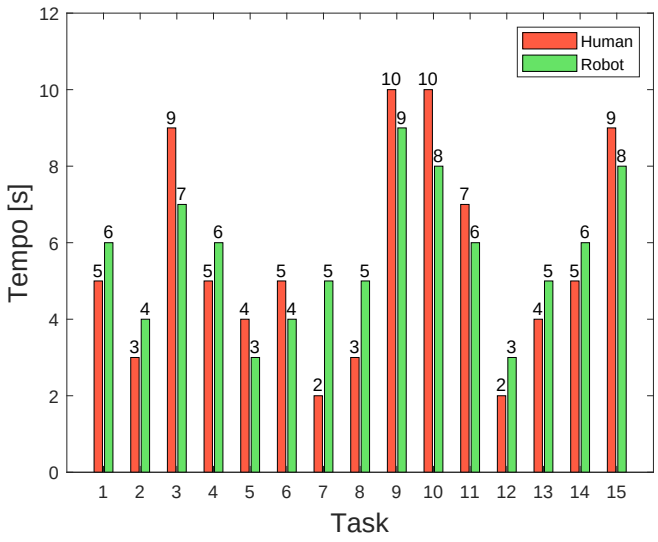
L'obiettivo delle simulazioni è quello di valutare il modello sviluppato. Questo avverrà in due modi: prima analizzando gli indici presentati nel capitolo precedente, poi attraverso un confronto con la versione non ottimizzata per i sistemi collaborativi.

5.3.1 *Descrizione delle simulazioni per la valutazione degli indici*

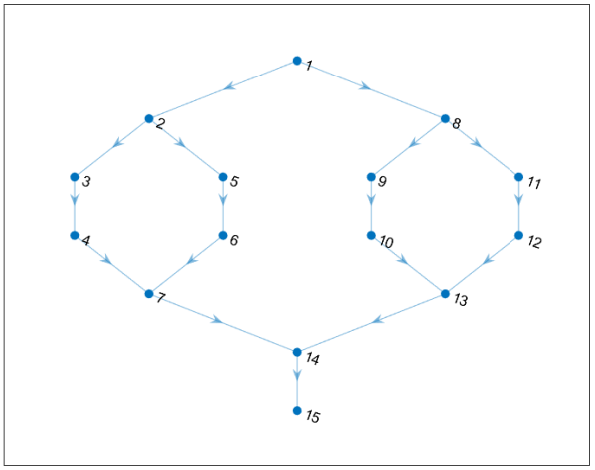
Riprendendo alcune considerazioni effettuate precedentemente, ricordiamo che è possibile individuare per il nostro modello dei dati caratteristici di input e output:

- in ingresso troviamo n numero dei task, t_{jk} tempi di esecuzione dei task e G il diagramma delle precedenze. Un esempio di set di dati di input utilizzati nelle simulazioni è riportato in Fig. 9;

- in uscita abbiamo un grafico carta-uomo-macchina da cui ricaviamo C ovvero il makespan, e il tempo di collaborazione definito come T_{coll}, Un esempio di output è riportato in Fig. 10.



(a) Numero e durata dei task



(b) Diagramma delle precedenze

Figura 9: Esempio di dati di input nelle prove di simulazione.

L'obiettivo delle simulazioni sarà quello di valutare l'andamento degli output al variare degli ingressi, prendendo in considerazione gli indici presentati al capitolo precedente.
La prima di queste prove analizza per ogni set di dati la percentuale di collaborazione e l'indice sul makespan al variare dell'indice di parallelizzazione. Viene di conseguenza analizzata l'influenza del dia-

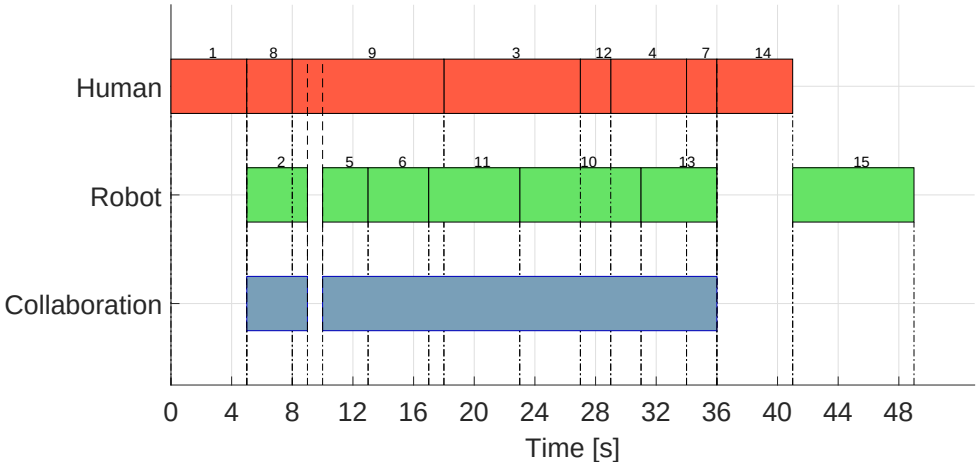


Figura 10: Esempio di schedulazione ottenuta dalle prove di simulazione.

gramma delle precedenze, andando a valutare $m_{\%}$ e $c_{\%}$ al variare del $p_{\%}$. La prova è effettuata mantenendo costanti i tempi dei task mentre viene variato il loro numero, ovvero n. In Tab. 3 sono riportati i dati utilizzati.

Tabella 3: Tabella riassuntive prove effettuate su $p_{\%}$.

$p_{\%}$	$t \in [2s\ 10s]$									
$n = 10$	0.08	0.13	0.2	0.31	0.4	0.55	0.6	0.73	0.82	0.89
$n = 12$	0.06	0.13	0.22	0.27	0.42	0.5	0.6	0.75	0.82	0.91
$n = 15$	0.04	0.1	0.19	0.27	0.42	0.48	0.6	0.71	0.75	0.85
$n = 17$	0.07	0.14	0.25	0.31	0.41	0.52	0.59	0.65	0.71	0.8
$n = 20$	0.05	0.13	0.19	0.33	0.39	0.46	0.6	0.66	0.7	0.78

Ovviamente per ottenere valori di $p_{\%}$ diversi nelle varie configurazioni, devono essere elaborati diversi diagrammi delle precedenze. Purtroppo, non è possibile automatizzare questo passaggio, di conseguenza ogni grafico deve essere generato e solo successivamente si potrà calcolare il valore dell'indice. In ogni caso i diagrammi sono stati costruiti in modo che i valori di $p_{\%}$ studiati siano il più possibile simili tra di loro al variare di n.
Successivamente son state effettuate delle simulazioni per valutare l'effetto dei tempi dei task, ovvero è stato variato il $t_{\%}$. In questo caso è stata implementata in Matlab una semplice funzione che a partire dai tempi di una delle due risorse calcola i tempi dell'altra in modo casuale, ma facendo sì che il valore dell'indice sia quello desiderato: data la facilità con cui è possibile variare tale ingresso è stato anche possibile effettuare un numero maggiore di prove, variando il $t_{\%}$ da 0.2 a 1 con una risoluzione di 0.05. Come fatto precedentemente l'a-

nalisi viene svolta al variare del numero n di task e, per valutare l'eventuale effetto combinato tra diagramma di precedenze e tempi, anche al variare di $p_{\%}$. I dati utilizzati sono riportati in Tab. 4

Tabella 4: Tabella riassuntiva prove effettuate sul $t_{\%}$.

$p_{\%}$	$t_{\%} \in [0.2, 1]$		
$n = 10$	0.667	0.488	0.27
$n = 12$	0.67	0.49	0.3
$n = 15$	0.7	0.48	0.27

In questa simulazione le prove sono state limitate a casi con un numero di task basso. Questa scelta è dovuta al fatto che per valori di n e $t_{\%}$ elevati, il risolutore non riesce a riportare una schedulazione valida, poiché i tempi di elaborazione del risultato eccedono il limite imposto in fase di programmazione. Tale limitazione discussa anche nella sezione 5.2, conduce a valori di makespan distanti da quello ottimo. Di conseguenza le prove risultanti ad alti $t_{\%}$ non possono essere ritenute valide.

5.3.2 *Descrizione delle simulazioni per confronto tra i modelli di bilanciamento*

Nel capito 3 sono stati presentati due modelli matematici per l'allocazione dei task in un sistema composto da un operatore e un robot. Come precedentemente detto il primo è attualmente utilizzato per il bilanciamento delle linee di assemblaggio cadenzate e risolve il ben noto problema del SALBP. È stato introdotto perché implementando una semplice variazione nelle caratteristiche del layout, come già rappresentato in Fig. 2, può essere utilizzato anche in un sistema collaborativo. Si ricorda però che in questo caso la funzione obiettivo è il tempo di ciclo, una variabile che perde significato nel caso tipico della collaborazione, dove è presente una singola stazione in cui le risorse condividono lo spazio di lavoro. Lo scopo di questa simulazione è quindi dimostrare che la formulazione implementata successivamente con funzione obiettivo il makespan, è più adatta e permette di ottenere prestazione in termini di collaborazione e produttività migliori. Le prove di confronto sono state effettuate variando e combinando i seguenti dati ingresso:

- $n = 10, 15$;
- $p_{\%} = 0.27, 0.67$;
- $t_{\%} = 0.4, 0.8$.

Sono state effettuate un numero limitato di prove perché analizzando i risultati è evidente come in qualsiasi caso la soluzione migliore de-

riva dalle simulazioni effettuate con il modello matematico che tiene conto della variabile tempo.

5.4 RISULTATI DELLE SIMULAZIONI

In questa sezione vengono riportati i risultati delle varie simulazioni descritte.

5.4.1 *Simulazioni sull'indice di parallelizzazione*

Per analizzare le prove di simulazione descritte in Tab. 3, sono stati plottati tre grafici riassuntivi dei risultati ottenuti:

- nel primo sono rappresentati i valori dell'indice di makespan ottenuti, in funzione dell'indice di parallelizzazione;
- il secondo grafico ritrae l'andamento dell'indice di collaborazione al variare sempre dell'indice di parallelizzazione;
- nell'ultimo vengono messi in relazione $m_{\%}$ e $c_{\%}$.

In tutti i casi i campioni rappresentati derivano dalle diverse prove effettuate con tutti i valori di n analizzati: si è valutato infatti che in questo caso le prove non risultano influenzate da tale parametro.
Inoltre, per una più chiara comprensione dei risultati, sono state estrapolate delle curve di fitting di forma polinomiale e calcolate attraverso un'analisi ai minimi quadrati. Il parametro R^2, utilizzato come principale indice di bontà per l'analisi della regressione lineare, è risultato in ognuno dei tre casi maggiore di $0,95$, dimostrando così la buona affidabilità dell'approssimazione trovata. Le principali osservazioni che possono essere effettuate a partire da tali grafici son le seguenti:

- osservando la Fig. 11 si nota che per parallelizzazione nulla, ovvero con n task tra di loro in serie come in Fig. 9, il valore di $m_{\%}$ è unitario: questo risultato conferma che il problema di allocazione viene risolto attraverso il modello matematico in modo ottimo, quindi scegliendo per ogni task la risorsa che ha il tempo di esecuzione minore.
- Sempre dalla stessa figura è possibile constatare che all'aumentare dell'indice $p_{\%}$ il makespan diminuisce fino a diventare asintotico ad un valore $1/m$, che in questo caso corrisponde ad 0.5. Questo valore può essere raggiunto solo in una situazione ideale dipendente dal numero dei task e dalla loro durata, e garantisce sempre un indice di collaborazione unitario. Non è escluso che un valore di $c_{\%}$ massimo possa esser raggiunto anche per $m_{\%}$ prossimi al valore limite.

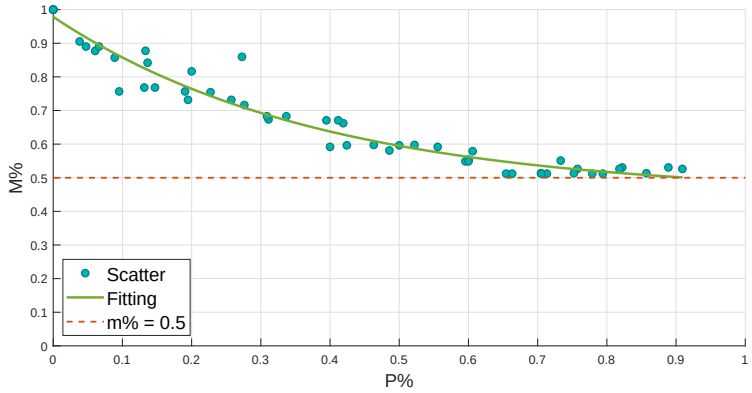


Figura 11: Andamento dell'indice sul makespan al variare del $p_{\%}$.

- Valutando invece la Fig.12 si nota che giustamente si ha una collaborazione nulla nel caso di parallelizzazione nulla; poi questa aumenta fino ad arrivare a valori prossimi alla collaborazione unitaria per $p_{\%}$ elevati. La certezza della $c_{\%}$ massima si ha con parallelizzazione unitaria, anche se osserviamo,sulla base di quanto detto precedentemente, che ciò non implica indice sul makespan pari a $1/m$.

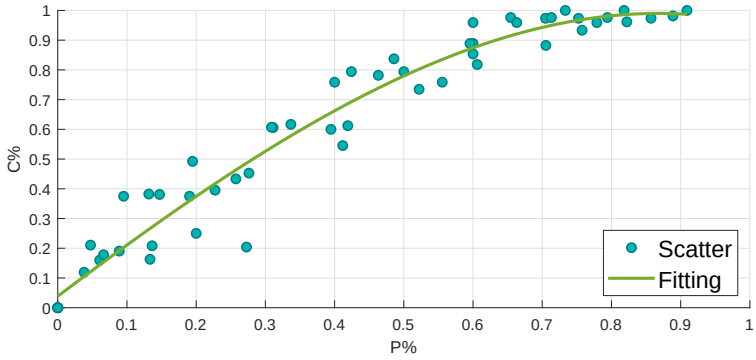


Figura 12: Andamento dell'indice di collaborazione al variare del $p_{\%}$.

- In Fig. 13 è facile notare che per $c_{\%}$ nullo, $m_{\%}$ è sicuramente unitario, mentre all'aumentare della collaborazione il tempo di esecuzione totale dell'assemblaggio diminuisce.

I risultati ottenuti dalle simulazioni confermano quelle che erano le aspettative di tali prove. Infatti si è dimostrato che un diagramma delle precedenze con una più alta possibilità di parallelizzazione permette di migliorare sostanzialmente la produttività dell'intero sistema. Questo risultato può essere giustificato considerando che in questa situazione vi è maggior libertà per il modello di allocazione di assegnare i task, e quindi di ottimizzare il makespan, condizione che porta conseguentemente all'aumento pure della collaborazione.

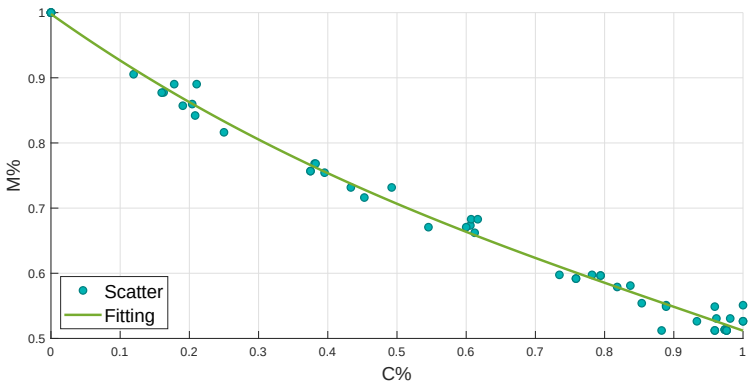


Figura 13: Andamento dell'indice sul makespan al variare del $c_{\%}$.

5.4.2 *Simulazioni sui tempi dei task*

Con le simulazioni effettuate utilizzando i dati descritti inTab. 4 viene valutata l'influenza dei temi dei task sulla schedulazione. In questo caso l'analisi si limiterà a prendere in considerazione solo gli effetti del $t_{\%}$ sul makespan, in quanto non è possibile ricavare dalle prove effettuate una relazione tra la percentuale di collaborazione e tempi dei task. Le principali osservazioni che possono essere effettuate a partire dalla Fig.**??** son le seguenti:

- Al diminuire dei tempi dei task si ha che il makespan stesso diminuisce, fino a portarsi per un determinato $t_{\%}$ ad un valore all'incirca costante. Tale comportamento può essere dovuto da due motivi:
  - il decremento può essere dato dalla minor variabilità dei tempi che permette al modello di assegnare i task in modo bilanciato, influenzando positivamente il makespan;

$$m_{\%} = \frac{makespan[s]}{\min\{makespan_{|p_{\%}=0}\}[s]} \tag{32}$$

  - l'andamento non è di tipo lineare a causa della definizione stessa di $m_{\%}$. Riprendendo la formula (32) vediamo che questo è il rapporto tra il makespan ricavato dalla simulazione e il makespan ottimo con parallelizzazione nulla. Entrambi i membri sono influenzati dai tempi dei task: essi diminuiscono all'aumentare dell'indice $t_{\%}$. Si è notato però che tale decremento risulta meno pronunciato nel valore a denominatore, rispetto al dato di simulazione a numeratore.
- Analizzando le tre figure contemporaneamente si può notare che il trend discusso nella sezione precedente secondo il quale l'indice $m_{\%}$ diminuisce all'aumentare del $p_{\%}$ è confermato. Tale

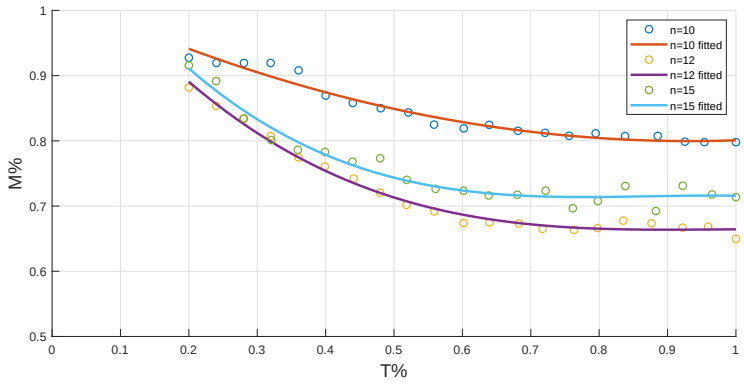


Figura 14: Valutazione di $m_{\%}$ al variare di $t_{\%}$ e n con parallelizzazione bassa.

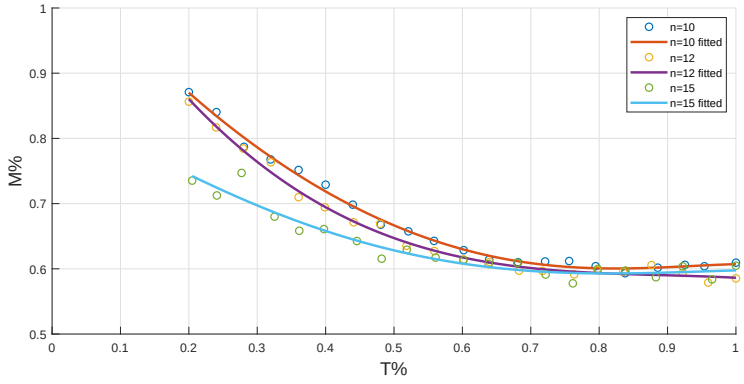


Figura 15: Valutazione di $m_{\%}$ al variare di $t_{\%}$ e n con parallelizzazione media.

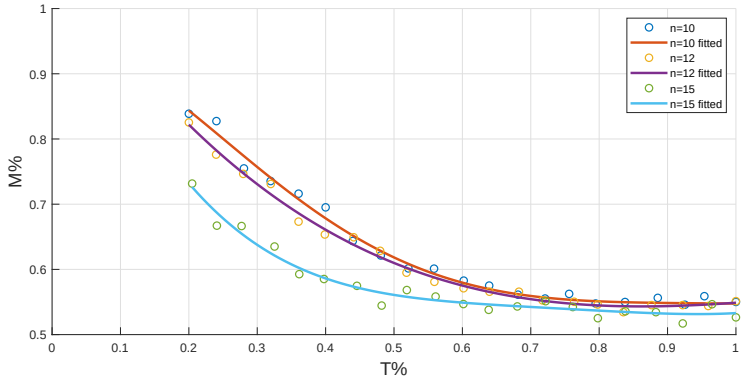


Figura 16: Valutazione di $m_{\%}$ al variare di $t_{\%}$ e n con parallelizzazione alta.

osservazione è maggiormente apprezzabile nelle Fig.17, 18 e 19, dove è stato ricostruito l'andamento del makespan al variare sia dell'indice di parallelizzazione che del $t_{\%}$. In questo grafico per semplificare l'analisi è stata effettuata un'interpolazione lineare dei dati, la quale seppur meno precisa permette lo stesso di analizzare gli andamenti dei parametri. In questo modo è stato anche possibile osservare che la diminuzione del makespan al variare dei tempi è più marcata per valori di $p_{\%}$ elevato.

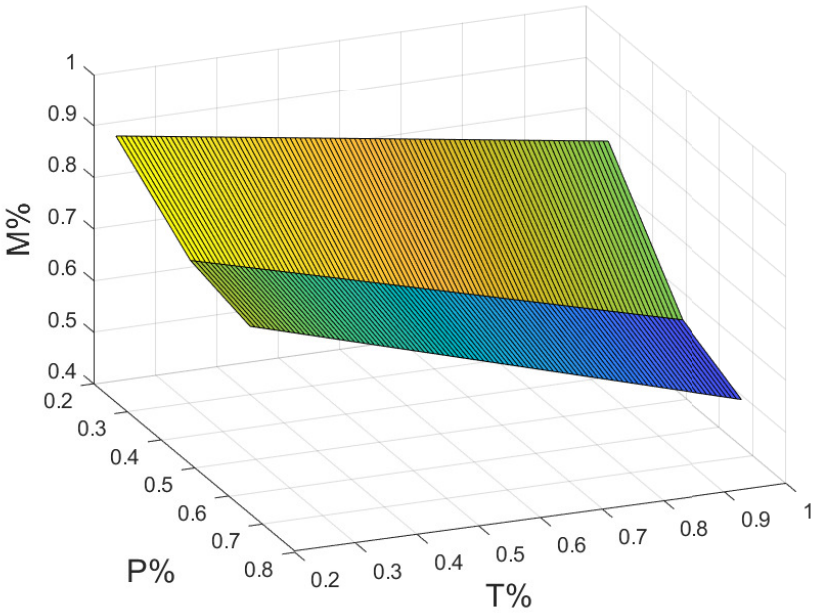


Figura 17: Andamento dell'indice sul makespan al variare di $t_{\%}$ e $m_{\%}$ con $n = 10$.

Si può concludere che per ottimizzare il processo di assemblaggio conviene avere tempi di esecuzione che tra le due risorse siano il più possibile simili tra di loro. Tale considerazione risulta valida non tanto per il fatto che l'indice di makespan diminuisce all'aumentare del $t_{\%}$, poiché come abbiamo visto tale andamento risulta alterato dalla definizione stessa di $m_{\%}$; ma il principale vantaggio deriva dal fatto che è possibile ottenere un effetto maggiore dalla variazione del $p_{\%}$ se l'indice sui tempi è alto.

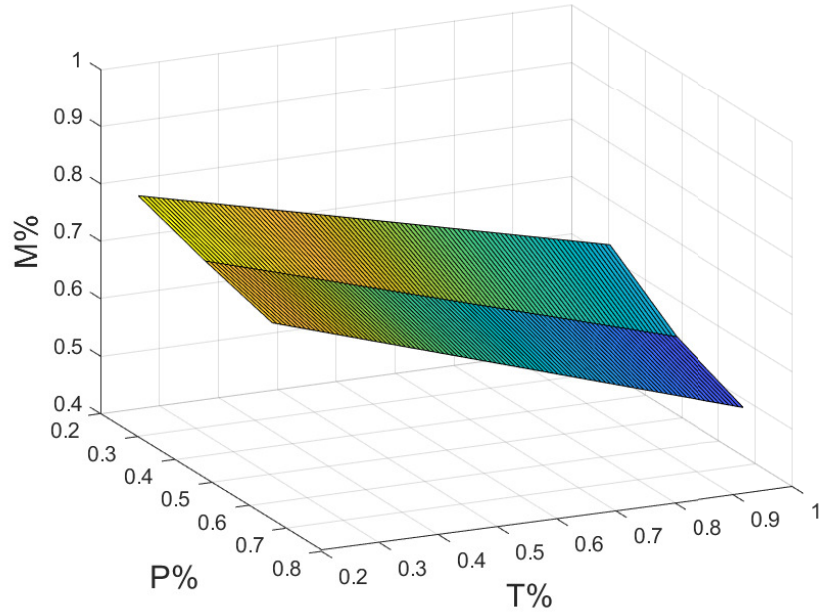


Figura 18: Andamento dell'indice sul makespan al variare di $t_{\%}$ e $m_{\%}$ con $n = 12$.

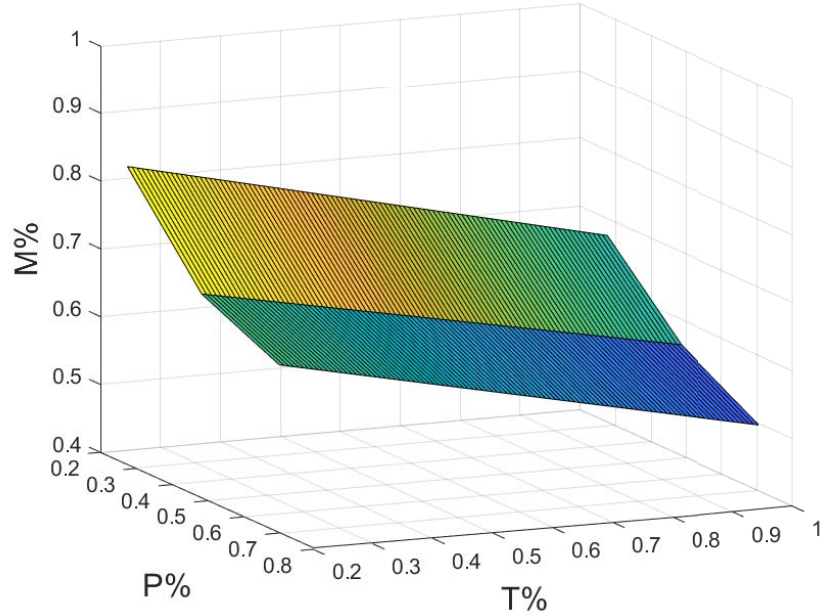


Figura 19: Andamento dell'indice sul makespan al variare di $t_{\%}$ e $m_{\%}$ con $n = 15$.

5.4.3 *Simulazioni per il confronto dei modelli di bilanciamento*

Come anticipato durante la descrizione di questa simulazione dalle prove effettuate emerge come il modello implementato risulti migliore in tutte le situazioni rispetto al SALBP riadattato per un sistema collaborativo. A dimostrazione di questo fatto vengono riportate in Fig 21 e Fig. 20 due delle prove effettuate: a parità di n sono state confrontate le diverse schedulazioni risultanti prima con condizioni il più possibili sfavorevoli alla collaborazione ($p_{\%} = 0,27$ e $t_{\%} = 0.4$) poi con quelle migliori ($p_{\%} = 0,67$ e $t_{\%} = 0.8$). I risultati sono riportati in Tab. 5 e 6.

Tabella 5: Confronto risultati con $p_{\%} = 0.27$ e $t_{\%} = 0.4$.

	Sequenziale	Collaborativo
$m_{\%}$	1,2	0,86
$c_{\%}$	0,1	0,59
makespna [s]	77	53
Tcoll [s]	5	31

Tabella 6: Confronto risultati con $p_{\%} = 0.67$ e $t_{\%} = 0.8$.

	Sequenziale	Collaborativo
$m_{\%}$	0,833	0,56
$c_{\%}$	0,54	0,8
makespan [s]	45	30
Tcoll [s]	16	24

È possibile osservare che in entrambe le prove riportate la schedulazione dei task risulta migliore nelle prove in cui viene utilizzato il modello sviluppato appositamente per l'ambito collaborativo. Tali conclusioni possono considerarsi valide per entrambi gli output valutati, ovvero makespan e tempo di collaborazione dato che questi sono direttamente collegati come giustificato durante lo studio dell'indice di parellizzazione.
Si osserva inoltre che nel caso di balanging effettuato con il modello non collaborativo si trovano addirittura valori di $m_{\%}$ maggiori di 1: come già discusso precedentemente tale effetto è da imputare al fatto che il modello di bilanciamento basato sul SALBP non tiene conto della variabile tempo e di conseguenza data la forma della sua funzione obiettivo riesce a minimizzare solamente il tempo di ciclo. Quest'ultima grandezza però perde di significato con un layout collaborativo poiché non sono presenti più stazioni da bilanciare ma soltanto due

risorse che condividono lo stesso spazio di lavoro. Il risultato finale, apprezzabile nelle Fig. 21 e 20, dimostra quindi che il makespan non risulta minimizzato, ma al contrario viene aumentato in modo da far sì che il carico di lavoro tra le due risorse risulti bilanciato.

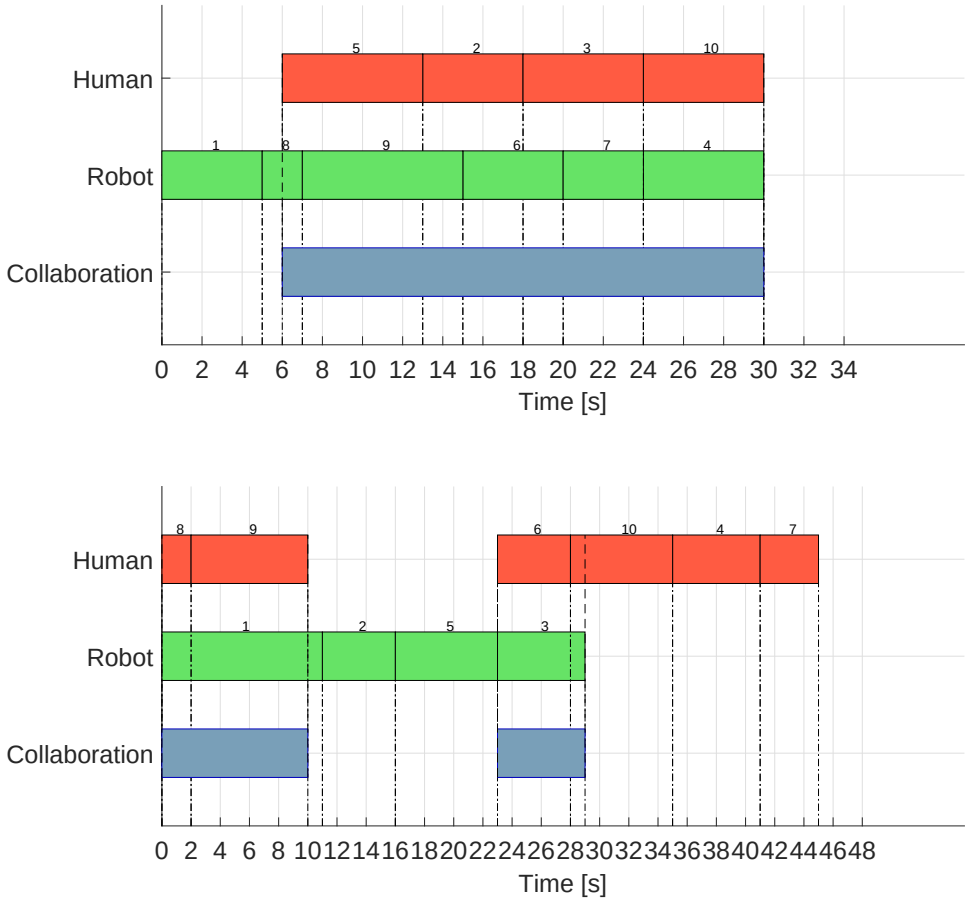


Figura 20: Confronto tra le schedulazioni risultati con $p_{\%} = 0.67$ e $t_{\%} = 0.8$.

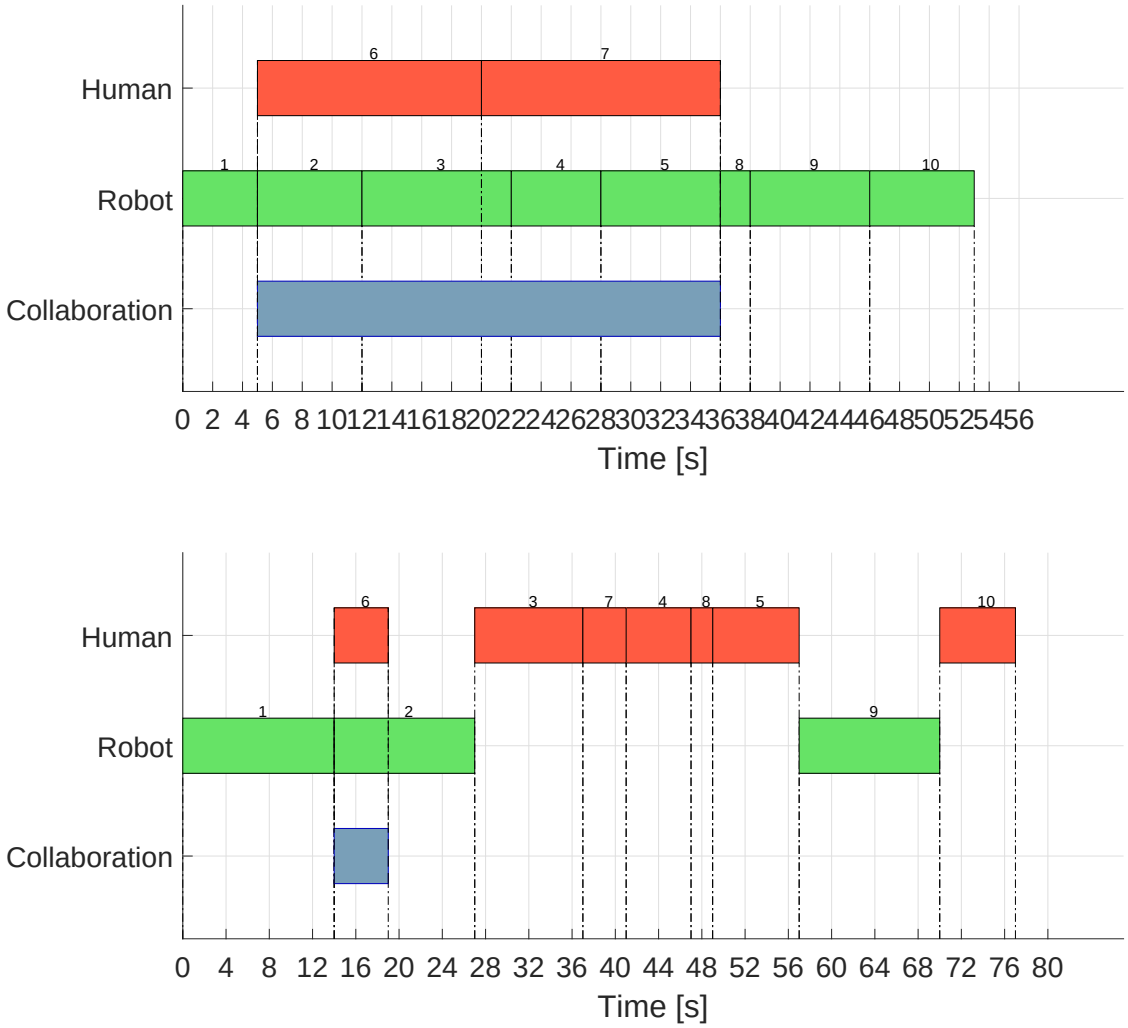


Figura 21: Confronto tra le schedulazioni risultati con $p_{\%} = 0.27$ e $t_{\%} = 0.4$.

6 VALIDAZIONE SPERIMENTALE DEL MODELLO

Per verificare e validare i risultati teorici ottenuti con le simulazioni sono state effettuate delle prove sperimentali utilizzando il robot collaborativo *KUKA LBR iiwa 14 820R*, in dotazione al laboratorio di robotica del DTG. Esse consistono nel simulare parte dell'assemblaggio di un prodotto finito, andando a misurare i valori di makespan e collaborazione ottenuti.
Nella prima parte di questo capitolo verrà presentato l'apparato strumentale e l'ambiente di programmazione utilizzato, di seguito si discuterà di alcune caratteristiche del robot che permettono di implementare le funzioni collaborative. Infine sarà descritta la prova sperimentale vera e propria.

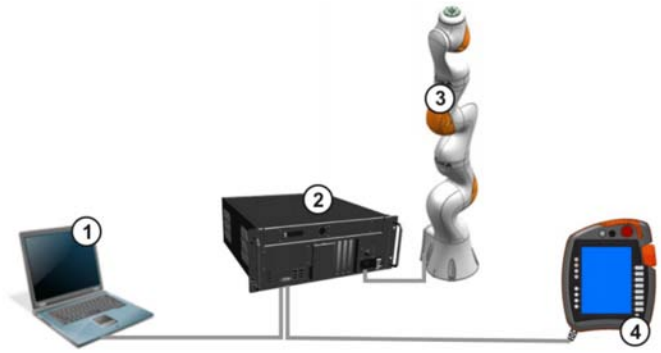


Figura 22: Setup sperimentale [8].

6.1 SETUP SPERIMENTALE

In questa sezione sarà presentato il sistema fornito da KUKA, ponendo particolare attenzione agli aspetti tecnici e di sicurezza, analizzando il software sviluppato per implementare le funzionalità collaborative sulla base delle capacità del robot stesso.

6.1.1 *Apparato strumentale*

Il robot *LBR iiwa 14 820R* è un manipolatore industriale a sette assi realizzato dall'azienda tedesca KUKA. Esso appartiene alla categoria dei cobot e quindi può essere impiegato per sviluppare innovativi processi di produzione dove uomini e robot possono lavorare fianco a fianco nella risoluzione di compiti altamente complessi.

I dati tecnici del robot estrapolati dal catalogo [8] del produttore sono riportati in Tab. 7:

Tabella 7: Dati tecnici a catalogo.

	LBR iiwa 14 R820
Numero di assi	7
Numero di assi controllati	7
Volume dello spazio di lavoro	1,8 m
Ripetibilità	0,15 mm
Peso	≈ 29,9 Kg
Carico nominale	14 Kg
Portata massima	820 mm
Grado di protezione	IP 54
Livello di rumore	<75 dB
Posizione di Montaggio	Pavimento

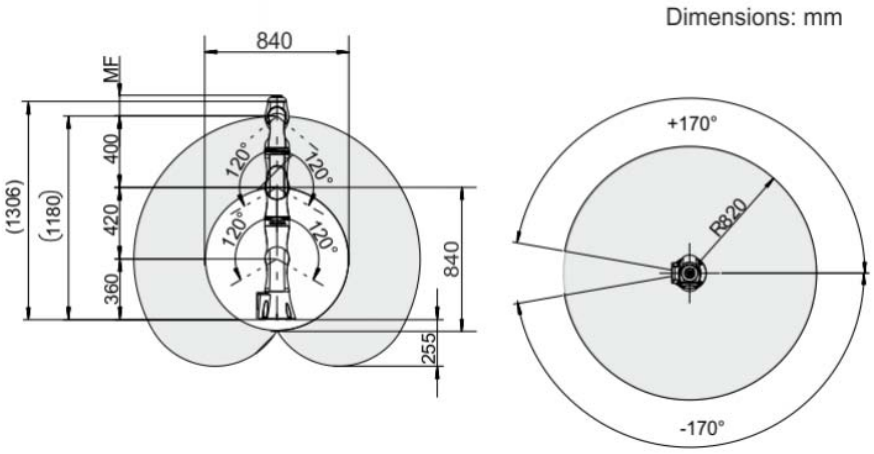


Figura 23: Spazio di lavoro e dimensioni del robot KUKA LBR iiwa 14 r820 [8].

Ad essi è possibile aggiungere delle considerazioni riguardanti i limiti fisico-strutturali del robot, ovvero i dati relativi al range di movimento, alla coppia e velocità massima di ognuno dei sette giunti riportati invece in Tab. 8, i quali possono essere utilizzati anche per ricavare lo spazio di lavoro del robot riportato in Fig. 23.
Date le sue caratteristiche fisiche e in particolare vista la presenza di 7 gradi di libertà con conseguente ridondanza cinematica , il robot ha la possibilità di effettuare un movimento in uno spazio nullo. Questo consiste nel tener fisso l'organo terminale ma contemporaneamente variare la configurazione degli assi del robot, ad esempio modificando la posizione del "gomito" del robot. Osserviamo inoltre a partire

Tabella 8: Caratteristiche fisico-strutturali del robot.

N. asse	Range di movimento [°]	Coppia massima Nm	Velocità massima °/s
1	±170	320	85
2	±170	320	85
3	±170	176	100
4	±170	176	75
5	±170	110	130
6	±170	40	135
7	±170	40	135

dalla Tab. 7 che questo manipolatore come indica il nome stesso attraverso l'acronimo LBR è piccolo e leggero. Questa è una delle caratteristiche di cui si era parlato nel capitolo 2 e che differenziano i cobot dai robot tradizionali. Il LBR iiwa 14 820R infatti oltre ad esser capace di svolgere in modo autonomo attivatà semplici e ripetitive è stato sviluppato prima di tutto per implementare sistemi collaborativi. Per questo motivo esso è stato progettato in modo da poter rispettare le normative di sicurezza per robot industriali (ISO 10218-1 e ISO 10218-2) ma anche la normativa ISO/TS 15066 relativa al funzionamento dei cobot. Quest'ultima è quella di maggior interesse ai fini dello studio successivo che verrà effettuato, in quanto contiene molti concetti che son stati valutati per la realizzazione del software utilizzato nelle prove sperimentali. In particolare all'interno della normativa vengono definiti i tipi di interazioni in ambito collaborativo:

- *Safety-rated monitored stop*: in questo caso il robot si ferma e rimane in tale posizione se un operatore entra all'interno dello spazio di lavoro. Nel caso che verrà preso in esame questa tipologia di interazione non viene considerata, ma può risultare utile in applicazioni dove l'operatore deve per esempio modificare il tipo di end-effector utilizzato.

- *Speed and separation monitoring*: questa tipologia di collaborazione prevede che durante l'interazione il robot e l'operatore mantengano una distanza sufficiente in modo da evitare ogni possibile collisione. Questo risultato può essere raggiunto se viene effettuato un continuo monitoraggio della distanza tra le due risorse, e in caso venga violata la condizione di velocità massima o distanza minima il moto deve essere immediatamente fermato.

- *Hand Guiding*: in questo caso l'operatore prende il diretto controllo del movimento del robot attraverso dei comandi manuali o con un contatto diretto. Questa situazione prevede la condi-

visione dello spazio di lavoro durante tutto il tempo di collaborazione e per ottenere questo risultato è necessario che vi sia un pulsante dedicato per l'avvio di tale modalità e un altro pulsante con cui effettuare uno stop di emergenza in caso di pericolo.

- *Power and force limiting*: questo tipo di collaborazione prevede che l'operatore e il robot condividano lo spazio di lavoro, rischiando così di avere un contatto fisico diretto e non volontario. La normativa suddivide quest'ultimi in due categorie:
  - contatto transitorio: è un impatto dinamico di breve durata (al massimo 50ms), la cui pericolosità è misurata a partire dall'energia trasmessa da robot a uomo. Essa può essere calcolata a partire dalla velocità relativa, dalla massa del robot in movimento e dalla grandezza dell'area di contatto; il valore risultante deve rimanere al di sotto di un valore limite dipendente dalla parte del corpo con cui avviene il contatto.
  - contatto quasi statico: ha durata maggiore rispetto al contatto transitorio, in cui il robot ha la possibilità di ridurre velocità e forza. La parte dell'operatore che viene a contatto con il robot di solito rimane schiacciata tra il robot stesso e un'altra superficie, per questo il pericolo viene misurato sulla base della pressione che viene esercitata sulla persona.

  Per garantire la sicurezza dell'operatore è possibile intervenire su alcune caratteristiche fisiche del robot, andando a ridurre la massa effettiva o aumentando l'area di contatto con soluzione ergonomiche diverse. È possibile agire anche a livello di controllo diminuendo le velocità, coppie e forze massime in gioco. In entrambi i casi è obbligatorio implementare delle funzioni di controllo per la sicurezza che in accordo con la normativa ISO 13849-1 garantiscano lo stop del moto nel caso vengano superati i limiti indicati.

Tra le principali caratteristiche del *LBR iiwa R820* troviamo la presenza di sensori di coppia su ognuno dei sette giunti, i quali permettono di riconoscere subito i contatti e ridurre immediatamente forza e velocità, oltre a rendere il robot capace di maneggiare componenti piccoli e delicati. È per questo motivo che le funzioni collaborative che sono state implementate e che verranno descritte nella sezione successiva sono la *"Power force limiting"* e in parte anche l'*"Hand guiding"*.
La programmazione del robot è avvenuta tramite il software *KUKA Sunrise Workbench*, ovvero un ambiente di sviluppo Java che si interfaccia con il controllore del robot denominato *KUKA Sunrise Cabinet*. La configurazione finale del setup sperimentale è riportata in Fig. 22, dalla quale è possibile notare che oltre agli elementi appena descritti

troviamo il *KUKA smartPAD* ovvero un panello di controllo portatile per il robot industriale. Esso ha diverse funzionalità tra le quali troviamo ad esempio: avviare i vari programmi caricati all'interno del controllore, muovere manualmente il robot, ricavare informazioni sulla posizione del terminale, visualizzare informazioni su velocità e coppie durante il movimento del robot e fermare il robot in situazioni di emergenza.

6.1.2 *Funzioni collaborative implementate*

Per utilizzare il robot all'interno di un layout collaborativo è stata sviluppata in Java una classe denominata `cobot`. All'interno di essa sono state definite usa serie di funzioni per implementare le funzionalità di cui si era gia discusso: *Hand guiding* e *Power and force limiting*.
Per far questo sono state sfruttate alcune caratteristiche del *LBR iiwa R820*, ovvero:

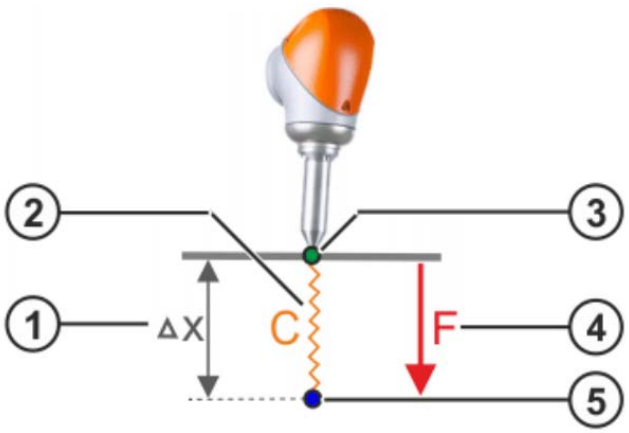


Figura 24: Modello del robot con controllo di impedenza attivo [8].

- le *break conditions* o condizioni di stop, le quali causano l'immediato arresto del moto se si verifica una determinata situazione. Vengono attivate attraverso il metodo `breakWhen` che deve essere richiamato durante un comando di motion.
- Il metodo `positionHold`, il quale obbliga il robot a mantenere una determinata posizione per il tempo specificato o fino al verificarsi di un determinato evento.
- L'*impedance mode control* o modalità di controllo ad impedenza, la quale differisce dal classico controllo di posizione. Infatti, quest'ultimo permette al robot di eseguire un specifico percorso con la maggior accuratezza e senza deviazioni. In aggiunta a questo, il moto non è mai influenzato da fattori esterni quali ostacoli o forze applicate sul manipolatore. Nel caso di un controllo di impedenza il comportamento del robot è definito

compliant e diversamente da prima esso è sensibile e può reagire a fenomeni esterni: l'applicazione di una forza può causare la deviazione del moto rispetto al percorso pianificato.
Il modello del robot che può essere utilizzato per studiare il comportamento del robot in queste condizione è rappresentato in Fig. 24, dove:

1. è la deformazione tra la posizione imposta e quella dove effettivamente si trova il TCP (il punto centrale dell'utensile);
2. è una molla fittizia;
3. la posizione attuale del TCP;
4. la forza generata dai giunti del robot;
5. la posizione desiderata in cui si dovrebbe trovare il TCP;

Come è possibile notare esso è basato su molle e smorzatori virtuali, i quali possono risultare più o meno allungati a causa della differenza tra la posizione attuale del TCP e il punto in cui esso dovrebbe essere. Nel caso in cui queste due corrispondano la molla virtuale risulta allentata: in questa situazione se viene applicata una forza esterna o viene imposto un movimento il robot risulta libero e si sposterà dalla sua posizione iniziale di una quantità Δx. Questo provoca anche un allungamento della molla virtuale e una conseguente forza generata dal robot, in accordo con la legge di Hooke:

$$F = C \cdot \Delta x. \tag{33}$$

Si avrà quindi che sarà più difficile spostare il TCP del robot dalla sua posizione iniziale tanto maggiore è la resistenza imposta dalla molla virtuale. Questa opposizione che si incontra è causata dal robot stesso il quale calcola in automatico la forza F in base alla legge di Hooke e la genera partendo dai valori di coppia ai giunti. Si ricorda infatti che questo è possibile perché in ognuno degli attuatori è presente un sensore di coppia che permette di generare il valore esatto richiesto.
Nel caso invece in cui il robot sia già in movimento con il controllo di impedenza il robot devierà dal percorso stabilito a causa delle forze di attrito interne ai giunti, e tale deviazione sarà maggiore tanto minore è il valore di rigidezza imposto.
È inoltre possibile impostare un controllo di impedenza in più di una direzione cartesiana come in Fig. 25: la forza e la direzione totale sarà calcolata a partire dall'addizione vettoriale delle singole forze nelle diverse direzioni. Un'ulteriore alternativa concessa dal software in dotazione permette di impostare la rigidezza e la viscosità fittizia di ognuno dei sette giunti del robot.

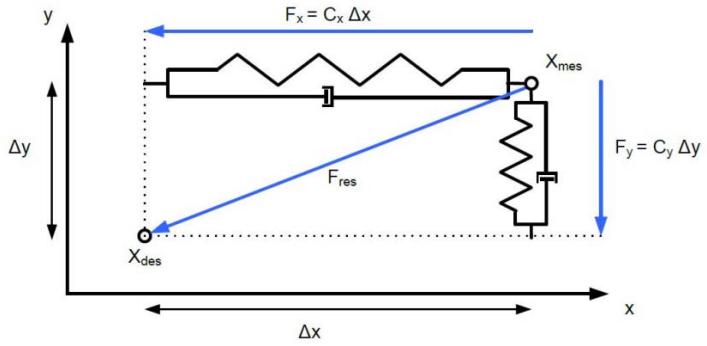


Figura 25: Modello del robot con controllo di impedenza attivo su più direzioni [8].

Ora è possibile introdurre i metodi collaborativi implementati:

- `waitTouch` è una funzione che permette di far entrare il robot in uno stato di attesa; se entro un determinato limite di tempo, che può esser settato quando il metodo viene chiamato, il robot viene toccato esso si riattiva e inizia ad eseguire l'istruzione successiva del programma. Questo risultato è stato raggiunto nel modo seguente: viene continuamente misurata attraverso un ciclo while l'entità delle forze esterne a cui sono sottoposti i giunti 3 e 4 del cobot e quando si registrano valori superiori una determinata soglia o viene superato il tempo limite si esce dal loop permettendo al cobot di proseguire con l'esecuzione del codice.

- `ManualMode` che permette di implementare un'iterazione uomo-robot simile a quella che precedentememte era stata definita come *Hand guiding*: in questo caso però non è presente il pulsante di enabling come invece è richiesto dalla normativa.
  Questo metodo è stato implementato nel modo seguente:
  - viene imposto al robot di mantenere la posizione in cui si trova tramite il metodo `PositionHold` con il controllore in modalità di impedenza.
  - i parametri relativi alla rigidezza e viscosità fittizia dei giunti sono stati imposti in modo da far si che il robot sia libero di essere mosso dall'operatore senza particolari difficoltà.
  - questo movimento nei primi istanti (2 secondi) è completamente libero, mentre successivamente viene vincolato ad una condizione tramite `breakWhen`: se la forza misurata dal robot va al di sotto di una determinata soglia il movimento con controllo di impedenza termina. Questo procedimento serve per far si che quando l'operatore lascia il robot que-

sto si fermi istantaneamente nella posizione in cui è stato rilasciato

  - viene quindi nuovamente detto al robot di mantenere la posizione attuale, questa volta con un controllo di posizione per far si che non possa più esser mosso.

- Infine è stato definito il metodo `move` e i sui derivati ovvero `moveCart`, `appro` e `depart`, i quali permettono di eseguire le operazioni collaborative classificate a normativa all'interno della categoria *"Power and force limiting"*. Con questo metodo infatti il movimento è vincolato ad una condizione di arresto (implementata sempre con il `breakWhen`) che viene attivata quando il robot misura valori di forza troppo elevati, ovvero quando avviene una collisione con l'operatore o un oggetto esterno. Se si verifica questa situazione si attiva una routine la quale fa entrare il robot in modalità guida manuale attraverso il metodo `ManualMode`. Questa scelta è stata fatta per rendere il robot libero appena è avvenuta la collisione e permettere all'operatore di spostarlo in caso si presenti una situazione di pericolo. In seguito, il robot si porta in una condizione di attesa attraverso la funzione `waitTouch` e quando viene toccato riprende ad eseguire il movimento che stava facendo prima di esser bloccato.

Durante le prove di simulazione tutti questi metodi sono stati utilizzati direttamente o indirettamente; è stato quindi necessario definire i valori dei limiti che vengono interrogati quando avvengono le condizioni di arresto. In particolare si presentano tre situazioni in cui è necessario utilizzare il metodo `breakWhen`:

- nel caso di collisioni, dove il valore limite è stato imposto a 20N lungo tutte le direzioni. Questa condizione è più stringente rispetto a quella calcolata sulla base delle richieste della normativa, ma è stata comunque ritenuta accettabile sulla base delle prove effettuate. Per rispettare i parametri imposti dalla normativa è stato inoltre limitato il valore della velocità massima a $250 \quad mm/s$;

- quando si utilizza la guida manuale. In questo caso è stato valutato sulla base di osservazioni sperimentali che il limite ideali al di sotto del quale deve andare la forza misurata affinché il robot si fermi garantendo un'esperienza il miglior possibile all'operatore è di 5N.

- quando si utilizza il `waitTouch` il robot viene risvegliato quando la forza impressa è superiore a 5N. Anche in questo caso il valore è stato determinato sperimentalmente: l'obiettivo era quello di evitare risvegli casuali, cercando comunque di riattivare il robot con un semplice tocco.

Infine in Tab. 9 troviamo riportati i valori di rigidezza torsionale utilizzati per la modalità di impedenza attivata durante la guida manuale. Possiamo osservare che è stato definito un valore unitario per il primo e l'ultimo giunto in modo da facilitare sia la rotazione del robot su se stesso, sia quella del tool. Negli altri casi i valori sono stati trovati sperimentalmente con l'obiettivo di rendere il movimento il più fluido possibile, evitando però che il robot si muovesse a causa del suo stesso peso.

Tabella 9: Impedenze fittizie ai giunti del robot.

N. asse	Impedenza fittizia
1	1 [Nm/rad]
2	7 [Nm/rad]
3	7 [Nm/rad]
4	7 [Nm/rad]
5	7 [Nm/rad]
6	6 [Nm/rad]
7	1 [Nm/rad]

6.2 DESCRIZIONE DELLE PROVE

Nell'esperienza di laboratorio sono stati effettuati dei test per validare i risultati simulativi e confermarli in un scenario reale, dove potrebbero presentarsi diversi effetti fino ad ora non considerati.
Per tali prove si è deciso di riprodurre parte del processo di assemblaggio di un case per un modulo di potenza. Si è deciso di non replicare il processo nella sua interezza ma di limitare il numero di task in modo da garantire una riproduzione dell'assemblaggio più snella a ripetibile. I task considerati sono riportati in Tab. 10 e sostanzialmente si dividono in due tipo:

- picking and placing delle superfici che formano i bordi della scatola, ovvero la base, i quattro lati e la copertura.
- picking e placing dei supporti necessari per l'aggancio del modulo di potenza.

Essi sono stati scelti in modo da poter effettuare dei test che permettano di variare il numero di task n e il diagramma delle precedenze G considerando sempre un box per un modulo di potenza. Le varie combinazioni valutate sono le seguenti:

- nel primo caso il processo produttivo implica che la base del box debba esser posizionata per prima; in seguito è possibile

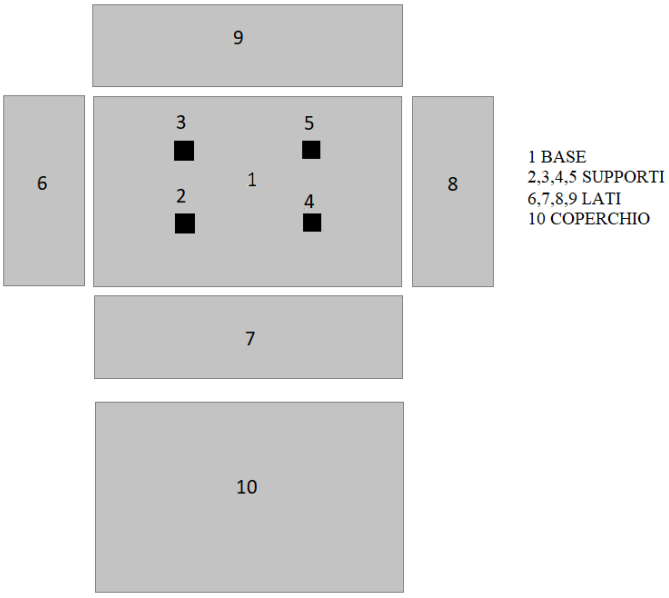


Figura 26: Illustrazione elemento da assemblare e relativi task.

Tabella 10: Descrizione dei task utilizzati.

Numero del task	Descrizione
1	pick and placing della base della scatola
2	pick and placing di uno dei supporti
3	pick and placing di uno dei supporti
4	pick and placing di uno dei supporti
5	pick and placing di uno dei supporti
6	pick and placing del primo lato della scatola
7	pick and placing del secondo lato della scatola
8	pick and placing del terzo lato della scatola
9	pick and placing del quarto lato della scatola
10	pick and placing del coperchio della scatola

posizionare uno dei supporti oppure il primo lato del contenitore. Dalla Fig. 27 osserviamo inoltre che i lati devono obbligatoriamente essere applicati in modo sequenziale, mentre l'ultimo task è sempre il placing del coperchio del box.
Sono riportati in figura due casi di diagramma delle precedenze i quali sono stati realizzati per il caso $n = 10$ e $n = 7$; quest'ultimo prevede tre task in meno in quanto si suppone che uno dei supporti non sia necessario, mentre due lati prevedono un componente con una serie di aperture, diverso da quello utilizzato fino ad ora.
Infine, si osserva che queste configurazioni permettono di avere un indice di parallelizzazione pari a 0.42 nel caso con 7 task, e 0.48 nell'altro.

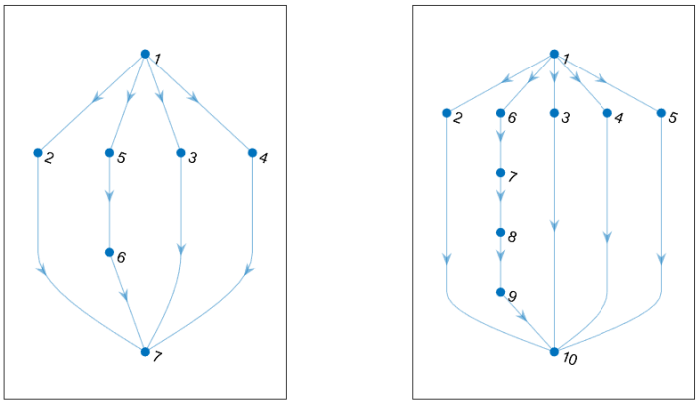


Figura 27: Diagrammi delle precedenze con $n = 7$ e $p_{\%} = 0.42$, $n = 10$ e $p_{\%} = 0.48$.

- nel secondo caso il processo produttivo richiede che come prima inizialmente venga posizionata la base del box; in seguito devono essere eseguiti i task di pick and place dei supporti e solo dopo aver finito questi può iniziare l'assemblaggio delle pareti della scatola. In Fig. 28 sono rappresentati i diagrammi delle precedenze nei due casi valutati al variare di n per i quali valgono le considerazioni fatte precedentemente. Si osserva che i valori di $p_{\%}$ in queste prova sono più bassi, ovvero 0.13 e 0.14.

Per quanto riguarda i tempi dei task, essi sono stati scelti riprendendo valori noti in letteratura. In seguito per valutare l'effetto del $t_{\%}$, sono stati variati: si è scelto di mantenere costanti quelli relativi all'operatore, cambiando invece i tempi di esecuzione del robot. Si è infatti supposto che in determinate situazioni possa esser necessario diminuire la velocità del robot, con il conseguente aumento della durata dei task. Si osserva inoltre che i tempi sono riferiti all'azione congiunta di pick and placing in ognuno dei task considerati.

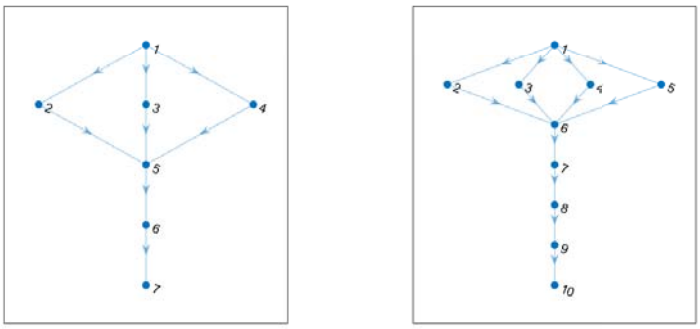


Figura 28: Diagrammi delle precedenze con $n = 7$ e $p_{\%} = 0.14$, $n = 10$ e $p_{\%} = 0.13$.

In Tab. 11, sono riportati i valori impegati durante le simulazioni e utilizzati come riferimento anche per quelle sperimentali.

Tabella 11: Tempi dei task.

Numero del task	Tempo di esecuzione operatore	Tempo di esecuzione robot $t_{\%} = 0.48$	Tempo di esecuzione robot $t_{\%} = 0.61$
1	1	3	3
2	3	6	5
3	3	6	5
4	3	6	5
5	3	6	5
6	7	14	10
7	7	14	10
8	7	14	10
9	7	14	10
10	10	22	20

Ricapitolando il range considerato delle variabili d'ingresso è ristretto e non contempla tutti i casi di simulazione effettuati per la valutazione del modello ma combina i seguenti ingressi:

- $n = 7$ e $n = 10$;
- $p_{\%} \approx 0.48$ e $p_{\%} \approx 0.13$;
- $t_{\%} \approx 0.91$ e $t_{\%} \approx 0.13$.

Durante i test sono stati forniti all'operatore solo la schedulazione dei task ottenuta nelle varie prove, mentre per il robot è stato sviluppato

un programma che utilizza i metodi collaborativi di cui si è parlato precedentemente. Nel codice allegato in Appendice B è possibile vedere che si è deciso di far eseguire il robot il task appropriato solo dopo aver ricevuto un comando dall'operatore: infatti inizialmente il cobot entra in uno stato di attesa grazie al metodo `waitTouch`, e inizia il task solo dopo esser stato toccato dall'operatore; alla fine dell'esecuzione il robot ritorna nella posizione di base se non ha altri task in coda, altrimenti ritorna in uno stato di attesa.
Non avendo a diposizione l'elemento da assemblare, durante tutti i test l'esperienza è stata riprodotta nel modo più accurato possibile utilizzando dei componenti presenti in laboratorio. Inoltre il *KUKA LBR iiwa* coinvolto nella prova era sprovvisto di tool, motivo per il quale è stato necessario simulare i task da esso compiuti: il robot partendo dalla posizione di picking doveva portarsi nel punto di placing, rimanere fermo per un determinato intervallo Δt per poi ritornare indietro al punto di partenza. La lunghezza della pausa realizzata dal robot è stato calcolata analiticamente considerando il layout dell'esperienza rappresentato in Fig. 29: dato il valore di velocità, che per tutta l'esperienza è stato imposto a $250 mm/s$, è stato calcolato il tempo necessario al robot per spostarsi dalla posizione di picking alle varie posizioni di placing:

$$t_{move} = \frac{d}{v}. \tag{34}$$

Dato questo valore e conoscendo la durata totale dei task è stato poi possibile calcolare la lunghezza della pausa:

$$\Delta t = t_{task} - t_{move}. \tag{35}$$

Infine, per rendere l'esperienza più veritiera ogni singola prova sperimentale è stata eseguita per un minimo di sei volte; i risultati riportati nella sezione sono quindi frutto della media dei valori misurati in ogni singolo test.

6.3 ELABORAZIONE DATI E CONFRONTO CON MODELLO SIMULATIVO

In questa sezione vengono presentati i dati risultanti dalle prove sperimentali effettuate, i quali successivamente verranno confrontati con quelli ricavati dalle simulazioni. Al fine di riportare le risultanze dell'esperimento nella maniera più accurata possibile, ogni test effettuato è stata ripreso e successivamente analizzato, questo per garantire la precisione dei dati rilevati. In particolare, i tempi di ogni task sono stati aqcuisiti osservando con attenzione i movimenti in sincrono del robot e dell'operatore, analizzando il filmato in slow motion al fine di garantire la precisione al decimo di secondo. In seguito, sono poi stati estratti sperimentalmente i makespan di ogni esperimento, acquisendo i tempi di inizio e fine del processo di assemblaggio. Infine,

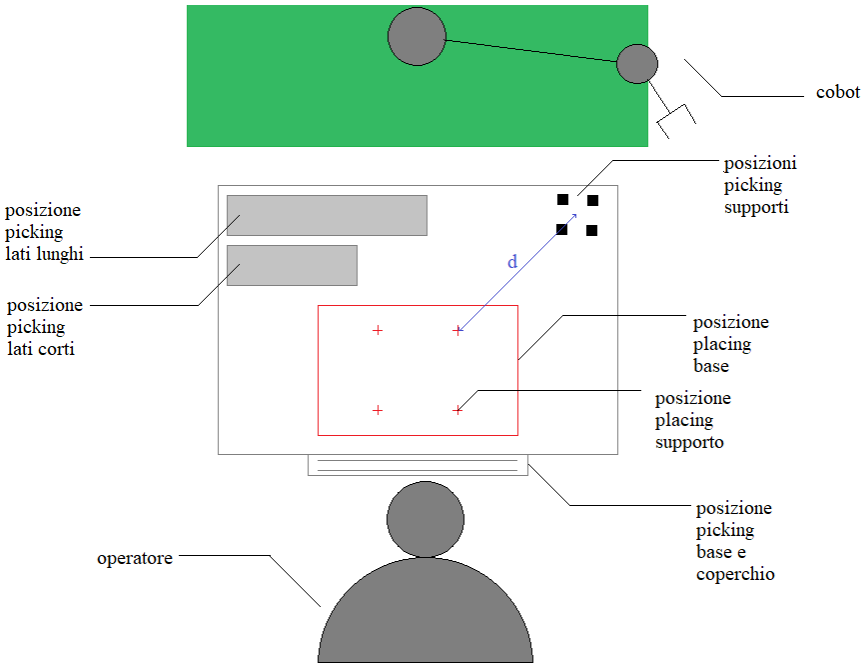


Figura 29: Layout della prova sperimentale.

è stato ricavato il tempo di collaborazione, ovvero gli istanti in cui il robot e l'operatore eseguivano operazioni simultaneamente occupando nello stesso momento lo spazio di lavoro. Queste informazioni sono state poi riportate all'interno del software Matlab, in maniera da poter analizzare ed elaborare i dati per ricavarne gli indici utilizzati per la valorizzazione del modello (Indice sul makespan ed Indice di collaborazione). I dati inseriti, prima del calcolo degli indici, sono stati mediati sul numero di prove effettuate per ogni set di dati, al fine ottenere delle risultanze più accurata e veritiere possibili. Per calcolare gli indici di valorizzazione del modello sono state effettuate le seguenti operazioni:

- Indice sul makespan: è stato determinato per ogni prova effettuata il valore del makespan ottimo ottenuto con parallelizzazione nulla. Successivamente, è stato considerato il valore di makespan delle singole prove e diviso per il corrispondente makespan ottimo precedentemente calcolato.

$$m_{\%} = \frac{makespan[s]}{\min\{makespan^{*}_{|p_{\%}=0}\}[s]}. \tag{36}$$

- Indice di collaborazione: per calcolarlo, sono stati messi in relazione il tempo di collaborazione ed il tempo makespan di ogni singola prova effettuata.

$$c_{\%} = \frac{T_{coll}[s]}{makespan[s]}, \quad c_{\%} \in [0, 1]. \tag{37}$$

Una volta calcolati gli indici basati sui dati ricavati dai test effettuati, si è provveduto a determinare gli stessi indici a partire dai risultati ottenuti dalle prove di simulazione. L'operazione è stata effettuata in maniera da confrontare i risultati ottenuti e trarre le conclusioni dell'esperimento effettuato. Le valutazioni delle risultanze sopra

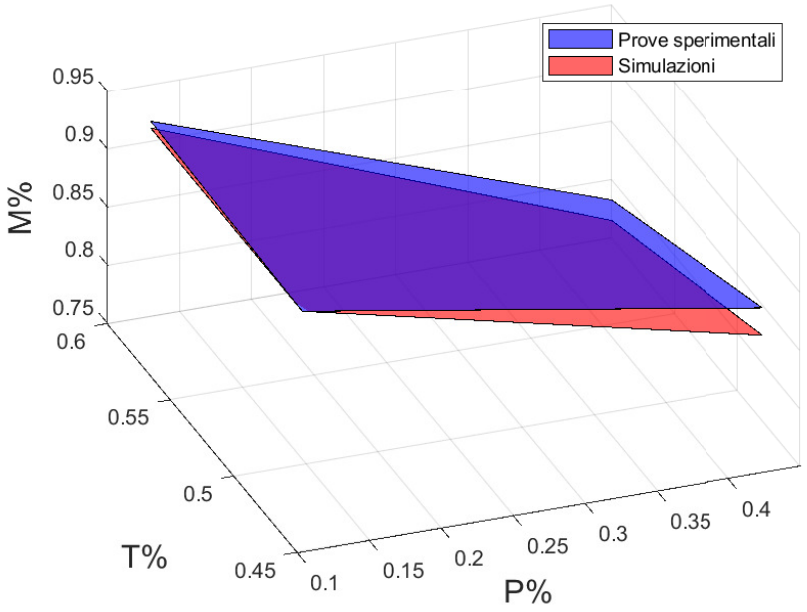


Figura 30: Makespan: confronto tra le prove simulative e sperimentali con $n = 7$.

riportate sono state analizzate al variare dell'indice di parallelizzazione e dell'indice sui tempi dei task. Questo perché valutare i risultati dell'esperimento non considerando gli indici sopra citati sarebbe stato fuorviante da un punto di vista teorico dell'esperimento. Dalle risultanze ottenute, si possono evincere le seguenti considerazioni:

- All'aumentare dell'indice di parallelizzazione il valore del makespan disunisce e come supposto durante le simulazioni, non è mai minore del valore limite definito da $1/m$. Questo accade perché nelle prove effettuate con $p_{\%}$ dello 0,48, il diagramma delle precedenze, visibile in Figura 27, permette di ottenere una schedulazione più efficiente in cui robot e operatore eseguono contemporaneamente il maggior numero di task. Si noti inoltre come nelle prove sperimentali i valori di $m_{\%}$ siano maggiori rispetto a quelli simulativi. Tale risultanza più essere giustificata dal fatto che nelle prove reali non è possibile mantenere i tempi utilizzati durante le simulazioni. Inoltre, è stata calcolata la percentuale di errore tra i dati sperimentali e i dati simulativi, la quale risulta nel caso di makespan pari a $1,91\%$ con deviazione standard di $3,27\%$.

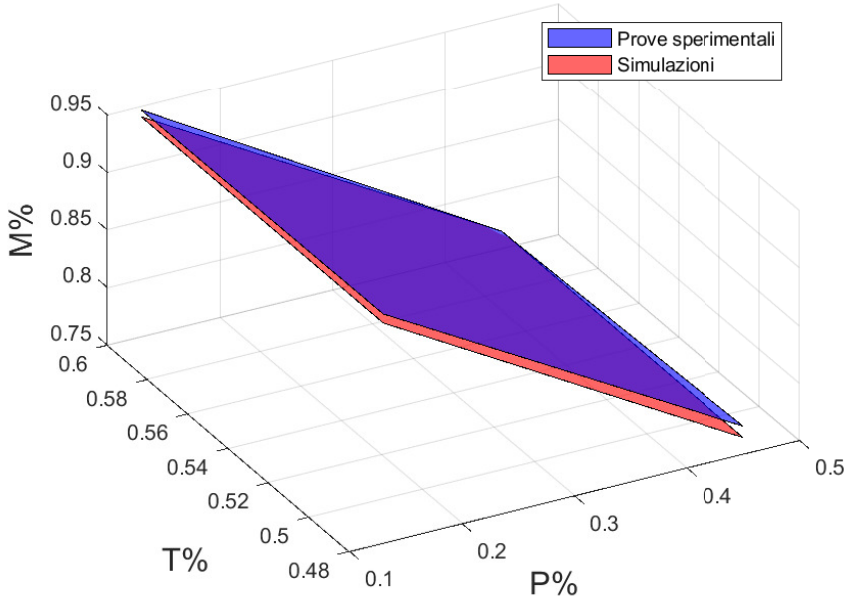


Figura 31: Makespan: confronto tra le prove simulative e sperimentali con $n = 10$.

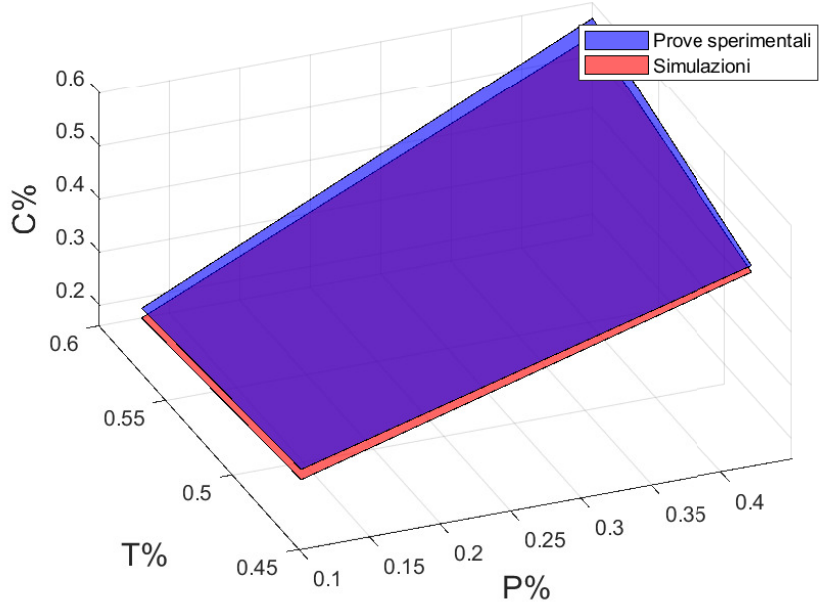


Figura 32: Collaborazione: confronto tra le prove simulative e sperimentali con $n = 7$.

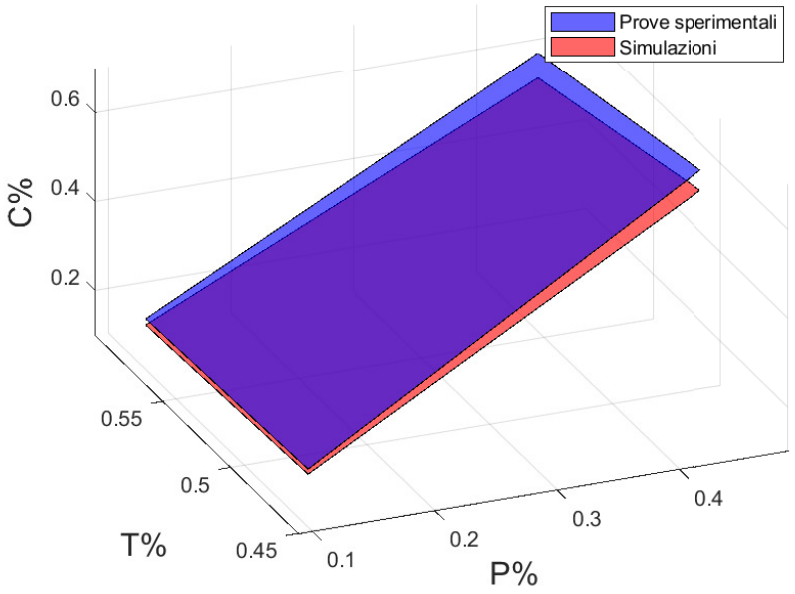


Figura 33: Collaborazione: confronto tra le prove simulative e sperimentali con $n = 10$.

- Relativamente all'indice di collaborazione, durante l'analisi delle riprese delle prove sperimentali con un $p_{\%}$ basso, è stato notato che il robot commetteva un errore nell'esecuzione del task che gli veniva assegnato. Infatti, prendendo come esempio la Fig. 34, il robot invece di eseguire tale task in 6 secondi, lo ultimava con un tempo maggiore. Questo fatto può essere spiegato ricordando quanto detto precedentemente riguardo alla programmazione del robot stesso, e in particolare relativamente all'esecuzione di una pausa all'interno del task. Infatti, il robot era stato programmato per raggiungere una velocità di 250 mm/s, ma è possibile che, data la scarsa lunghezza del movimento che doveva effettuare, il robot non riuscisse a raggiungere tale picco. Inoltre, è possibile che la stessa distanza non sia stata stimata in maniera accurata. L'errore, nella maggior parte dei task, è quantificabile in un valore compreso fra 0,5 e 0,7 secondi. Questo fatto influenza di conseguenza anche i valori dell'indice di collaborazione misurati, nello specifico questi risulteranno maggiori rispetto a quanto previsto. A supporto di quanto detto, l'errore percentuale tra il valore sperimentale ed il valore di simulazione è maggiore rispetto al caso precedente, assestandosi intorno al $7,08\%$ con deviazione standard di $2,7\%$.

- Si nota inoltre che l'indice di collaborazione aumento all'aumentare della parallelizzazione, in accordo con la contemporanea dimensione dell'indice sul makespan analizzato nei punti

precedenti.

In conclusione, osservando le Fig. 30, 31, 32 e 33, è possibile affermare che i risultati ottenuti dalle prove sperimentali convalidano il modello utilizzato per l'allocazione ottima dei task. Infatti, come si evince dalle figure, i piani che raffigurano i risultati dei test sperimentali e simulativi, risultano essere fra loro paralleli, indice di una corretta riproduzione del modello teorico.

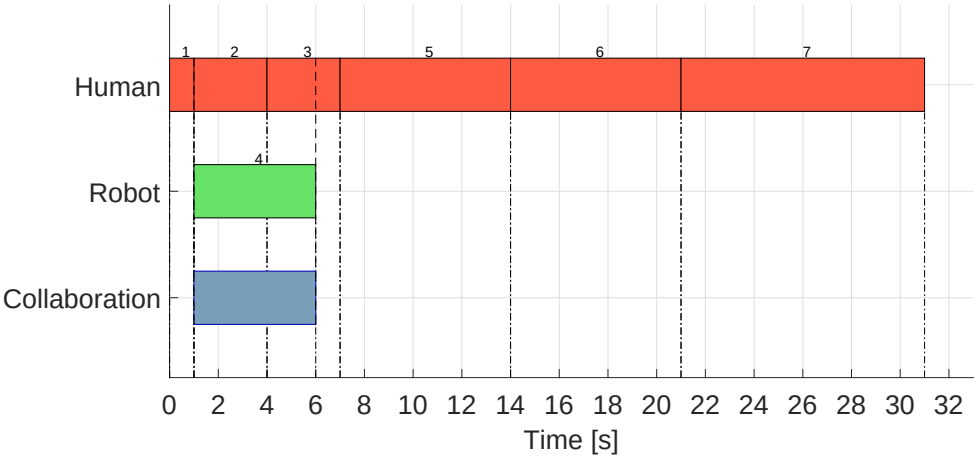


Figura 34: Schedulazione utilizzate per la valutazione dell'errore su collaborazione.

CONCLUSIONI

In questo lavoro di tesi si è quindi presentato un metodo esatto per l'assegnazione dei task in ambito collaborativo con il fine ultimo di minimizzare il tempo di produzione. In seguito, sono stati introdotti degli indici per caratterizzare gli output del problema, ovvero makespan e tempo di collaborazione, e gli input, quali il numero di task, la loro durata e il diagramma delle precedenze. Utilizzando il software Matlab è stato risolto il problema di allocazione dei task e sono state effettuate una serie di prove al variare degli indici precedentemente introdotti. Si è quindi dimostrato che all'aumentare del grado di parallelizzazione del diagramma delle precedenze relativo ad un determinato processo produttivo, il makespan totale diminuisce mentre aumenta il tempo di collaborazione. Inoltre, si è constatato che questi due parametri sono strettamente relazionati; in particolare un aumento della collaborazione si incide sui tempi di produzione, provocandone una significativa diminuzione. Infine, si è valutato che nel caso in cui le risorse abbiano tempi di esecuzione dei task simili, l'incidenza del grado di parallelizzazione diventa maggiormente apprezzabile sul makespan, mentre non è stata rilevata alcuna correlazione con il tempo di collaborazione. Queste osservazioni infine sono state convalidate attraverso le prove sperimentali. In questa dissertazione il metodo di risoluzione del problema di allocazione utilizzato, è basato su un algoritmo branch and bound. Non sono state analizzate altre alternative quali metodi euristici o algoritmi genetici perché non pertinenti con l'obiettivo della tesi. In uno sviluppo futuro però questo soluzioni potrebbero essere prese in considerazione così da permettere un'analisi più accurata e veloce su un numero di casistiche superiori. Inoltre, in futuro potrebbe essere interessante introdurre delle funzioni obiettivo diverse, le quali prendano in considerazioni altri aspetti centrali nella collaborazione.

APPENDIX

A

CODICE MATLAB

```
clear all
%% DATA SELECTION
n = 12;                          % task number
m = 2;                           % resource number

T_OP    =   [5  3  9  5  4  5  2  3  10  10  7  2];
T_R     =   [10  3  11  8  6  7  4  6  10  6  6  4];

% min makespan cwith p% = 0
M0 = 0;
 for iterTask = 1:n
    M0 = M0 + min(T_OP(iterTask),T_R(iterTask));
 end

T = M0;                          % time interval
T_min = ceil(M0/2-1);

% precedence diagram edge
EdgeTable = table([  1 2; 2 3; 3 4; 4 5; 5 6; 6 7; 7 8; 8 9; 9
    10; 10 11; 11 12], 'VariableNames', {'EndNodes'});%[ 1 2; 2
    3; 3 4; 4 5; 5 6; 6 7; 7 8; 8 9; 9 10; 10 11; 11 12; 12 13;
    13 14; 14 15; 15 16; 16 17; 17 18; 18 19; 19 20; 20 21; 21
    22; 22 23; 23 24; 24 25]


j = [1:n];  % task index
k = [1:m];  % resource index
t = [1:T];  % time index

% Create precedence diagram
TjkTable = table(transpose(T_OP),transpose(T_R), 'VariableNames',
     {'Operator','Robot'});
G = digraph(EdgeTable, TjkTable);

% Tjk matrix
Tjk = TjkTable.Variables;


%% OPTIMIZATION VARIABLE
% Create a binary optimization variable 3D-Matrix
x_jkt = optimvar('x_jkt', [m n T] , 'Type', 'integer', '
    LowerBound', 0,'UpperBound', 1);

% Creates a optimization variable for mutal exclusion
y_jj =optimvar('y_jj', [m n n] , 'Type', 'integer', 'LowerBound',
     0, 'UpperBound',1);
```

```
% Create a variable to optimize
makespan = optimvar('makespan', 'LowerBound', T_min, 'UpperBound
    ', T);

%% OPTIMIZATION EXPRESSION
startTime = optimexpr();
for iterR = 1:m
    for iterTask = 1:n
       sumPartial = 0;
       for iterTime = 1:T
          sumPartial = sumPartial + iterTime*x_jkt(iterR,iterTask
              ,iterTime);
       end
       startTime(iterR,iterTask) = sumPartial;
    end
end

endTime = optimexpr();
for iterR = 1:m
    for iterTask = 1:n
    sumPartial = 0;
    for iterTime = 1:T
        sumPartial = sumPartial + (iterTime+Tjk(iterTask,iterR))*
            x_jkt(iterR,iterTask,iterTime);
    end
        endTime(iterR,iterTask) = sumPartial;
    end
 end

%% OPTIMIZATION CONSTRAINTS
fprintf("Computing constraints...");
tic
% Each task j must starts exactly at one point t and must be done
     by only one resource
oneTask = sum(sum(x_jkt,3),1) == 1;

% precedence constraints
precedence = optimconstr();
for iterTask = 1:n
    % Right part of inequality, task j
    right = sum(startTime(:,iterTask),1);

    % Left part of inequality, tasks i or rather predecessors of
        j
    left = x_jkt(:,transpose( predecessors(G,iterTask)),:);
        TjkMatrixPredec = Tjk(predecessors(G,iterTask),:);
        totPred = size(TjkMatrixPredec);
        left_R = optimexpr(totPred(1));

        for iterPred = 1 : totPred(1)
            for iterR = 1:m
```

```
                left_T = 0;
                for iterTime = 1:T
                    left_T = left_T+(iterTime+TjkMatrixPredec(
                        iterPred,iterR))*left(iterR,iterPred,
                        iterTime);
                end
               left_R(iterPred) = left_R(iterPred)+left_T;
            end
        end


        % Definition of precedence condition
        numberCond = size(left_R);
        if numberCond(1) <= 0
            precedence(iterTask)= 0<=right;
        else
            for iterPred = 1:numberCond(1)
                precedence(iterTask,iterPred)= left_R(iterPred,1)
                    <=right;
            end
        end

end

M = 200;
% Each recource must do only one task at a time
mutualExclusion = sum(x_jkt,2) <= 1;
mutualExclusion1 = optimconstr();
mutualExclusion2 = optimconstr();
for iterR = 1:m
   for iterTask = 1: n

       % Right
       taskStart = startTime(iterR,iterTask);
       taskEnd = endTime(iterR,iterTask);

       parallelNode = j;
       sequentialNode = [allPredecessors(G,iterTask),iterTask,
           allSuccessors(G,iterTask)];
       parallelNode(sequentialNode)=[];


       % Left and conditions
       for iterTask2 = parallelNode
           if (iterTask2 > iterTask)
               taskCompStart = startTime(iterR,iterTask2);
               taskCompEnd = endTime(iterR,iterTask2);
               mutualExclusion1(iterR,iterTask,iterTask2) =
                   taskCompEnd - M*y_jj(iterR,iterTask,iterTask2)
                    <= taskStart;
               mutualExclusion2(iterR,iterTask,iterTask2) =
                   taskEnd - M*(1-y_jj(iterR,iterTask,iterTask2))
```

```
                    <= taskCompStart;
            end
        end


    end
end

% feasibility constraint
feasibility = optimconstr();
for iterTask = 1: n
    for iterR = 1:m
        if (Cjk(iterR,iterTask)==1)
            if iterR == 1
                feasibility(iterTask) = sum(x_jkt(1,iterTask,:),3)
                    ==1;
            else
                feasibility(iterTask) = sum(x_jkt(2,iterTask,:),3)
                    ==1;
            end
        end
    end
end

% makespan constraints
makespanCond= optimconstr(n);
for iterTask = 1:n
    makespanR=0;
    for iterR = 1:m
        conMakespan = 0;
        for iterTime = 1:T
            conMakespan = conMakespan+(iterTime + Tjk(iterTask,
                iterR))*x_jkt(iterR,iterTask,iterTime);
        end
        makespanR = makespanR+conMakespan;
    end
    makespanCond(iterTask) =makespan >= makespanR;
end

ConstraintsElapsedTime = toc
%% OPTIMIZATION PROBLEM
% Create an optimization problem container. Include the objective
     function in the problem
optimizationproblem = optimproblem('ObjectiveSense','min','
    Objective',makespan);

optimizationproblem.Constraints.mutualExclusion1 =
    mutualExclusion1;
optimizationproblem.Constraints.mutualExclusion2 =
    mutualExclusion2;
optimizationproblem.Constraints.mutualExclusion = mutualExclusion
    ;
```

```
optimizationproblem.Constraints.precedence = precedence;
optimizationproblem.Constraints.oneTask = oneTask;
optimizationproblem.Constraints.feasibility = feasibility;
optimizationproblem.Constraints.makespanCond = makespanCond;

%% PROB2STRUCT
problem = prob2struct(optimizationproblem);

%% CPLEX OPTIONTS
cplexOpts = cplexoptimset('cplex');
cplexOpts.display = 'on';
cplexOpts.mip.strategy.startalgorithm = 1;
cplexOpts.mip.tolerances.objdifference = 1;
cplexOpts.mip.tolerances.mipgap = 5;
cplexOpts.mip.display = 5;
cplexOpts.mip.strategy.probe = 3;
cplexOpts.preprocessing.symmetry = 5;
cplexOpts.timelimit = 3600;

% string that tells the solver which variables are integer and
    which binary
ctype = 'I';
for iterString = 1: length(problem.f)-1
    ctype = strcat(ctype,'B');
end

%% EXECUTION
tic
[x, fval, exitflag, output] = cplexmilp (problem.f, problem.Aineq
    , problem.bineq, problem.Aeq, problem.beq,...
  [ ], [ ], [ ], problem.lb, problem.ub, ctype, [ ], cplexOpts);
ElapsedTime = toc
```

B

CODICE PROVA SPERIMENTALE

```
import javax.inject.Inject;

import com.kuka.common.ThreadUtil;
import com.kuka.roboticsAPI.applicationModel.
    RoboticsAPIApplication;
import static com.kuka.roboticsAPI.motionModel.BasicMotions.*;

import com.kuka.roboticsAPI.conditionModel.ForceCondition;
import com.kuka.roboticsAPI.deviceModel.LBR;
import com.kuka.roboticsAPI.geometricModel.Frame;
import com.kuka.roboticsAPI.geometricModel.math.CoordinateAxis;
import com.kuka.roboticsAPI.motionModel.controlModeModel.
    CartesianImpedanceControlMode;
import com.kuka.roboticsAPI.motionModel.controlModeModel.
    JointImpedanceControlMode;
import com.kuka.roboticsAPI.motionModel.controlModeModel.
    PositionControlMode;

public class Test_lab extends RoboticsAPIApplication {
        @Inject
        private LBR lBR_iiwa;
        private Cobot cobot;
        private PositionControlMode posMode;
        private JointImpedanceControlMode impMode;
        private CartesianImpedanceControlMode cartImpMode;

        private ForceCondition stopCondSafety;
        private ForceCondition stopCondManual;
        private ForceCondition stopCondSafetyAppro;
        @Override
        public void initialize() {
                // initialize your application here
                // definizione controllo di Posizione e impedenza
                                posMode = new PositionControlMode
                                    ();
                                impMode = new
                                    JointImpedanceControlMode
                                    (1.0,7.0, 7.0, 7.0, 7.0, 6.0,
                                     1.0);
                                cartImpMode = new
                                    CartesianImpedanceControlMode
                                    ();

                                // condizioni di stop di
                                    sicurezza e stop per la guida
                                     manuale
```


[13 luglio 2020 at 8:37 - classicthesis]

```
                            stopCondSafety = ForceCondition.
                                createSpatialForceCondition(
                                lBR_iiwa.getFlange(), 20);
                            stopCondManual = ForceCondition.
                                createSpatialForceCondition(
                                lBR_iiwa.getFlange(), 5);
                            stopCondSafetyAppro =
                                ForceCondition.
                                createNormalForceCondition(
                                lBR_iiwa.getFlange(),
                                CoordinateAxis.X, 5);


                            // inizializzazione classe da
                                utilizzare per effettuare
                                movimenti collaborativi
                            cobot = new Cobot(lBR_iiwa,
                                impMode, cartImpMode, posMode
                                , stopCondSafety,
                                stopCondSafetyAppro,
                                stopCondManual);
    }

    @Override
    public void run() {
            long Ttask = 5;
            Frame uscita = new Frame(412.2,-327.3,431.1,Math.
                toRadians(130.58),Math.toRadians(-2.41),Math.
                toRadians(180));
            Frame t1 = new Frame(-17.42,-623.79,450,Math.
                toRadians(84.42),Math.toRadians(-2.41),Math.
                toRadians(-180));
            Frame pick = new Frame(283.94,-398.3,450,Math.
                toRadians(84.42),Math.toRadians(-2.41),Math.
                toRadians(-180));

            // your application execution starts here
            //lBR_iiwa_14_R820_1.move(ptpHome());
            //lBR_iiwa.move(ptp(Math.toRadians(-42.16),Math.
                toRadians(28.4),0,Math.toRadians(-87.14),0,
                Math.toRadians(66-85),0).setJointVelocityRel
                (0.1));
            cobot.moveCart(pick, 250);
            /*
            //task 2
            cobot.waitTouch(20000,1);
            Frame t2 = new Frame(-83.52,-580.23,400,Math.
                toRadians(84.42),Math.toRadians(-2.41),Math.
                toRadians(-180));
            cobot.moveCart(t2, 250);
            ThreadUtil.milliSleep((Ttask)*1000-3200);
            cobot.moveCart(pick, 250);
```

```
            */
            //task3
            cobot.waitTouch(20000,1);
            Frame t3 = new Frame(55.62,-580.17,400,Math.
                toRadians(84.42),Math.toRadians(-2.41),Math.
                toRadians(-180));
            cobot.moveCart(t3,250);
            ThreadUtil.milliSleep((Ttask)*1000-2400);
            cobot.moveCart(pick, 250);

            /*
            //task4
            cobot.waitTouch(20000, 1);
            Frame t4 = new Frame(-83.52,-674.37,400,Math.
                toRadians(84.42),Math.toRadians(-2.41),Math.
                toRadians(-180));
            cobot.moveCart(t4,250);
            ThreadUtil.milliSleep((Ttask)*1000-3600);
            cobot.moveCart(pick, 250);


            //task5
            cobot.waitTouch(20000, 1);
            Frame t5 = new Frame(55.62,-674.37,400,Math.
                toRadians(84.42),Math.toRadians(-2.41),Math.
                toRadians(-180));
            cobot.moveCart(t5,250);
            ThreadUtil.milliSleep((Ttask)*1000-2800);
            cobot.moveCart(pick, 250);
            */
            // CODICE DI USCITA
            //cobot.departCart(lBR_iiwa.
                getCurrentCartesianPosition(lBR_iiwa.
                getFlange()), 250, 250);
            cobot.moveCart(uscita, 250);
            System.out.println("fine");
        }
```

BIBLIOGRAFIA

[1] Karin Bogner, Ulrich Pferschy, Roland Unterberger, and Herwig Zeiner. Optimised scheduling in human–robot collaboration–a use case in the assembly of printed circuit boards. *International Journal of Production Research*, 56(16):5522–5540, 2018.

[2] F. Chen, K. Sekiyama, F. Cannella, and T. Fukuda. Optimal subtask allocation for human and robot collaboration within hybrid assembly system. *IEEE Transactions on Automation Science and Engineering*, 11(4):1065–1075, 2014.

[3] Maurizio Faccio, Riccardo Minto, Giulio Rosati, and Matteo Bottin. The influence of the product characteristics on human-robot collaboration: a model for the performance of collaborative robotic assembly. *The International Journal of Advanced Manufacturing Technology*, 106(5):2317–2331, 2020.

[4] Manuel Fechter, Carsten Seeber, and Shengjian Chen. Integrated process planning and resource allocation for collaborative robot workplace design. *Procedia CIRP*, 72:39–44, 2018.

[5] L. De Giovanni. Branch-and-bound per problemi di programmazione lineare intera- appunti di ricerca operativa.

[6] IBM. Cplex optimizer - ibm, 2020. URL https://www.ibm.com/it-it/analytics/cplex-optimizer.

[7] L. Johannsmeier and S. Haddadin. A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes. *IEEE Robotics and Automation Letters*, 2(1):41–48, 2017.

[8] KUKA. Lbr iiwa, 2020. URL https://www.kuka.com/it-it.

[9] MathWorks. Optimization toolbox matlab. URL https://it.mathworks.com/products/optimization.html.

[10] George Michalos, Jason Spiliotopoulos, Sotiris Makris, and George Chryssolouris. A method for planning human robot shared tasks. *CIRP journal of manufacturing science and technology*, 22: 76–90, 2018.

[11] Nikolaos Nikolakis, Niki Kousi, George Michalos, and Sotiris Makris. Dynamic scheduling of shared human-robot manufacturing operations. *Procedia CIRP*, 72:9–14, 2018.

[12] Margaret Pearce, Bilge Mutlu, Julie Shah, and Robert Radwin. Optimizing makespan and ergonomics in integrating collaborative robots into manufacturing processes. *IEEE transactions on automation science and engineering*, 15(4):1772–1784, 2018.

[13] Fabian Ranz, Vera Hummel, and Wilfried Sihn. Capability-based task allocation in human-robot collaboration. *Procedia Manufacturing*, 9:182–189, 2017.

[14] Universal Robot. Robot collaborativi o cobot, 2020. URL https://info.universal-robots.com/it/robot-collaborativi-o-cobot-la-guida-definitiva/#definizione_e_storia_dei_cobot.

[15] A. Scholl. *Balancing and Sequencing of Assembly Lines*. Production and logistics. Physica-Verlag, 1995. ISBN 9783790808810. URL https://books.google.it/books?id=nfBTAAAAMAAJ.

[16] J Steven Moore and Arun Garg. The strain index: a proposed method to analyze jobs for risk of distal upper extremity disorders. *American Industrial Hygiene Association Journal*, 56(5): 443–458, 1995.

[17] Shozo Takata and Takeo Hirano. Human and robot allocation method for hybrid assembly systems. *CIRP annals*, 60(1):9–12, 2011.

[18] F Brian Talbot. Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. *Management science*, 28(10):1197–1210, 1982.

[19] Jeffrey Too Chuan Tan, Feng Duan, Ryu Kato, Tamio Arai, and Ernest Hall. *Collaboration planning by task analysis in human-robot collaborative manufacturing system*. INTECH Open Access Publisher, 2010.

[20] Panagiota Tsarouchi, Sotiris Makris, and George Chryssolouris. Human–robot interaction review and challenges on task planning and programming. *International Journal of Computer Integrated Manufacturing*, 29(8):916–931, 2016. doi: 10.1080/0951192X.2015.1130251. URL https://doi.org/10.1080/0951192X.2015.1130251.

[21] Panagiota Tsarouchi, Alexandros-Stereos Matthaiakis, Sotiris Makris, and George Chryssolouris. On a human-robot collaboration in an assembly cell. *International Journal of Computer Integrated Manufacturing*, 30(6):580–589, 2017. doi: 10.1080/0951192X.2016.1187297. URL https://doi.org/10.1080/0951192X.2016.1187297.

[22] Valeria Villani, Fabio Pini, Francesco Leali, and Cristian Secchi. Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications. *Mechatronics*, 55:248 – 266, 2018. ISSN 0957-4158. doi: https://doi.org/10.1016/j.mechatronics.2018.02.009. URL http://www.sciencedirect.com/science/article/pii/S0957415818300321.

[23] Wikipedia. Np-difficile — wikipedia, l'enciclopedia libera, 2020. URL http://it.wikipedia.org/w/index.php?title=NP-difficile&oldid=113588181. [Online; in data 8-luglio-2020].