ALMA MATER STUDIORUM
UNIVERSITA' DI BOLOGNA

Corso di laurea magistrale in
Ingegneria meccanica

Tesi di laurea in

Meccanica delle Macchine

Progettazione ed emulazione virtuale di un sistema di sincronizzazione dinamica per macchina automatica

Relatore
Prof. Marco Carricato

Presentata da
Giovanni Andriani

Correlatori
Ing. Carlo Zanini
Ing. Enrico Pagliaro

Sessione di laurea dicembre 2021

Anno accademico 2020/2021

INDICE

Abstract

Nell'industria delle macchine automatiche è di fondamentale importanza garantire un processo di lavorazione rapido ed efficace; tuttavia, sempre più frequentemente per ragioni economiche ed organizzative è più impellente la richiesta di collegare in una linea macchine già progettate, adattandole entro dei limiti stabiliti.

Se tali macchine sono progettate per elaborare quantità diverse di prodotti per ciclo o se il loro funzionamento è di tipologia diversa (alternato o continuo), una delle strategie per collegare in linea tali macchine è quella di utilizzare un sistema di sincronizzazione dinamica (a volte chiamato sistema sincro-dinamico), il quale è in grado di sincronizzare il flusso di prodotti tra due macchine che per loro natura non lavorano con lo stesso ritmo produttivo. Il sistema analizzato è costituito da due cinghie di movimentazione con dei cassetti vincolati su di esse predisposti per ospitare il prodotto.

In questo elaborato di tesi si affronta la progettazione di un sistema di questa tipologia analizzando il problema in tre parti.

Nella prima parte si elabora un algoritmo in grado di calcolare il numero di cassetti idoneo da essere montato sul sistema, rispettando determinati vincoli di progetto. Il numero di soluzioni ottenuto costituisce un set di configurazioni possibili di macchina.

Nella seconda parte si definiscono delle leggi di moto per ogni configurazione in modo tale da poter garantire la movimentazione dei cassetti nei tempi prestabiliti e nei limiti fisici degli azionamenti.

Infine, nella terza parte si utilizza un software di emulazione virtuale (o virtual commissioning) per la simulazione del processo produttivo, nella quale si studia la prestazione di ogni configurazione implementando le leggi di moto definite in precedenza, ottenendo in automatico la carta delle operazioni macchina e decretando quale tra le configurazioni sia quella ottima. L'utilizzo di un software emulativo non solo permette di ottenere una carta delle operazioni in maniera semplice, ma permette di estendere tale studio ad un numero maggiore di configurazioni in modo rapido ed efficace. La simulazione del processo ha seguito le linee guida del formalismo modulare e gerarchico basato sulla modellazione a stati dove per ciascuno dei sottosistemi considerati vengono evidenziati i diversi stati di funzionamento e le modalità attraverso le quali è possibile muoversi da uno stato all'altro.

1. Introduzione

1.1. Marchesini Group

Questo lavoro di tesi è stato svolto in collaborazione con l'azienda Marchesini Group, attraverso lo svolgimento di un tirocinio per la preparazione della tesi.

L'azienda Marchesini Group nasce in Italia a Pianoro (BO) nel 1974 come piccola azienda familiare e nel corso degli anni è cresciuta fino a divenire una delle maggiori realtà industriali nella meccanica.

L'azienda progetta e produce macchine automatiche per il packaging nell'ambito farmaceutico e cosmetico.

Le tecnologie impiegate dall'azienda spaziano in tutti gli ambiti innovativi dell'ingegneria meccanica ed elettronica e proprio in questo spirito, il lavoro di tesi qui presentato approfondirà sia tematiche tipiche dell'ingegneria meccanica che tematiche provenienti dall'ingegneria dell'automazione.

L'azienda Marchesini utilizza i software PTC Creo e industrialPhysics che vengono utilizzati come strumenti per lo svolgimento di questo elaborato di tesi.

1.2. Industria 4.0 e Digital Twin

Con il termine *industria 4.0* si denota la quarta rivoluzione industriale che rappresenta la trasformazione di una industria di tipo "tradizionale" in una nuova industria attraverso l'utilizzo massiccio della tecnologia *Internet of Things* (IoT) (Figura 1).

Il termine industria 4.0 è utilizzato per incapsulare un cambio di paradigma nell'economia. Questo termine è stato utilizzato per la prima volta nel report dell'Industry-Science Research Alliance Working Group presentato al cancelliere tedesco ed è stato utilizzato anche nella fiera di Hannover del 2013 in cui è stata presentata per la prima volta la piattaforma "Industrie 4.0".

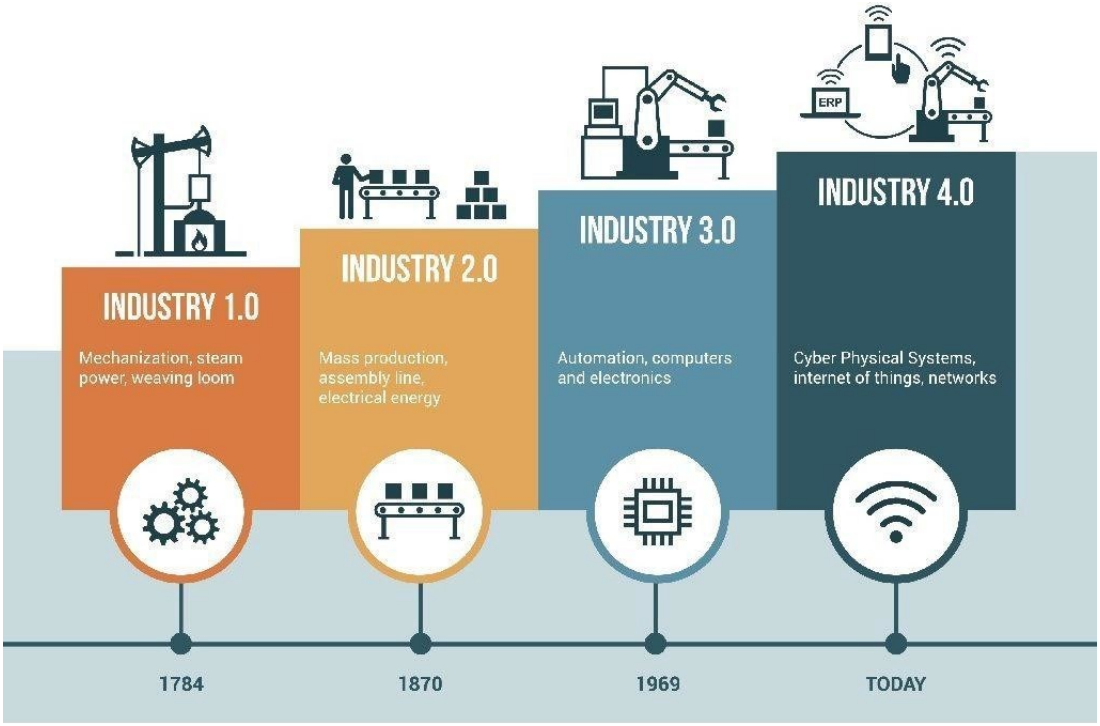


Figura 1 - Le quattro rivoluzioni industriali

Il networking in tempo reale di prodotti, processi e infrastrutture sta inaugurando la quarta rivoluzione industriale in cui fornitura, produzione, manutenzione, consegna e servizio clienti sono tutti collegati tramite internet. [1]

Si iniziano a definire nuovi termini che si accostano a questo nuovo tipo di industria:

- *Smart production*, nuove tecnologie produttive che creano collaborazione tra tutti gli elementi presenti nella produzione, ovvero tra operatore, macchine e strumenti;
- *Smart service*: tutte le "infrastrutture informatiche" e tecniche che permettono di integrare i sistemi tra di loro;
- *Smart energy*: i sistemi diventano più performanti e riducono gli sprechi di energia.

Con l'avvento dell'industria 4.0 si vengono a modificare anche i metodi di utilizzo dei modelli simulativi di un sistema. La simulazione consiste nell'utilizzo di un modello reale o virtuale, che approssima il funzionamento del sistema che si vuole studiare, con il fine di predire o di capire meglio il comportamento del sistema in vari contesti operativi.

Nell'ingegneria, l'uso di modelli simulativi aiuta a ridurre i costi di sviluppo di un prodotto, riduce i tempi di progettazione e facilita il management della conoscenza aziendale. Con l'industria 4.0, un computer centrale coordina lo scambio di informazioni fra tutti i sottosistemi aziendali, anche separati geograficamente, in un unico spazio cyber-fisico. Gli operatori umani possono facilmente interagire con lo spazio cyber-fisico con cui definiscono i requisiti di progetto e le richieste di informazioni, il tutto mentre il processo di management lavora autonomamente. Questa stretta interazione tra il mondo reale e quello virtuale cambia profondamente gli aspetti del processo produttivo aziendale, tra cui anche l'utilizzo ingegneristico della simulazione di un sistema.

Fino ad ora il metodo simulativo prevedeva lo sviluppo di un modello standalone il cui scopo e durata di utilizzo erano limitati. Tuttavia, l'utilizzo della simulazione in diverse aree dei processi aziendali ha portato alla necessità di collegare i modelli di simulazione utilizzati nelle diverse parti di un'organizzazione. Inoltre, la tendenza nei modelli simulativi si è spostata da modelli puramente analitici e orientati all'ottimizzazione all'integrazione di modelli di simulazione in strumenti di supporto decisionale da utilizzare in modo ricorrente. [2]

Il nuovo paradigma di simulazione, che si delinea con l'introduzione dell'industria 4.0, è sintetizzato al meglio con il concetto di "*Digital Twin*" (Figura 2).

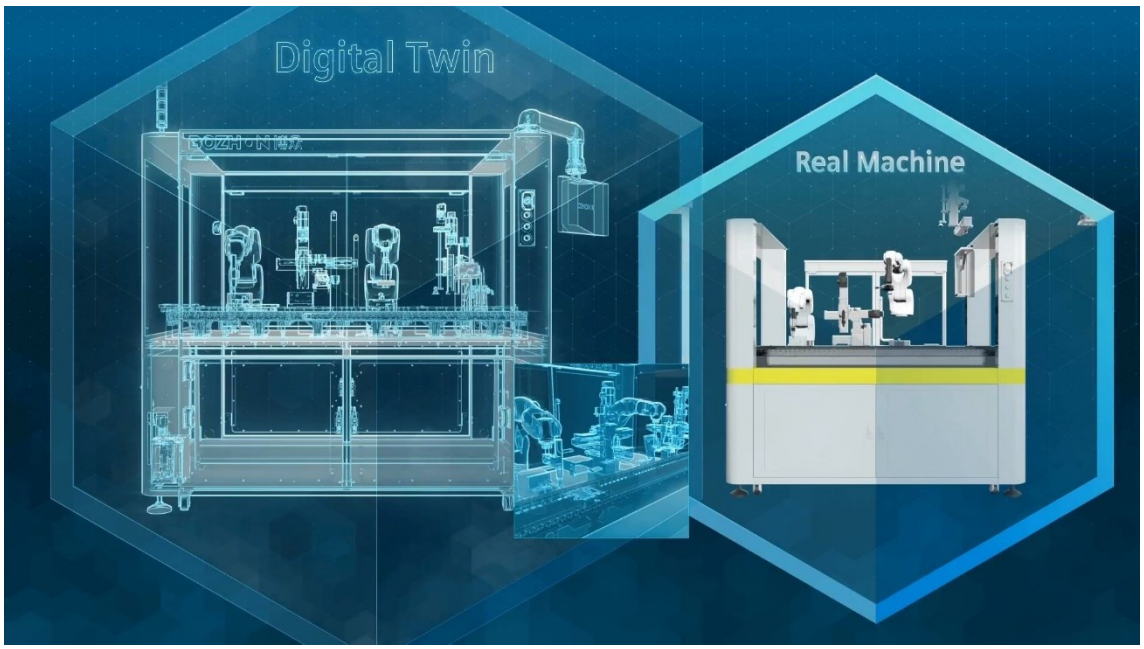


Figura 2 - Concetto di Digital Twin

Questo concetto estende l'uso della simulazione a tutte le fasi del ciclo di vita del prodotto, quindi simulando il prodotto (o processo) non solo per la fase progettuale o di testing, ma anche per la manutenzione, integrazione ecc. Si utilizzano le informazioni dei precedenti cicli di vita di un prodotto per simulare il nuovo ciclo di vita.

Automatizzare la costruzione e l'adattamento dei modelli simulativi può facilitare in modo significativo lo sviluppo di modelli di sistemi complessi. L'ottimizzazione, attraverso la modifica della struttura del modello, può essere eseguita costruendo diverse versioni del modello e dei dati di input (ovvero diversi scenari) e confrontando i risultati della simulazione.

Per accelerare lo sviluppo di versioni dei modelli, è possibile costruire algoritmi che costruiscono o modificano i modelli della simulazione in base ai dati di input del progetto. Questo metodo è particolarmente utile nei casi di modelli di grandi dimensioni e se le varianti del modello vengono preparate da un algoritmo, ad esempio un algoritmo di ottimizzazione. [3]

In questa tesi, seguendo il concetto di digital twin, viene presentato nel capitolo 3 un algoritmo che permette la generazione di varianti del sistema, mentre nel capitolo 5 viene descritta la fase di ottimizzazione del sistema attraverso la simulazione del modello, utilizzando le varianti generate dall'algoritmo.

1.3. Virtual Commissioning

Nell'industria digitale, uno degli argomenti di spicco è il *Virtual Commissioning*.

La messa in opera, anche detta *commissioning*, di una macchina automatica o di una linea è una delle più importanti fasi dello sviluppo di un progetto industriale. Il commissioning è l'ultimo step della fase evolutiva del progetto e precede solo la fase di produzione vera e propria (Figura 3). La messa in opera, infatti, costituisce il montaggio verso il cliente (o nella propria azienda) delle macchine e attrezzature. Durante questa fase si provvede anche al collaudo di tutti i componenti e processi, per verificare che la progettazione del sistema e i suoi componenti soddisfino i requisiti di progetto.

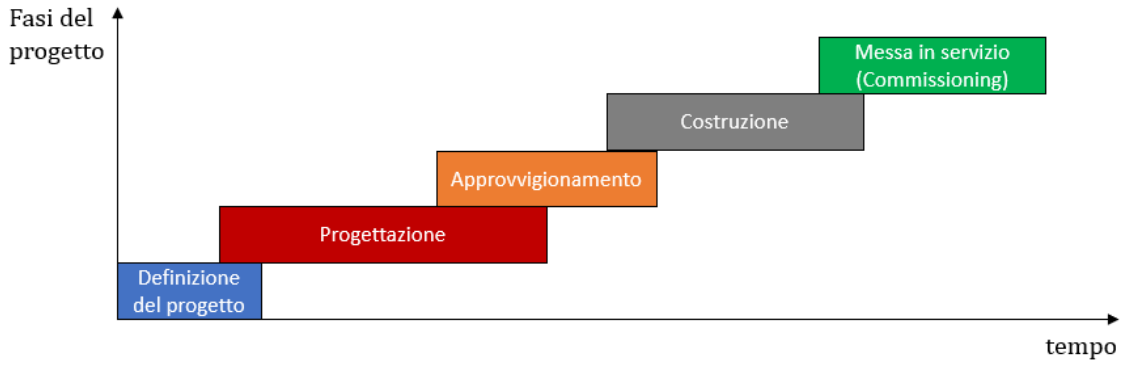


Figura 3 - Realizzazione tradizionale di un progetto

Tuttavia, la fase di massa in opera è conosciuta come una fase laboriosa e richiede molto tempo per essere completata. Inoltre, in un contesto tradizionale, in questa fase i modelli di simulazione non integrano nessun componente di automazione industriale come, ad esempio, i PLC (*Programmable Logic Controller*).

La *digital factory* integra attraverso l'uso di un modello virtuale anche la simulazione del PLC per il controllo del sistema. L'idea base del virtual commissioning è di connettere il modello dell'impianto digitale (*digital plant*) con un controller reale dell'impianto, in modo tale che ingegneri di differenti ambiti, come progettisti meccanici e softwaristi, possano lavorare su un modello comune insieme.

In questo modo, ad esempio, il programma sviluppato per il PLC può essere testato virtualmente in una simulazione delle operazioni prima che il montaggio fisico della macchina sia completato. Questo tipo di tecnologia viene definita come *SiL* (Software in the Loop). Questo tipo di approccio simulativo viene adoperato in questa tesi nel paragrafo 5.2.

Una delle motivazioni per cui si richiede la necessità di simulare la programmazione del PLC di una macchina è la posizione temporale in cui questa fase viene adoperata nella realizzazione del progetto.

Infatti, dopo la definizione del progetto, generalmente si inizia con la progettazione meccanica, in cui si definiscono le macchine e i componenti necessari al progetto, nonché si esegue anche la progettazione cinematica dei movimenti dei robot e componenti vari. Si prosegue con la progettazione dell'impianto elettrico, dei segnali di controllo delle macchine e la connessione di tutti i componenti. Come ultimo step si passa alla progettazione del software di controllo del PLC.

Nella fase di commissioning, in cui vengono montati e testati tutti i componenti, il testing del software viene eseguito come ultima fase.

Il risultato finale della messa in opera è un impianto pronto per la produzione.

Il metodo di realizzazione dell'impianto è di tipologia a cascata, in cui ogni step può essere eseguito solo dopo il precedente. Come conseguenza di ciò, il debugging del software può essere eseguito solo nella fase di commissioning.

Questa condizione porta a due principali problemi:

- *Bug nel software*. Il software presenta degli errori che sono difficili da scovare e talvolta possono portare anche a danneggiamenti dell'hardware. Il Processo di debugging è molto lungo. Come si può vedere dalla Figura 4, la fase di commissioning rappresenta solo il 15-20% del tempo totale di realizzazione del progetto, ma il 90% del tempo di commissioning è dedicato alla realizzazione dell'impianto elettrico e di controllo, di cui il 70% è impiegato esclusivamente per la risoluzione degli errori sul software.

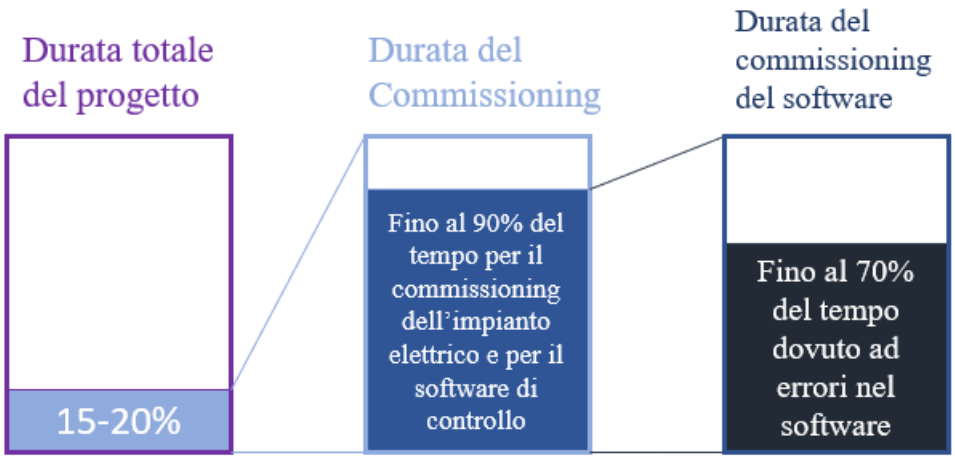


Figura 4 – Contributo degli errori nel software nei ritardi di consegna

- *L'errore è scoperto troppo tardi*. La fase di commissioning è portata avanti sotto costante pressione di tempo, pertanto, un ritardo in questa fase comporterebbe pesanti perdite a livello finanziario (Figura 5). Quando un errore viene scoperto in una fase avanzata del progetto, la sua correzione potrebbe comportare una modifica di tutte le componenti relazionate all'errore.

Da ciò si deduce che il debugging del software è preferibile che venga eseguito in una fase precedente del progetto.

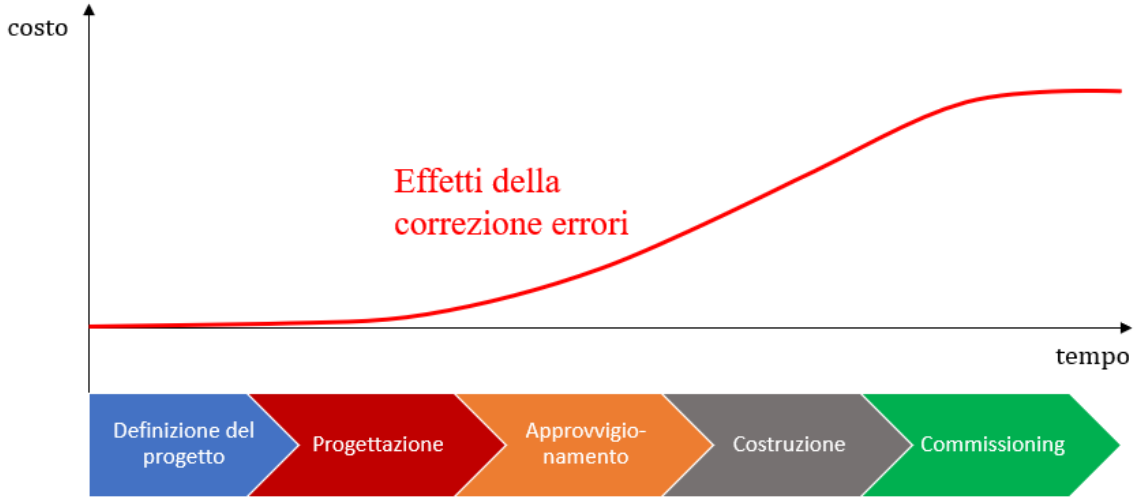


Figura 5 - Costo degli errori durante le fasi di realizzazzione di un progetto

Un approccio utilizzato per il superamento di queste problematiche è stato introdotto grazie al virtual commissioning. Una modellazione digitale completa del sistema permette la simulazione del programma del PLC sul modello digitale grazie alla tecnologia SiL, anticipando questa fase in un momento precedente alla messa in servizio (Figura 6).

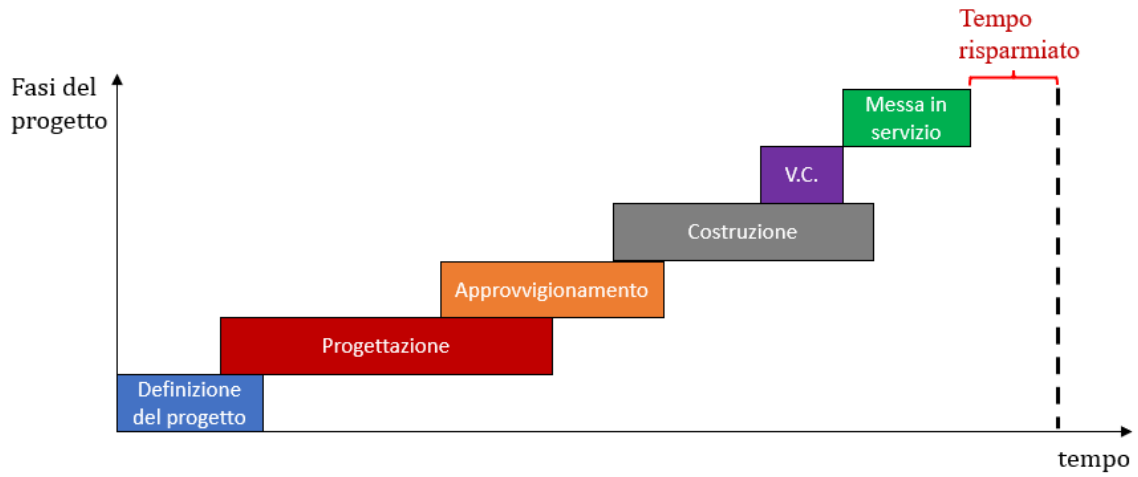


Figura 6 - Ruolo del virtual commissioning (V.C.) durante le fasi del progetto

Un'altra tecnologia utilizzata nel virtual commissioning è la tecnologia *HiL* (Hardware in the Loop). Questa tecnologia collega l'impianto reale, costituito da componenti meccatroniche, con l'impianto virtuale attraverso un PLC reale. Il collegamento può essere fatto sia in modalità semplice che con un reale sistema di comunicazione industriale.

L'obiettivo è quello di approssimare il comportamento del sistema simulato a quello di un sistema reale utilizzando un PLC messo in opera fisicamente all'impianto reale (Figura 7). Con questo metodo è possibile sviluppare e testare i sistemi automatici in maniera parallela ai sistemi elettrici e meccatronici. [4]

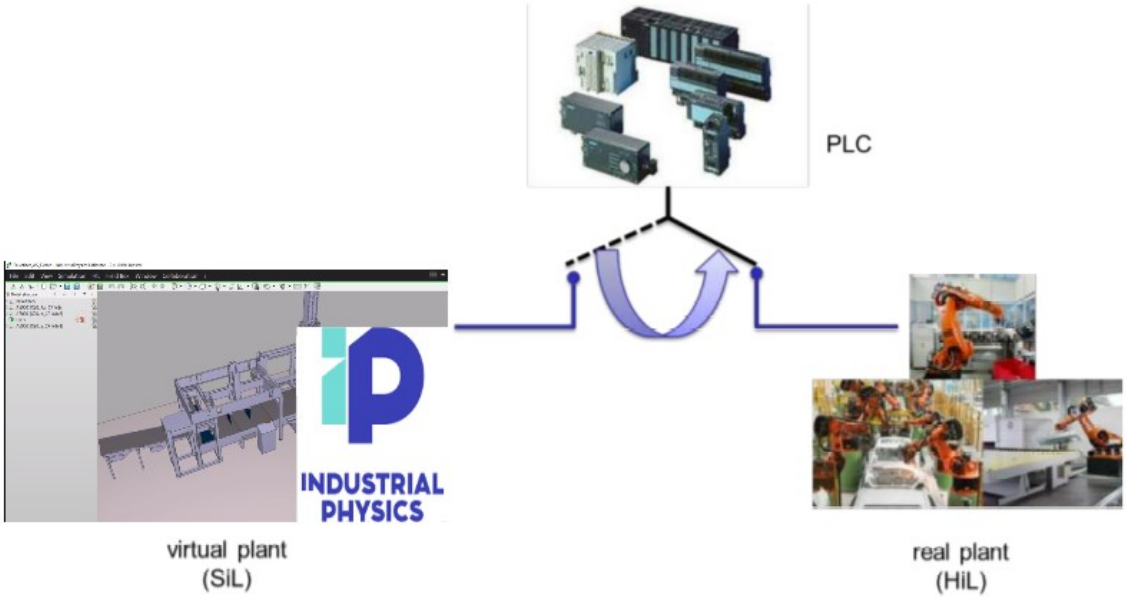


Figura 7 - Collegamento del virtual plant con il real plant attraverso il virtual commissioning

1.4. Macchine sequenziali e automazione

1.4.1. Introduzione sull'automazione

Un sistema di produzione può essere definito come un insieme di attrezzature integrate e risorse umane che esegue una o più operazioni di lavorazione e/o assemblaggio su una materia prima, una parte o un insieme di parti. I sistemi di produzione includono sia sistemi automatici che manuali.

L'automazione può essere definita come la tecnologia mediante la quale viene eseguito un processo o una procedura senza l'assistenza umana. Gli esseri umani possono essere presenti come osservatori o partecipanti, ma il processo stesso opera sotto la propria direzione. L'automazione è realizzata mediante un sistema di controllo che esegue un programma di istruzioni (Figura 8).

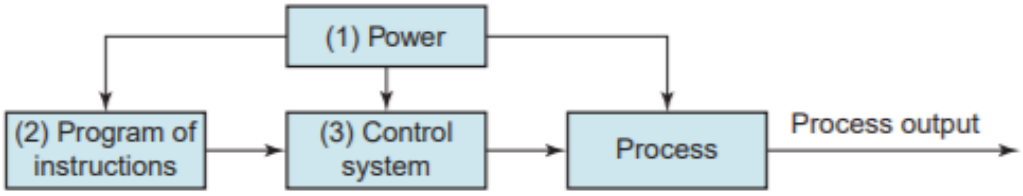


Figura 8 - Elementi di un sistema di automazione

Nei processi automatizzati più semplici, l'unica istruzione può essere quella di mantenere una certa variabile a un livello specificato, come la regolazione della temperatura in un forno. Nei processi più complessi invece, è richiesta una sequenza di azioni da eseguire durante il ciclo di lavoro, dove l'ordine e i dettagli di ciascuna azione vengono definiti da un programma di istruzioni. Ogni azione comporta modifiche in uno o più parametri di processo, come il cambiamento della posizione della coordinata x di un piano di lavoro di una macchina utensile o come l'accensione/spegnimento di un motore. I parametri di processo, che sono input del processo, possono essere continui o discreti (On o Off). I loro valori influenzano gli output del processo, che sono chiamati variabili di processo. Come i parametri di processo, le variabili di processo possono essere continue o discrete.

L'automazione e il controllo di processo vengono implementati utilizzando vari dispositivi hardware che interagiscono con la produzione e le apparecchiature associate (Figura 9). I sensori sono necessari per misurare le variabili di processo. Gli attuatori sono utilizzati per pilotare i parametri di processo. Inoltre, sono necessari vari dispositivi aggiuntivi per interfacciare i sensori e gli attuatori con il controller di processo, che di solito è un computer digitale.

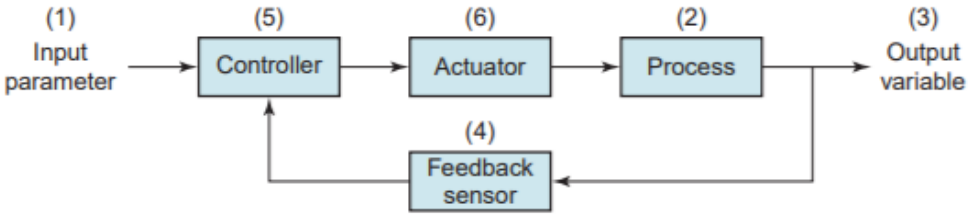


Figura 9 - Implementazione del processo in un sistema di controllo automatizzato

Un sensore è un dispositivo che trasduce uno stimolo fisico o una variabile di interesse (come ad esempio, temperatura, forza, pressione ecc.) in una forma fisica più conveniente (ad esempio, tensione elettrica) allo scopo di misurare la variabile. La conversione permette di interpretare la variabile come valore quantitativo.

Oltre al tipo di stimolo, i sensori sono anche classificati come analogici o discreti. Un sensore analogico misura una variabile analogica continua e la converte in un segnale continuo come la tensione elettrica. Termocoppie, estensimetri e amperometri sono esempi di sensori analogici. Un sensore discreto produce un segnale che può avere solo un numero limitato di valori. All'interno di questa categoria ci sono i sensori binari e i sensori digitali. Un sensore binario può assumere solo due possibili valori, come On e Off, oppure 0 e 1.

Per un dato sensore esiste una relazione tra il valore dello stimolo fisico e il valore del segnale prodotto dal sensore. Questa relazione tra ingresso e uscita è chiamata funzione di trasferimento del sensore, che può essere espressa come:

$$S = f(s)$$

dove S è il segnale di uscita del sensore (tipicamente tensione), s è lo stimolo o ingresso e $f(s)$ è la relazione che lega ingresso e uscita.

Un sensore binario (ad es. finecorsa, interruttore fotoelettrico) mostra una relazione binaria tra stimolo e uscita del sensore:

$$\begin{cases} S = 1, & se\ s > 0 \\ S = 0, & se\ s \leq 0 \end{cases}$$

Nei sistemi automatizzati, un attuatore è un dispositivo che converte un segnale di controllo in un'azione fisica, che di solito si riferisce ad un cambiamento di un parametro di processo (input del sistema). In genere, l'azione eseguita è di tipo meccanica.

Gli attuatori possono essere classificati in base al tipo di amplificatore utilizzato per la conversione. Possono essere di tipo elettrico, idraulico o pneumatico.

Molti sistemi automatici funzionano accendendo e spegnendo motori, interruttori e altri dispositivi per rispondere alle condizioni di funzionamento necessarie. Questi dispositivi di controllo utilizzano variabili binarie e pertanto, possono avere uno dei due possibili valori, 1 o 0, interpretati come On o Off, oggetto presente o non presente, livello di alta o bassa tensione, e così via.

La maggior parte dei sistemi di controllo dei processi utilizza un tipo di computer digitale come controller.

Un controllore di processo ampiamente utilizzato è un controllore logico programmabile (PLC) che è un controllore basato su microcomputer che utilizza istruzioni allocate nella memoria programmabile per implementare funzioni logiche, di sequenza, di temporizzazione, di conteggio e di controllo aritmetico, attraverso moduli di ingresso/uscita digitali o analogici , per il controllo di varie macchine e processi. [5]

1.4.2. Modellazione del comportamento di una macchina automatica

La descrizione del comportamento di una macchina automatica è un problema notoriamente difficile. La fisica prende in prestito dalla matematica la nozione di equazioni differenziali per descrivere entità mutevoli, fluidi, gas, ecc. Tuttavia, il comportamento delle macchine dipende da come queste vengono controllate, il dominio di interesse pertanto è puramente logico e non soddisfa praticamente mai le equazioni differenziali. Anche un semplice dispositivo come un motore a pistoni va oltre i formalismi delle equazioni differenziali.

Quello di cui si ha bisogno è la capacità di fornire una descrizione precisa di ciò che costituisce il comportamento di un sistema desiderato. Ad esempio, nel caso di un sistema prototipo, vorremmo descrivere le sequenze di interazione con l'ambiente di contesto. [6]

Per ottenere un linguaggio di questo tipo si fa riferimento alla teoria degli automi, che può essere definita come lo studio del comportamento dinamico di sistemi informatici a parametri discreti. All'interno di questa teoria si affrontano non solo problemi riguardanti i computer digitali ma anche problemi associati alla descrizione di come funziona una rete neurale, di come un uomo percepisce e reagisce all'ambiente, all'analisi di un sistema di trasmissione delle informazioni.

Qualunque sia il contesto in cui si opera, non si è interessati alla forma fisica del sistema ma piuttosto si desidera sviluppare dei modelli costituiti da componenti ideali, i cui comportamenti matematici approssimano il più possibile le proprietà del sistema. Quando un modello soddisfacente è pronto, si utilizzano le sue proprietà matematiche per studiare e descrivere il comportamento generale del sistema.

Il comportamento dinamico di sistemi di questo tipo è determinato da come il sistema è costruito a partire da componenti logici elementari (And, Or, Not ecc.) e da componenti di memoria. Tuttavia, in genere il sistema è presentato come una scatola nera e non è possibile conoscere in dettaglio quali siano i componenti elementari di cui è costruito. Il comportamento del sistema viene quindi descritto come un set di possibili stati in cui può trovarsi ad operare il sistema. [7]

Lo scopo di uno stato è ricordare la parte rilevante dello storico del sistema. Poiché esiste solo un numero finito di stati, generalmente non può essere ricordato l'intero storico e quindi il sistema deve essere progettato con attenzione per ricordare solo ciò che è davvero importante. Il vantaggio di avere solo un numero finito di stati è che rende possibile implementare il sistema attraverso un numero finito di risorse. Ad esempio, può essere implementato in hardware con un circuito o in software con un semplice programma in grado di prendere decisioni guardando una quantità limitata di dati.

Uno degli automi più semplici è l'interruttore on-off rappresentato in Figura 10. Il dispositivo ricorda se è nello stato "on" o se è nello stato "off" e consente all'utente di premere un pulsante il cui effetto dipende dallo stato dell'interruttore: se l'interruttore è spento, premendo il pulsante si passa allo stato acceso e se l'interruttore è acceso, premendo il pulsante si passa allo stato spento.

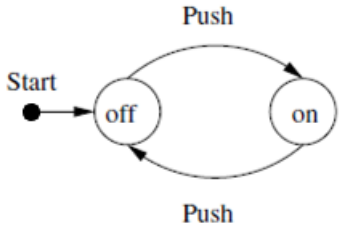


Figura 10 - Automa di un interruttore on-off

Gli automi vengono rappresentati utilizzando dei cerchi per identificare gli *stati* del sistema e delle frecce tra gli stati che sono etichettate da "*input*", che rappresentano influenze esterne sul sistema. Nell'esempio appena mostrato, l'intento delle frecce è che qualunque sia lo stato in cui si trova il sistema, quando viene ricevuto l'input "*Push*" si passa all'altro stato. Facoltativamente l'input sulle frecce può essere accompagnato anche da una condizione logica scritta tra parentesi quadre.

Uno degli stati è designato come "*stato iniziale*", ovvero lo stato in cui si trova inizialmente il sistema. Convenzionalmente lo stato inziale del sistema viene indicato da una freccia con pallino nero entrante nello stato.

Gli stati non sono disposti in maniera lineare ed è possibile spostarsi da uno stato in un altro attraverso delle transizioni di stato azionate da eventi.

I vantaggi dell'utilizzo di formalismi come gli automi è che il concetto di stato fornisce una nozione intuitiva del sistema che si sta modellando (gli stati sono interpretati come "foto istantanee" della situazione in un dato momento).

Le macchine a stati e i loro corrispondenti diagrammi di transizione di stato (o diagrammi di stato in breve) sono il meccanismo formale per raccogliere tali frammenti a formare un intero.

La grafica utilizzata è in realtà basata su un concetto più generale, l'*higraph*, che combina nozioni dai cerchi di Eulero, diagrammi di Venn e dagli ipergrafi. [8] [9]

Nonostante la loro semplicità, i diagrammi di stato possono diventare piuttosto difficili da leggere se il sistema cresce in complessità. Per armonizzare la lettura e facilitare l'utilizzo di questi diagrammi si utilizza una loro variante: i diagrammi di stato parte-intero (*Part-Whole Statecharts*).

I diagrammi di stato parte-intero sono basati su delle entità chiamate "*oloni*", unità comunicanti modulari, disposte in gerarchie parte-intero. Gli oloni ospitano un

comportamento (accessibile attraverso un'interfaccia e che viene eseguito in modo ricorsivo) e allo stesso tempo adempiono il duplice e complementare ruolo di parte (ovvero un componente) e intero (ovvero un insieme di componenti):

- come *parte*, l'interfaccia consente di controllare il comportamento dell'olone da altri oloni che lo possiedono come parte, attraverso input di comando diretti; allo stesso tempo il comportamento dell'olone emette output di notifica, attraverso l'interfaccia, verso altri oloni;
- come *intero*, il comportamento dell'olone controlla il comportamento di altri oloni contenuti al suo interno come parti, emettendo input di comando; allo stesso tempo il comportamento dell'olone può essere influenzato dagli output di notifica ricevuti dall'interfaccia di tali oloni.

Con tale approccio, l'interfaccia di ciascun olone è di fatto un automa che ha un insieme di stati e transizioni di stato. Le transizioni di stato possono essere attivate tramite l'interfaccia o possono avvenire in modo autonomo. L'interfaccia fornisce due insiemi di eventi, denominati *input* e *output* che etichettano, rispettivamente, le transizioni di stato attivabili e autonome. Gli eventi di input sono ricevuti dall'olone e innescano le transizioni di stato a cui sono associati. Gli eventi di output sono invece prodotti dall'olone quando vengono eseguite delle transizioni autonome a cui sono correlati. Gli eventi di input e output sono indicati come caratteristiche esterne, poiché descrivono il comportamento esterno dell'olone (Figura 11).

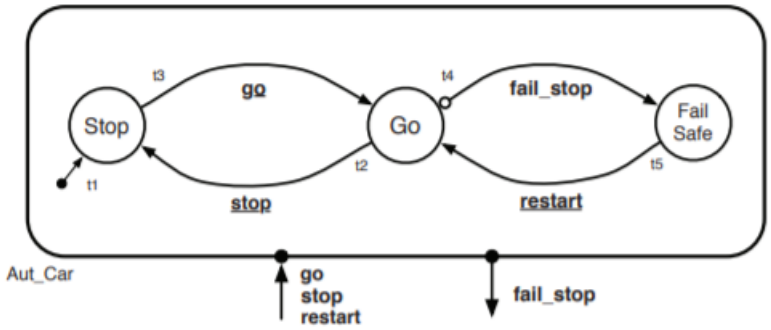


Figura 11 - Esempio di interfaccia di un olone

Il comportamento dell'olone che agisce nell'insieme sarà indicato come *comportamento completo* dell'olone o come *implementazione dell'interfaccia*, poiché

descrive come il comportamento dell'interfaccia viene reso disponibile attraverso il comportamento di altri oloni (Figura 12 e Figura 13).

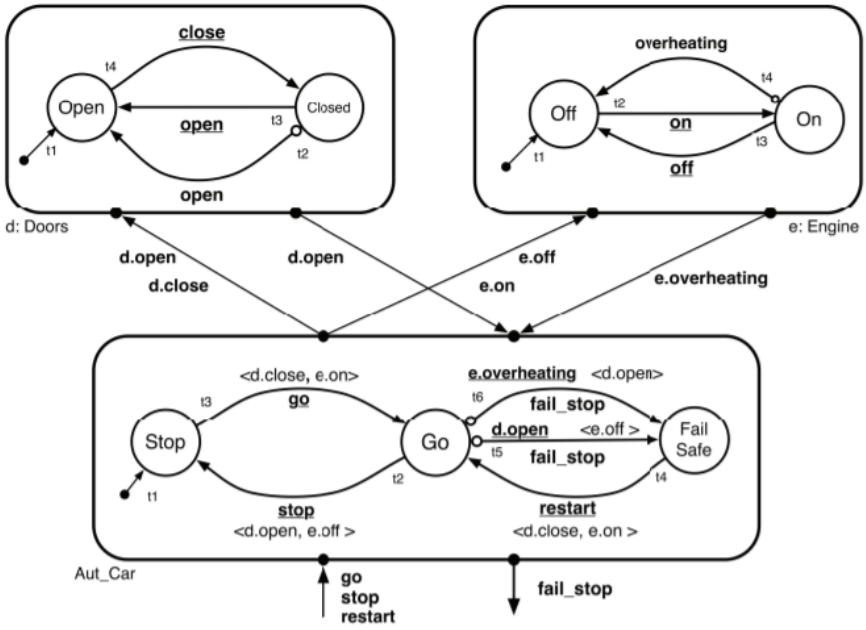


Figura 12 - Esempio di comportamento completo di un olone (implementazione)

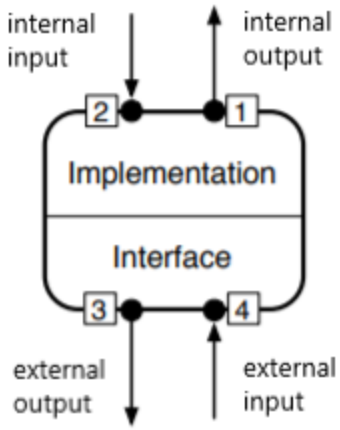


Figura 13 - Comportamento completo di un olone

Alcune interfacce però non possiedono alcuna implementazione, in quanto sono interfacciate direttamente all'oggetto o al fenomeno controllato, sia percependolo e sia agendo su di esso. In tal caso è presente solo l'interfaccia dell'olone.

Gli oloni, grazie alla loro modularità intrinseca, possono essere utilizzati per creare degli assemblaggi e formare una *oliarchia* (Figura 14).

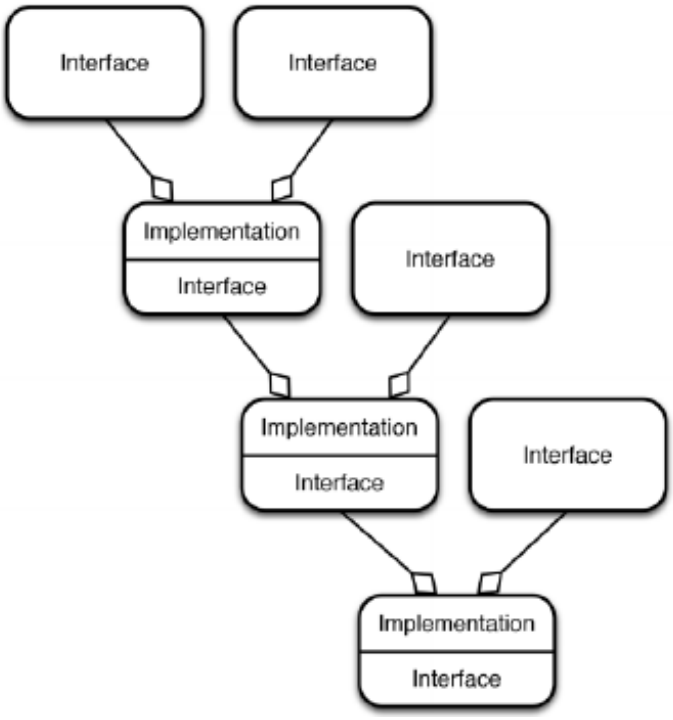


Figura 14 - Esempio di oliarchia

I due ruoli di parte e intero non sono simmetrici, poiché l'intero può controllare totalmente i suoi componenti, ma le parti non controllano l'intero. Le parti emettono semplicemente eventi di notifica verso l'intero, il cui comportamento specifica una reazione per ciascuno di tali eventi.

La logica alla base di tale asimmetria è che l'intero conosce le sue parti attraverso la loro interfaccia, ma le parti ignorano totalmente l'intero a cui appartengono. Questa logica serve al fine di preservare l'auto-contenimento e la riusabilità in molteplici contesti progettuali e realizzativi.

La sintassi dell'interfaccia di un olone (Figura 15) è costituita da:

- Gli stati dell'olone;
- I segnali di input e output. Il nome dei segnali viene disegnato accanto le frecce di input/output.

- Le transizioni degli stati. Si possono distinguere quelle autonome e quelle attivabili dall'esterno. Le transizioni attivabili dall'esterno sono provocate dai segnali di input, il nome del segnale posizionato accanto la freccia di transizione viene sottolineato per differenziarlo da un segnale di output. Le transizioni autonome invece vengono generate da sole, ad esempio dopo un timeout, e generano un segnale di output uscente dall'interfaccia. Le transizioni autonome sono differenziate dalle transizioni attivabili perché presentano un pallino bianco sulla coda della freccia.
- Un parametro booleano di attivazione. Viene scritto accanto ad una freccia di transizione ed è racchiuso tra parentesi quadre [C]. La presenza di tale elemento è facoltativa e la sua funzione è quella di generare la transizione quando è rispettata la condizione;

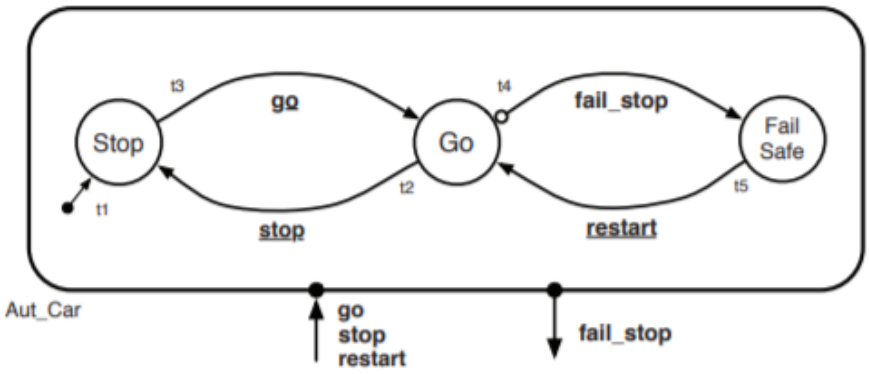


Figura 15 - Esempio di sintassi di un'interfaccia

La sintassi del comportamento completo di un olone (Figura 16) è costituita dalla stessa sintassi dell'interfaccia ma con l'aggiunta di altri elementi:

- Una lista di eventi di comando, disegnati accanto le frecce di comunicazione tra gli oloni di un insieme. Ogni comando rappresenta un evento di input per uno degli oloni dell'insieme;
- Un interruttore interno, scritto nella forma sottolineata olone-sorgente.evento, associato ad una transizione autonoma e che genera tale transizione;
- Un output interno, scritto nella forma <olone-bersaglio.evento>, che viene generato da una transizione autonoma e comanda un altro olone dell'insieme.

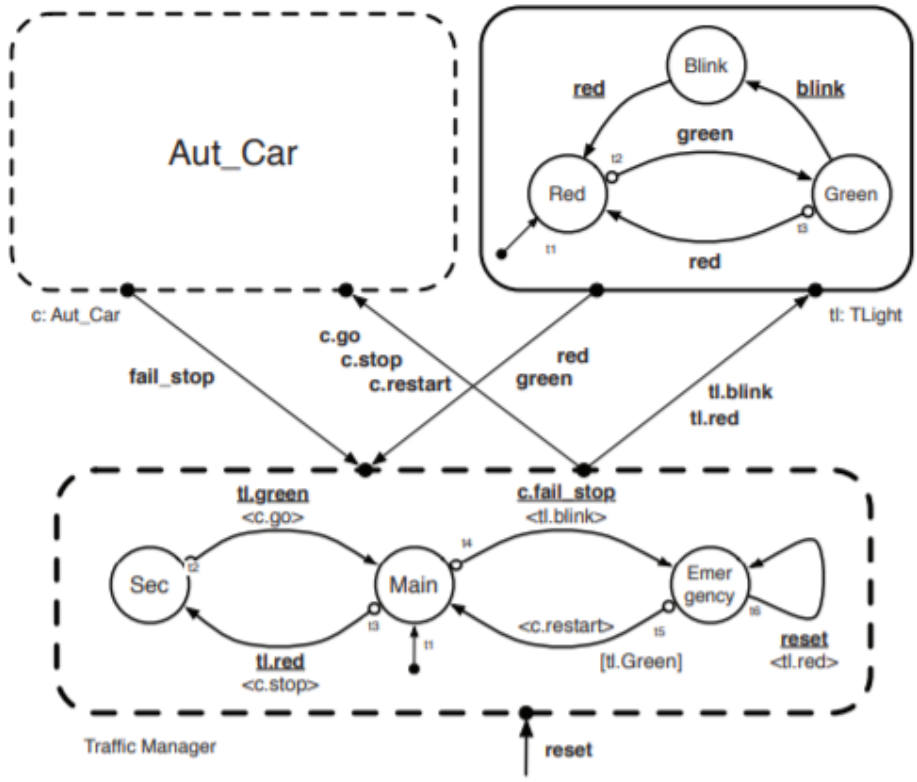


Figura 16 - Sintassi dell'implementazione di un olone

Le macchine a stati all'interno degli oloni operano attraverso un ciclo ricorsivo di computazione che itera volta per volta uno step di calcolo. Durante una iterazione di calcolo vengono eseguite tutte le operazioni matematiche contenute negli stati attivi, vengono eseguite eventualmente le transizioni di stato e vengono scambiati i segnali di comando tra gli oloni.

Il processo di scambio di informazioni non è simmetrico e in un sistema reale che funziona come un modello asincrono, ciascuna macchina è comandata da un singolo controllore o da un thread separato del processore. La comunicazione degli oloni è eseguita dal processore e le macchine, che sono indipendenti nel comportamento, vengono sincronizzate attraverso le porte di comunicazione in cui vengono scambiati i segnali di comando. [10] [11]

2. Sistemi sincro-dinamici

2.1. Modalità di funzionamento

I sistemi di sincronizzazione dinamica (o sincro-dinamici) sono macchine automatiche che funzionano come elemento di giunzione tra due diverse macchine di una linea di produzione. Le macchine collegate lavorano in maniera asincrona, pertanto lo scopo di un sistema di sincronizzazione dinamica è quello di sincronizzare il flusso di prodotti elaborato tra macchina a monte e macchina a valle.

Un sistema sincro-dinamico è di grande importanza nei casi in cui le due macchine collegate hanno un funzionamento diverso: una macchina lavora in funzionamento alternato e l'altra lavora in funzionamento continuo (Figura 17).

Macchina 2 / Macchina 1	Moto intermittente	Moto continuo
Moto intermittente	✓	✓
Moto continuo	✓	✗

Figura 17 - Combinazioni possibili di macchine collegate da un sistema sincro-dinamico

Le due macchine si interfacciano al sistema sincro-dinamico con una stazione di carico prodotti ed una stazione di scarico prodotti. Le stazioni di interfaccia consegnano o scaricano i prodotti ad una frequenza diversa: una macchina potrebbe essere più veloce dell'altra, oppure a parità di velocità una macchina lavora più prodotti dell'altra in una sola operazione. Il sistema sincro-dinamico aspetta di ricevere (o consegnare) un numero di prodotti sufficiente per eseguire l'operazione nella stazione successiva e per tale motivo è necessario che i prodotti siano trasportati da una stazione all'altra in tempi ben definiti affinché si sincronizzi il flusso di prodotti tra le due macchine.

La sincronizzazione delle due macchine avviene durante il trasporto dei prodotti da una stazione all'altra, dopo che questi sono stati raggruppati nel numero corretto. La

sincronizzazione può essere effettuata eseguendo una corretta sequenza di movimentazioni oppure, in alcuni casi, aumentando o diminuendo la velocità di movimentazione dei prodotti all'interno del sistema sincro-dinamico.

Di norma i sistemi sincro-dinamici sono integrati come modulo di una macchina che svolge altri compiti sul prodotto in lavorazione. [12] [13]

La macchina sincro-dinamica che viene progettata in questo elaborato di tesi è strutturata come in Figura 18.

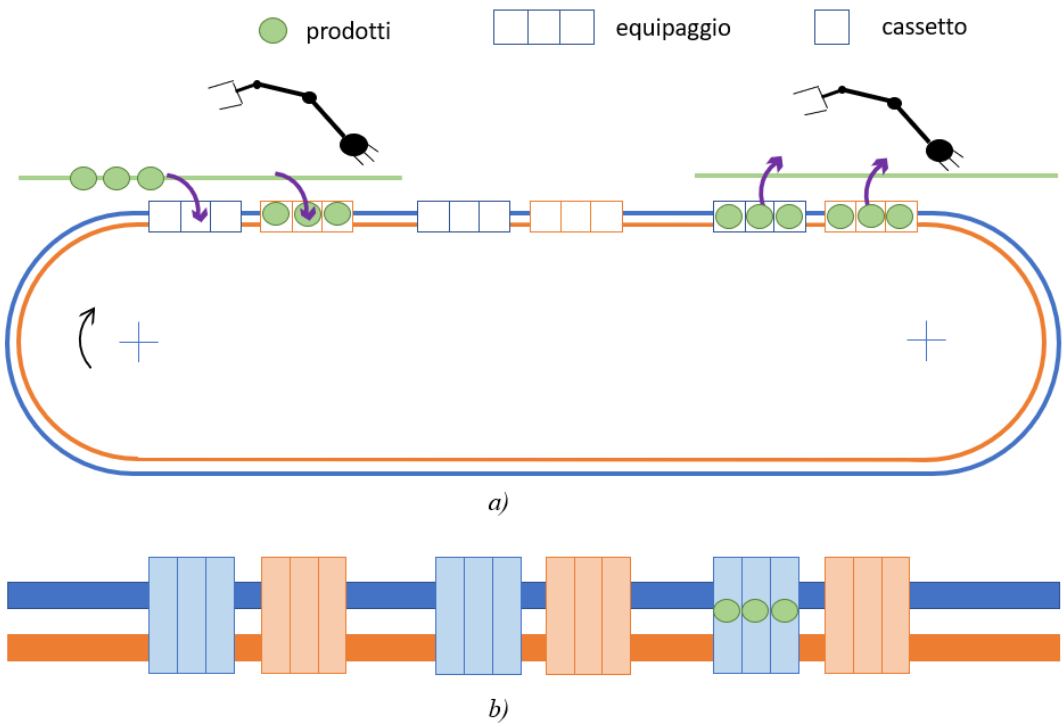


Figura 18 – Rappresentazione schematica della macchina sincro-dinamica, in cui i colori blu e arancione identificano cinghie di movimentazione distinte. a) vista frontale della macchina; b) vista dall'alto della macchina

Il sistema è composto principalmente da due cinghie di movimentazione disposte una accanto all'altra ed ognuna collegata ad un motore comandato separatamente.

I cassetti per ospitare i prodotti sono fissati sulle cinghie e raggruppati in equipaggi di cassetti che sono disposti con un certo passo l'uno dall'altro per poter essere equidistribuiti lungo tutta la cinghia.

In disposizione alterna ogni equipaggio sarà vincolato su una cinghia oppure sull'altra; pertanto, la movimentazione di un gruppo di equipaggi sarà indipendente dall'altro gruppo.

I prodotti vengono caricati all'interno dei cassetti nella zona rettilinea delle cinghie di movimentazione.

Le cinghie sono motorizzate separatamente sia per sincronizzare la produzione delle due macchine e sia per gestire eventuali ritardi: attraverso una gestione dinamica dei movimenti, si può evitare che eventuali ritardi di una macchina si ripercuotano sull'altra; si sfrutta un movimento più rapido di alcuni equipaggi per recuperare il ritardo e mantenere sempre sincronizzate le macchine collegate al sincro-dinamico.

L'utilizzo di questo sistema fornisce il vantaggio di avere la garanzia che i prodotti manipolati rimangano sempre a passo durante il passaggio dalla macchina a monte alla macchina a valle.

2.2. Dati di progetto e obiettivi

Il caso di studio progettato in questa tesi è composto da 8 equipaggi ciascuno con 20 cassetti, quattro equipaggi vincolati ad una cinghia e quattro vincolati sull'altra (Figura 19).

La lunghezza totale di una cinghia è di 10200 mm.

La produttività richiesta in uscita è di 500 pezzi al minuto.

I prodotti prelevati nella stazione di caricamento sono disposti con passo di 30 mm; pertanto, anche i cassetti sono disposti con lo stesso passo e quindi, la lunghezza totale dell'equipaggio è di 600 mm.

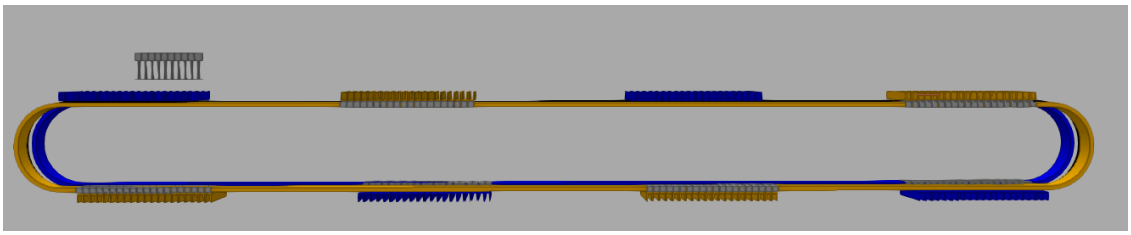


Figura 19 - Rappresentazione 3D del sistema sincro-dinamico con 8 equipaggi da 20 cassetti ognuno

La stazione di caricamento esegue le operazioni attraverso un robot pick & place a due assi mentre nella stazione di scaricamento sono presenti degli spingitori a stantuffo.

In **Errore. L'origine riferimento non è stata trovata.** è rappresentato il robot di caricamento della macchina reale.

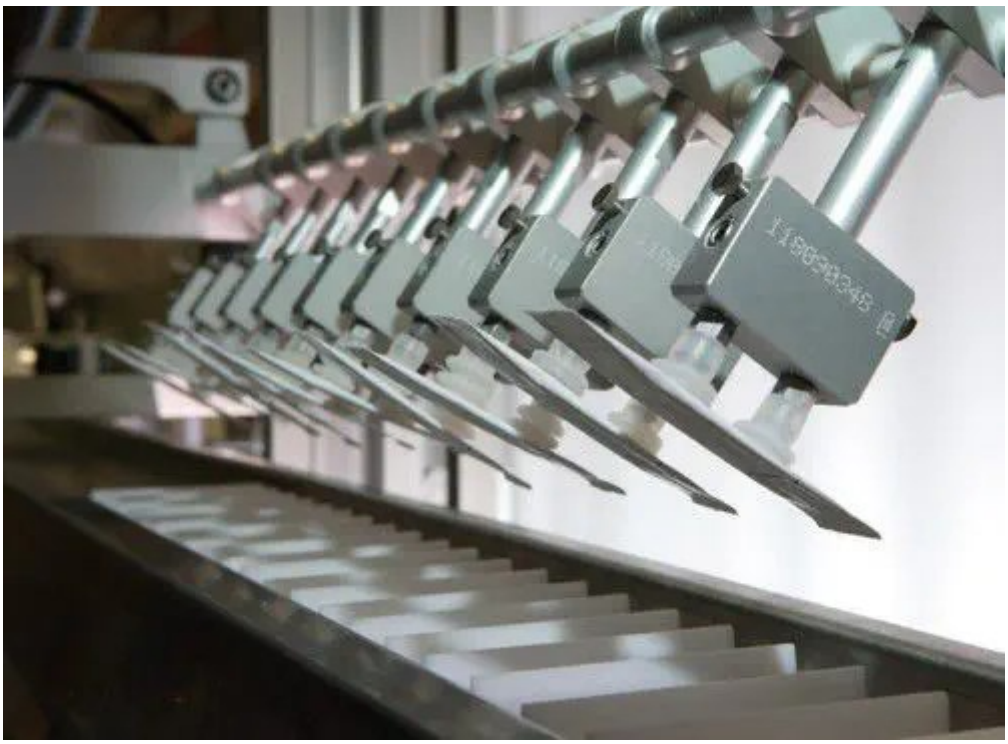


Figura 20 - Immagine rappresentativa del robot pick & place della macchina reale

Nella stazione di caricamento il robot preleva 10 prodotti alla volta e li deposita nell'equipaggio, inserendo un prodotto per ogni cassetto. Perché l'intero equipaggio venga riempito sarà quindi necessario eseguire due sessioni di caricamento.

Nella stazione di scaricamento gli spingitori espellono 20 prodotti assieme scaricando in una volta l'intero equipaggio.

La configurazione di questo sistema è rappresentata in Figura 21.

Prodotti manipolati dal robot di caricamento	Prodotti manipolati dagli spingitori di scaricamento
10	20

a)

Cassetti raggruppati in un equipaggio	n° operazioni per caricare un equipaggio	n° operazioni per scaricare un equipaggio
20	2	1

b)

Figura 21 - Configurazione della macchina; a) n° prodotti elaborati dalle stazioni; b) operazioni da eseguire in una stazione per completare un equipaggio

La zona della macchina in cui i prodotti vengono prelevati dalla stazione di caricamento si chiamerà zona di prelievo (Figura 22), mentre la zona della macchina in cui i prodotti vengono depositatati dalla stazione di scaricamento si chiamerà zona di consegna (Figura 23).

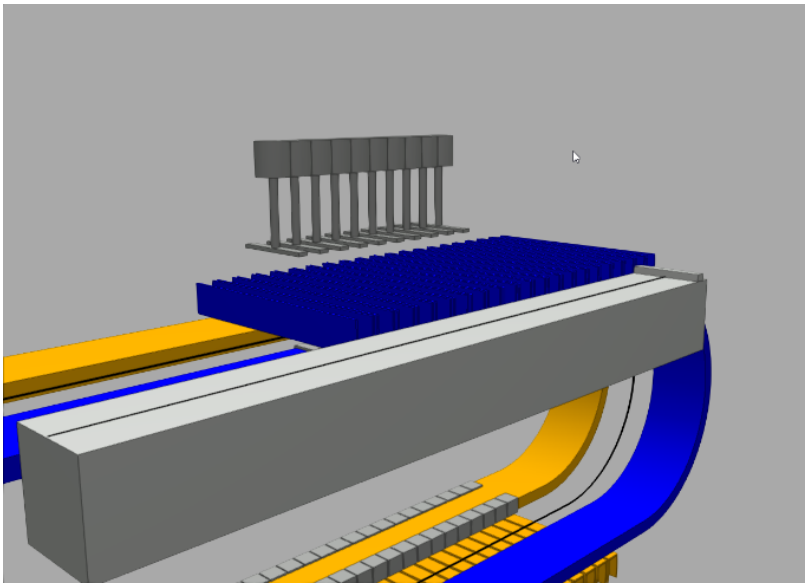


Figura 22 - Rappresentazione 3D della stazione di carico con annesso robot

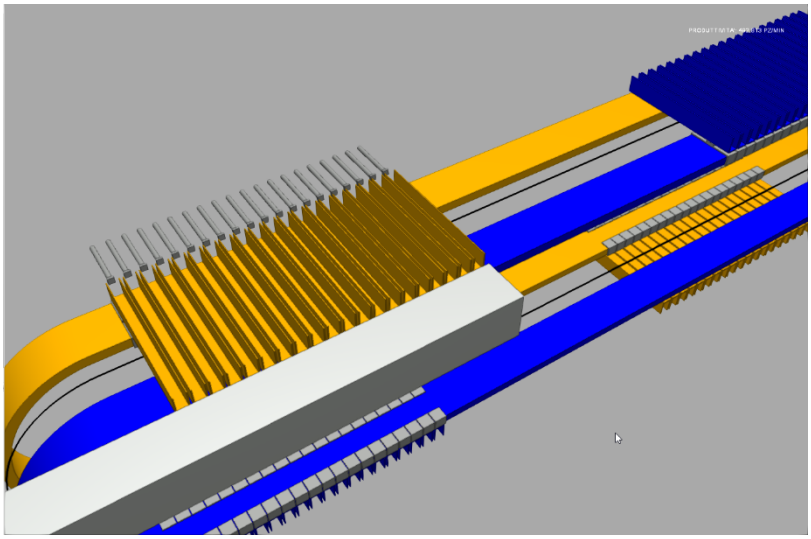


Figura 23 - Rappresentazione 3D della stazione di scarico con annessi spingitori

In Figura 24 è rappresentata la zona di prelievo della macchina reale.

Figura 24 - Immagine rappresentativa della macchina reale a monte del sistema sincro-dinamico

Per completare un ciclo di lavorazione, il robot di caricamento esegue i seguenti movimenti:

- prelievo dei prodotti dalla zona di prelievo;
- spostamento dalla zona di prelievo verso la cinghia;
- deposito dei prodotti sui cassetti;
- spostamento dalla cinghia verso la zona di prelievo.

Il tempo ciclo di operazioni del robot è in totale di 1,2 secondi, di cui 0,8 secondi è il tempo di deposito dei prodotti sui cassetti e 0,4 secondi è il restante tempo per il prelievo e gli spostamenti tra la zona di prelievo e la cinghia, e viceversa.

Nella stazione di caricamento, affinché l'equipaggio sia completamente riempito è necessario eseguire due operazioni di deposito e solo una operazione di movimento robot e prelievo; è sufficiente quindi attendere due volte un tempo di 0,8 secondi e solo una volta il tempo di 0,4 secondi. Infatti, in una situazione di funzionamento a regime, quando un equipaggio arriva nella stazione di caricamento il robot è già carico di prodotti e deve solamente depositarli; Dopo il primo deposito, il robot si sposta per prelevare altri dieci prodotti e l'equipaggio avanza per esporre i restanti cassetti vuoti. Dopo il prelievo, il robot si sposta nuovamente e deposita gli ultimi dieci prodotti nei cassetti. Una volta finito l'ultimo deposito, prima che il robot inizi a spostarsi verso la zona di prelievo, l'equipaggio di cassetti può già muoversi. Il tempo di 0,8 secondi, quindi rappresenta il tempo di sosta minima nella stazione di caricamento dell'equipaggio.

Le operazioni di scaricamento sono costituite semplicemente dalla spinta dei prodotti dai cassetti verso la zona di consegna. Ad ogni spinta vengono consegnati 20 prodotti per un tempo di operazione di 0,8 secondi.

In Figura 25 sono rappresentati gli spostamenti degli equipaggi durante una operazione di carico ed una operazione di scarico.

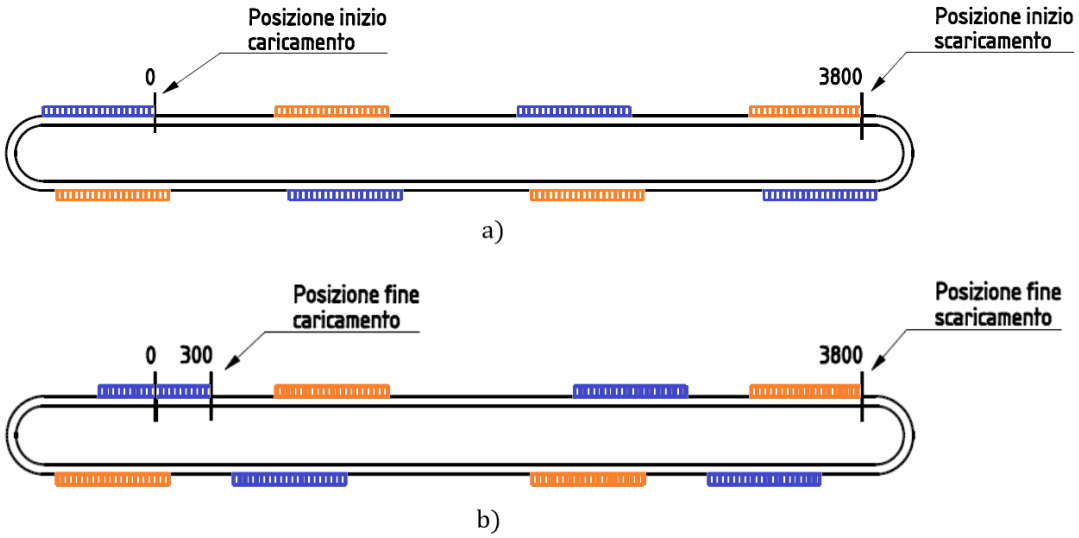


Figura 25 - a) inizio prima operazione di caricamento e prima operazione di scaricamento; b) inizio seconda operazione di caricamento; in figura è rappresentato il sistema di coordinate in mm.

La stazione di caricamento si trova all'inizio della zona rettilinea delle cinghie di movimentazione e la posizione della testa di un equipaggio, ovvero la posizione in cui vengono caricati i cassetti, viene definita come coordinata di origine.

Nella stazione di caricamento vengono eseguite due operazioni di deposito per poter riempire un equipaggio; pertanto, dopo la prima operazione di deposito è necessario che l'equipaggio si sposti in avanti di 300 mm per poter permettere il deposito nei restanti cassetti. La posizione che l'equipaggio deve raggiungere per poter permettere il caricamento dei restanti cassetti viene definita come posizione di fine caricamento e indicata come "FC".

In Figura 26 è rappresentata schematicamente la zona di lavoro di questa stazione.

Figura 26 – Zona di lavoro della stazione di caricamento

La stazione di scaricamento si trova a 3800 mm di distanza dalla coordinata 0. L'equipaggio viene scaricato tutto in una volta e pertanto la posizione di fine scaricamento, definita come "FS", è sempre 3800 mm (Figura 27).

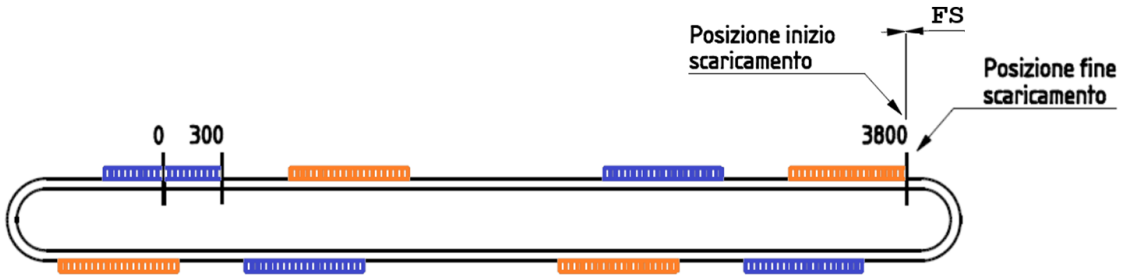


Figura 27 – Zona di lavoro stazione di scaricamento

L'obiettivo di progettazione è l'ottimizzazione della macchina sincro-dinamica in modo tale che possa garantire una migliore gestione dei tempi di movimentazione degli equipaggi. Con questi dati di progetto, come si vedrà in seguito, la macchina ha pochi margini di adattamento per compensare gli eventuali ritardi e per ottimizzare la macchina è necessario effettuare delle modifiche. In questo senso, per ottimizzare il sistema, viene lasciata libertà di modifica di alcuni parametri che generano di conseguenza una nuova configurazione del sistema.

Durante la progettazione si inizierà dapprima ad analizzare a fondo la configurazione di macchina, determinandone la prestazione. In seguito, verranno analizzate le altre configurazioni possibili per poterle confrontare e scegliere la migliore soluzione.

Una modifica apportabile al sistema è l'utilizzo di un robot nella zona di caricamento che preleva e deposita i prodotti a passo di 60 mm (i cassetti invece rimangono a passo di 30 mm). Con questa configurazione il caricamento dei primi 10 prodotti avviene depositando in cassetti alterni su tutta la lunghezza dell'equipaggio, in seguito l'equipaggio avanza di una distanza FC di 30 mm e vengono riempiti i restanti 10 cassetti (Figura 28).

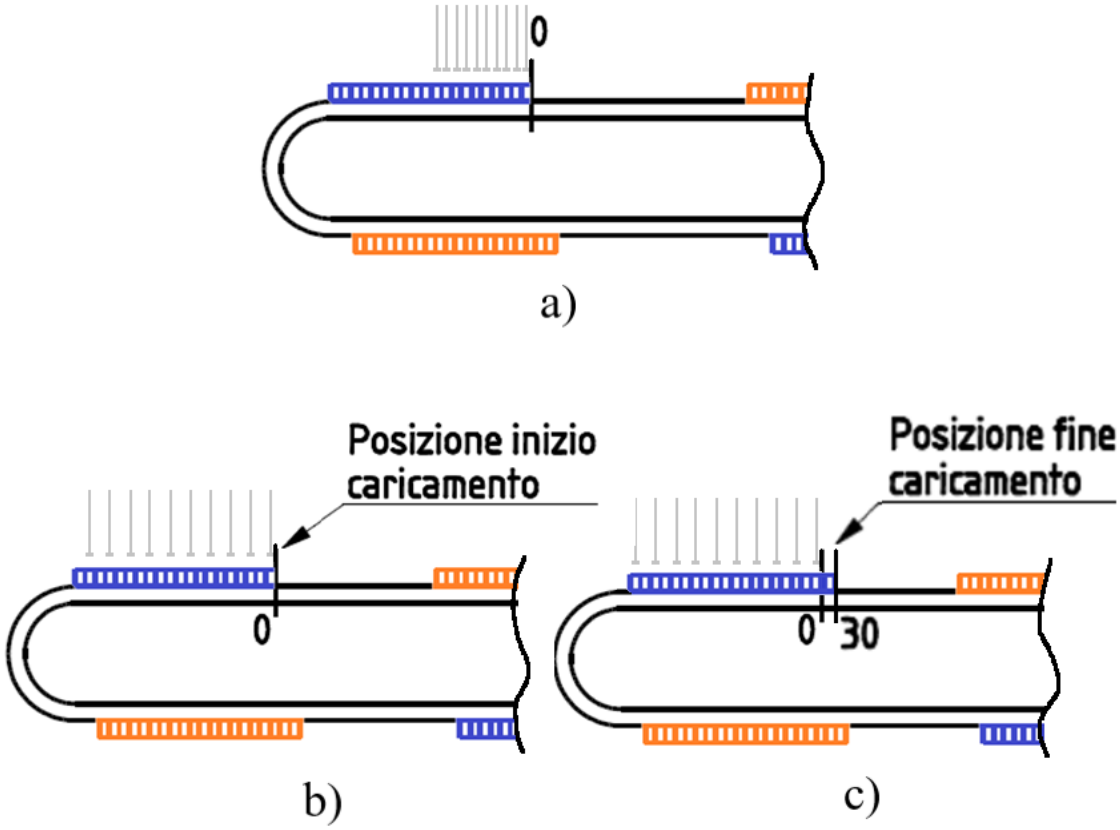


Figura 28 – a) Configurazione macchina con robot di caricamento a passo 30 mm; b) posizione di inizio caricamento per la configurazione macchina con robot di caricamento a passo 60 mm; c) posizione di fine caricamento per la configurazione macchina con robot di caricamento a passo 60 mm

Le altre configurazioni possono essere generate modificando il tipo di equipaggi da montare, ovvero il numero totale di equipaggi e il numero di cassetti in essi contenuto può variare per accomodare meglio le esigenze di progetto.

Aumentando il numero di cassetti in un equipaggio, questi diventano più lunghi e potrebbero non rispettare i vincoli di ingombro. Utilizzando equipaggi più lunghi ma in numero inferiore si potrebbe risolvere il problema di ingombro ma allo stesso tempo potrebbero sorgere dei problemi riguardo la cinematica del sistema: per poter riempire o svuotare un equipaggio è necessario eseguire più operazioni di carico e scarico e pertanto, la distanza da compiere da una stazione all'altra potrebbe non consentire il trasporto di un equipaggio nei tempi necessari Per affrontare i problemi nel generare queste configurazioni di equipaggi verrà esposto in questa tesi un algoritmo che

automaticamente fornisce un set di soluzioni possibili per determinati dati iniziali e vincoli.

Generando delle leggi di moto e utilizzando un modello digital twin del sistema è possibile eseguire infine una valutazione di prestazione di ogni configurazione, decidendo quale tra esse è la soluzione migliore per ottimizzare la linea.

3. Algoritmo per la generazione di configurazioni

3.1. Tipologia di algoritmo e metodo di calcolo

Nell'affrontare il problema, è necessario elaborare una strategia che consenta in maniera efficace di ottenere un insieme di soluzioni affidabili.

Come primo passo è necessario definire il tipo di problema da affrontare: il numero di equipaggi e il numero di cassetti sono sempre un numero intero; pertanto, il problema è di tipo discreto a numeri interi. Questa informazione suggerisce un metodo di calcolo da adottare: quello utilizzato in questo studio è l'*exhaustive search method*. [14]. Quest'ultimo è un approccio diretto che viene sfruttato per risolvere problemi di piccola scala. Il metodo prevede di enumerare tutti i possibili valori candidati come soluzione ed esamina tutte le possibili combinazioni di risultati tra le variabili del problema.

Insieme all'exhaustive search method si implementa anche un algoritmo di tipo *divide and conquer*. La logica alla base di questo approccio è di dividere il problema generale in sotto problemi più piccoli e più semplici da affrontare; in seguito, si risolvono i sotto problemi e si combinano le loro soluzioni per ottenere la soluzione generale del problema (Figura 29).

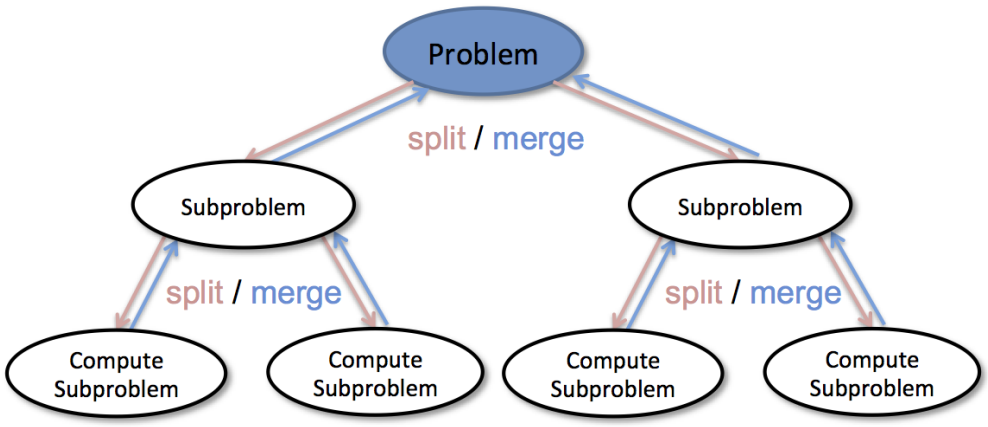


Figura 29 - Algoritmo divide and conquer

In generale la soluzione ai sotto problemi si ottiene in maniera ricorsiva, ma se il problema è abbastanza semplice si può risolvere in maniera diretta. Si parlerà rispettivamente di caso ricorsivo e caso base. [15]

In questa tesi il problema generale da risolvere è di trovare il numero di equipaggi che possono essere montati in macchina per dato numero di cassetti raggruppati in un equipaggio. Tutti gli equipaggi, il cui numero totale è un valore incognito, sono equidistribuiti sulla lunghezza della cinghia. Implementando la metodologia dell'algoritmo divide and conquer, si divide la lunghezza totale delle cinghie in due parti, una parte costituisce lo spazio tra la zona di carico e la zona di scarico e viene definita dal parametro "*IS*", l'altra parte invece è la lunghezza complementare che costituisce il resto della cinghia (Figura 30).

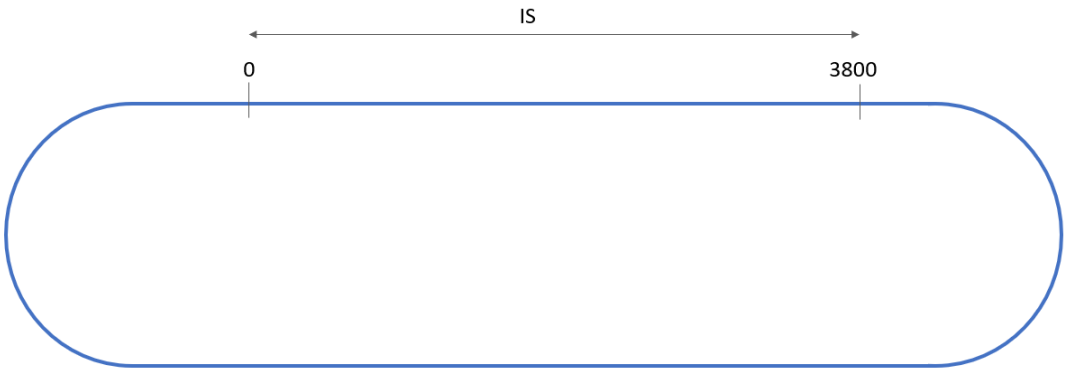


Figura 30 - Divisione lunghezza cinghia

Effettuando questa partizione, il problema generale viene diviso in due: calcolare il numero di equipaggi che possono essere montati nella zona IS e calcolare il numero di equipaggi che possono essere montati nella restante parte. Il numero di equipaggi della zona IS viene identificato dal parametro "N" mentre gli altri dal parametro "M".

Con questa partizione si può distribuire più facilmente la quantità di equipaggi lungo tutta la cinghia rispettando i vincoli di progetto, che sono determinati dalle zone di carico e scarico prodotti e sono rappresentate dagli estremi del dominio IS.

3.2. Impostazione del problema e definizione dei vincoli

Prima di analizzare la prestazione di una configurazione di macchina è necessario valutare quali siano le altre possibili configurazioni di utilizzo della macchina.

Ogni configurazione avrà lo stesso flusso di prodotti in uscita ed entrata ma i cicli di operazioni saranno differenti poiché, il numero di cassetti ed equipaggi cambia in ogni configurazione (Figura 31).

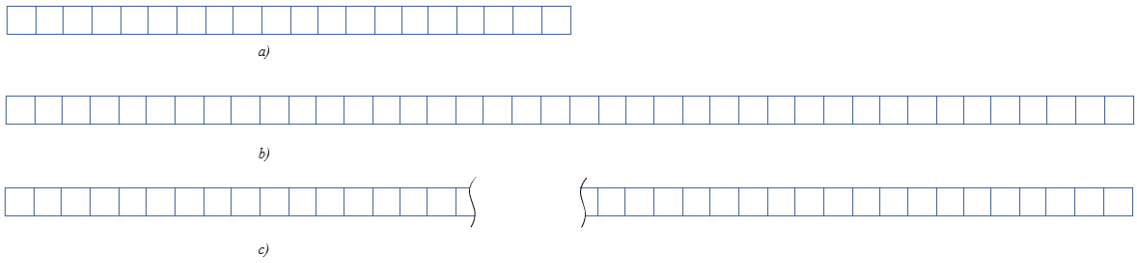


Figura 31 – a) configurazione con equipaggio di 20 cassetti; b) configurazione con equipaggio di 40 cassetti; c) configurazione con equipaggio di 60 cassetti

Ad esempio, per riempire completamente un equipaggio di 20 cassetti serviranno 2 operazioni di caricamento con un tempo totale di 2 secondi, mentre per riempire completamente un equipaggio di 40 cassetti serviranno 4 operazioni di caricamento con un tempo totale di 4,4 secondi.

Per determinare quali siano le configurazioni possibili si descrive la macchina geometricamente. Si definiscono i vincoli geometrici e si crea una funzione che rispetti questi vincoli.

Le ipotesi fatte sono le seguenti:

- Tutti gli equipaggi devono avere lo stesso passo.
- Quando un equipaggio di una cinghia si trova nella zona di caricamento prodotti, un equipaggio dell'altra cinghia deve trovarsi nella zona di scaricamento prodotti (Figura 32).

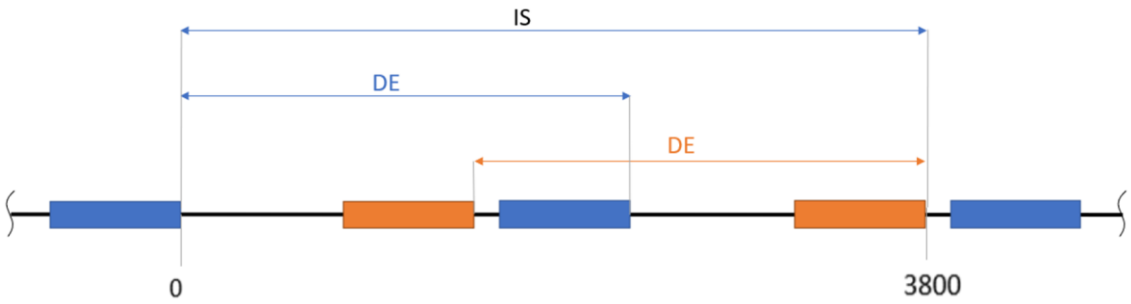


Figura 32 - Vincoli geometrici degli equipaggi

In figura si evidenzia con "DE" la distanza tra le teste degli equipaggi appartenenti alla stessa cinghia, ovvero il loro passo.

Per chiarezza di esposizione si descrive dapprima la geometria ipotizzando che ci siano soltanto due equipaggi di diverso colore tra le due stazioni di carico e scarico (escludendo i due equipaggi che per ipotesi si trovano già in una stazione). In seguito, si generalizzeranno i vincoli per considerare la presenza di un numero maggiore di equipaggi nella zona che separa le due stazioni.

Con "IS" si indica la coordinata in cui l'equipaggio inizia le operazioni di scaricamento e rappresenta di fatto la distanza che intercorre tra le due stazioni di carico e scarico.

In questa configurazione 8x20 (8 equipaggi da 20 cassetti) le operazioni di caricamento sono due e pertanto la posizione di fine caricamento non coincide con quella iniziale ma si trova a 300 mm da essa. Si identifica quindi con "FC" il parametro che tiene conto della variabilità di questo dato con le altre possibili configurazioni (Figura 33).

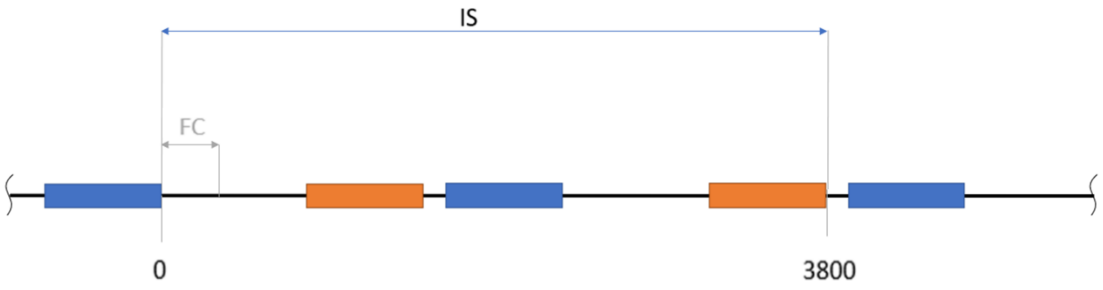


Figura 33 – Variabile FC per localizzare la posizione di fine caricamento

Allo stesso modo, nella configurazione 8x20 lo scarico dei prodotti avviene in una sola operazione e quindi posizione di inizio scaricamento e fine scaricamento coincidono. In altre configurazioni però è possibile che la posizione di fine scaricamento "FS" non coincida con quella di inizio scaricamento poiché le operazioni di scaricamento potrebbero terminare in più cicli (**Errore. L'origine riferimento non è stata trovata.**).

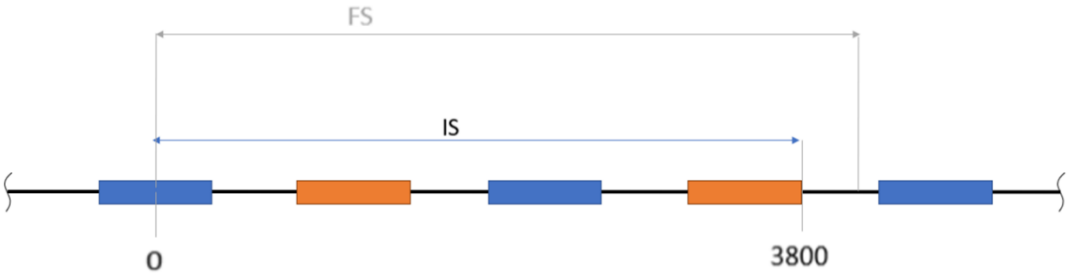


Figura 34 – Variabile FS per localizzare la posizione di fine scaricamento

Tutti gli equipaggi hanno sempre lo stesso passo, però la distanza relativa tra due equipaggi di diversa motorizzazione non è fissata; infatti, è proprio il movimento relativo tra questi equipaggi che sta alla base del funzionamento del sistema sincro-dinamico.

La distanza relativa tra due equipaggi di diversa motorizzazione è una diretta conseguenza del passo tra due equipaggi della stessa motorizzazione (variabile DE). Per evitare che ci siano compenetrazioni tra equipaggi diversi si possono identificare due casi limite.

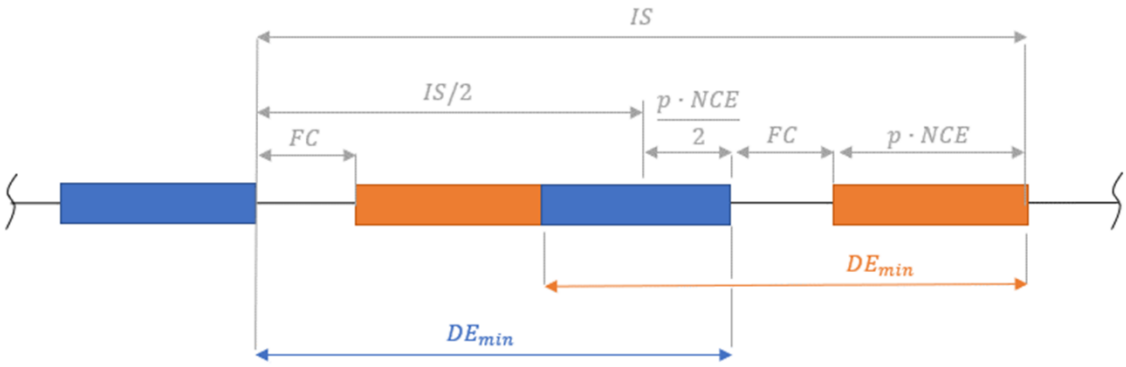


Figura 35 – Caso limite 1: variabile DE minima

In Figura 35 è rappresentato graficamente come la variabile DE non possa scendere sotto un certo valore minimo: se DE dovesse diminuire ulteriormente si avrebbe compenetrazione di due equipaggi appartenenti a cinghie diverse.

Identificando con "p" la costante che tiene conto del valore del passo dei cassetti in un equipaggio e con "NCE" la variabile che tiene conto di quanti cassetti possono esserci in equipaggio, la lunghezza totale di un equipaggio è ovviamente:

$$lunghezza\ equipaggio = p \cdot NCE$$

Il valore minimo che deve assumere la variabile DE lo si può verificare dalla Figura 35, basta risolvere il sistema:

$$\begin{cases} IS = DE_{min} + FC + p \cdot NCE \\ DE_{min} = 2 \cdot p \cdot NCE + FC \end{cases}$$

Da cui si ottiene:

$$DE_{min} = \frac{IS + p \cdot NCE}{2}$$

L'altro caso limite invece, si determina ipotizzando che un equipaggio non deve invadere la zona di caricamento di un equipaggio diverso, ovvero non deve sconfinare nella zona FC (Figura 36).

Infatti, se DE dovesse superare il valore suddetto ci sarebbe un equipaggio (arancione in figura) sconfinante nella zona di caricamento di un altro equipaggio (blu in figura), pertanto mentre un equipaggio si sta scaricando l'altro equipaggio non potrà avanzare nei

suoi successivi caricamenti (o viceversa) e se ci fossero dei ritardi da recuperare questa situazione potrebbe solo peggiorare le cose.

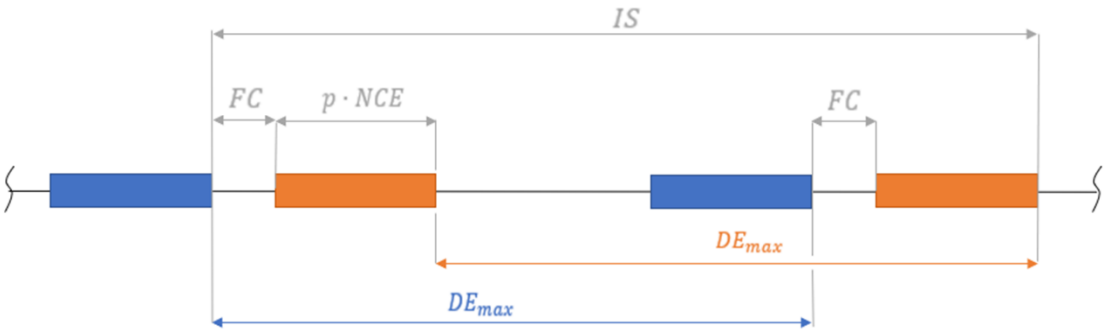


Figura 36 - Caso limite 2: Variabile DE massima

In Figura 36 si può vedere che per impedire ad un equipaggio di invadere la zona di fine caricamento FC, il valore massimo di DE deve essere:

$$DE_{max} = IS - FC - p \cdot NCE$$

Tra tutte le configurazioni possibili, potrebbe capitare di avere una stazione di scaricamento che ha bisogno di più cicli di operazioni per scaricare completamente un equipaggio e in particolare potrebbe capitare che le operazioni di scaricamento siano più numerose di quelle di caricamento. Se questo è il caso, allora bisogna garantire uno spazio libero di fine scaricamento "$FS - IS$" davanti ad un equipaggio.

I casi limite di quest'ultima situazione saranno simili ai precedenti ma si calcolano nel seguente modo.

Dalla Figura 37 si ottiene il valore minimo risolvendo il sistema:

$$\begin{cases} IS = p \cdot NCE + DE_{min} \\ DE_{min} = 2 \cdot p \cdot NCE + (FS - IS) \end{cases}$$

Da cui si ricava:

$$DE_{min} = \frac{FS + p \cdot NCE}{2}$$

Dalla Figura 38 si ottiene il valore massimo:

$$DE_{max} = IS - p \cdot NCE$$

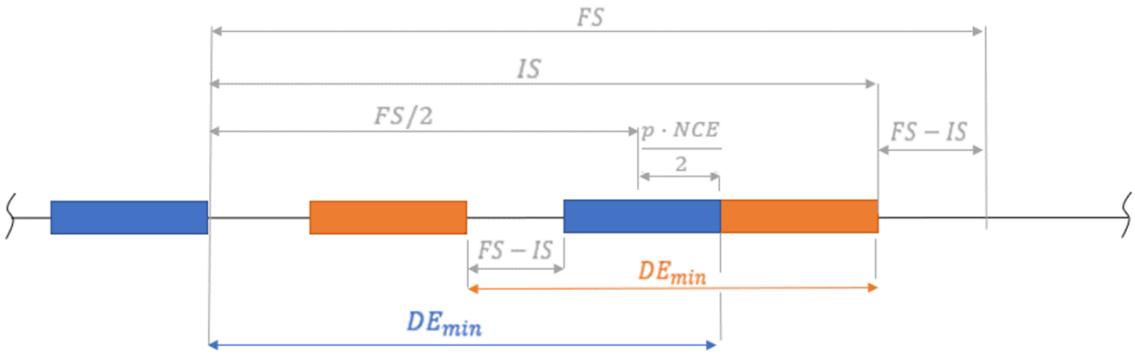


Figura 37 - Caso limite 1 con operazioni di scaricamento maggiori di quelle di caricamento

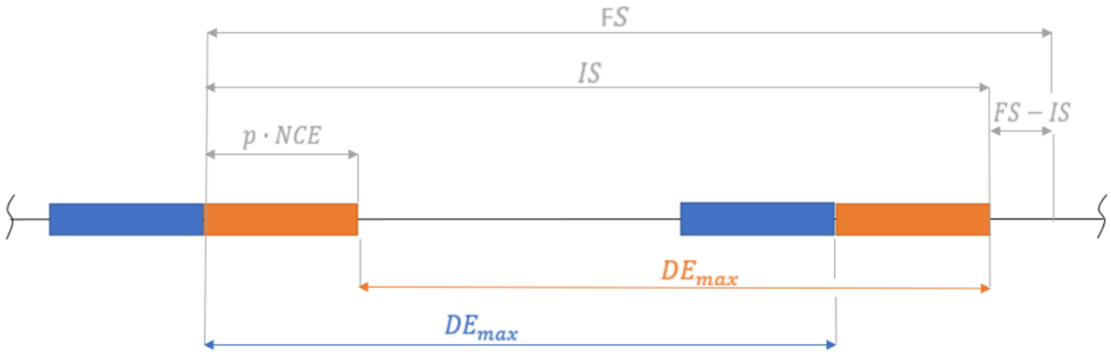


Figura 38 - Caso limite 2 con operazioni di scaricamento maggiori di quelle di caricamento

Non rimane adesso che generalizzare i vincoli in modo tale che si possano calcolare i valori limite anche nei casi in cui ci possano essere più di due equipaggi tra le stazioni.

Identifichiamo con "I" il numero di prodotti prelevati in una singola operazione di caricamento e con "U" il numero di prodotti consegnati in una singola operazione di scaricamento; ne segue automaticamente che se $I < U$ allora le operazioni di caricamento saranno più numerose di quelle di scaricamento e viceversa se $I > U$.

Tra le due stazioni, il numero di equipaggi blu è sempre in numero uguale a quello degli equipaggi arancioni (escludendo gli equipaggi che per ipotesi sono già in una

stazione). Per tale motivo indichiamo con "N" il numero di coppie di equipaggi che possono trovarsi nella zona IS.

Pertanto, nel caso più generale i vincoli diventano:

$$Se\ I < U: \begin{cases} DE_{min} = \dfrac{IS + p \cdot NCE}{N+1} \\ DE_{max} = \dfrac{IS - FC - p \cdot NCE}{N} \end{cases}$$

$$Se\ I > U: \begin{cases} DE_{min} = \dfrac{FS + p \cdot NCE}{N+1} \\ DE_{max} = \dfrac{IS - p \cdot NCE}{N} \end{cases}$$

3.3. Implementazione su calcolatore dell'algoritmo

Per costruire l'algoritmo divide and conquer si utilizza il software di calcolo MATLAB®.

L'algoritmo prevede di calcolare gli N possibili equipaggi nella zona IS e gli M possibili equipaggi nella restante parte della cinghia. Per ipotesi si è considerato che nella zona di carico sia sempre presente un equipaggio e nella zona di scarico sia sempre presente un altro equipaggio di diversa motorizzazione; nel calcolo degli N ed M equipaggi sono esclusi gli equipaggi presenti nelle zone di carico e scarico e verranno semplicemente sommati al risultato finale.

Un'ulteriore considerazione da fare è che il numero totale di equipaggi blu è uguale al numero degli equipaggi arancioni e gli equipaggi sono disposti in maniera sequenziale alternando sempre uno blu con uno arancione. Nel calcolare il numero possibile di equipaggi si può considerare più convenientemente una coppia di equipaggi blu ed arancione; quindi, i parametri N ed M saranno rappresentativi di una coppia di equipaggi di diversa motorizzazione e non di un singolo equipaggio (Figura 39).

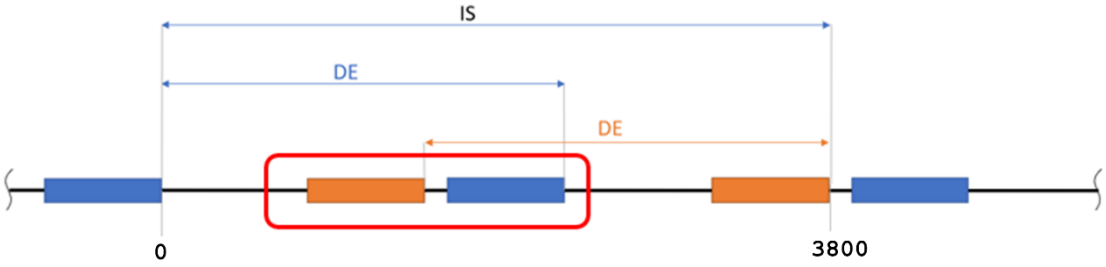


Figura 39 - Raggruppamento in coppia di equipaggi di diversa motorizzazione

Considerando gli equipaggi a coppie si può integrare nel raggruppamento anche il vincolo sulla distanza minima che questi due equipaggi debbano avere tra di loro, ovvero il vincolo sulla distanza FC se siamo nel caso $I < U$, sulla distanza FS se invece siamo nel caso $I > U$ (Figura 40).

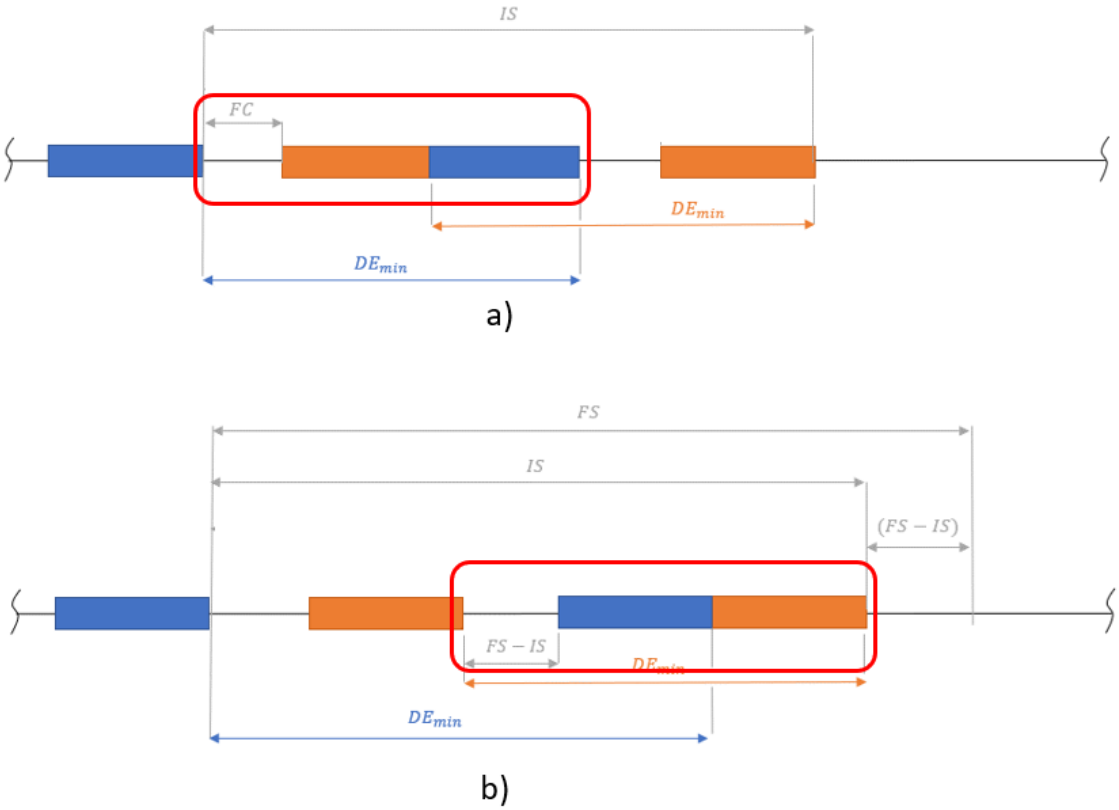


Figura 40 - Raggruppamento del vincolo sulla distanza minima con una coppia di equipaggi. a) caso $I < U$; b) caso $I > U$

Per ipotesi abbiamo stabilito che nella zona di scaricamento deve esserci sempre un equipaggio e anche questo equipaggio deve mantenere una distanza minima dagli altri; considerando questo vincolo, le N coppie di equipaggi si possono distribuire solo nella restante parte della zona IS, che chiameremo "lunghezza libera" e questa si calcola sottraendo dal parametro IS la lunghezza di un equipaggio più la distanza minima (Figura 41).

$$Se\ I < U: \quad \begin{cases} lunghezza\ libera = IS - FC - p \cdot NCE \\ lunghezza\ raggruppamento = 2 \cdot (p \cdot NCE) + FC \end{cases}$$

$$Se\ I > U: \quad \begin{cases} lunghezza\ libera = IS - (FS - IS) - p \cdot NCE \\ lunghezza\ raggruppamento = 2 \cdot (p \cdot NCE) + (FS - IS) \end{cases}$$

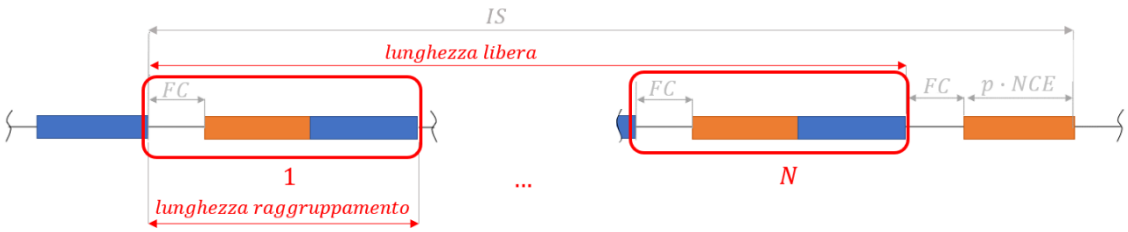


Figura 41 - Lunghezza libera e lunghezza raggruppamento nel caso $I < U$

Per distribuire le restanti coppie di equipaggi M è necessario ripetere il ragionamento precedente: si considera una lunghezza libera e una lunghezza di raggruppamento. In questo caso però la lunghezza libera si ottiene sottraendo dalla lunghezza totale della cinghia L la lunghezza della zona IS e la lunghezza occupata dagli equipaggi presenti nelle zone di carico e scarico assieme alle distanze minime a loro associate (Figura 42).

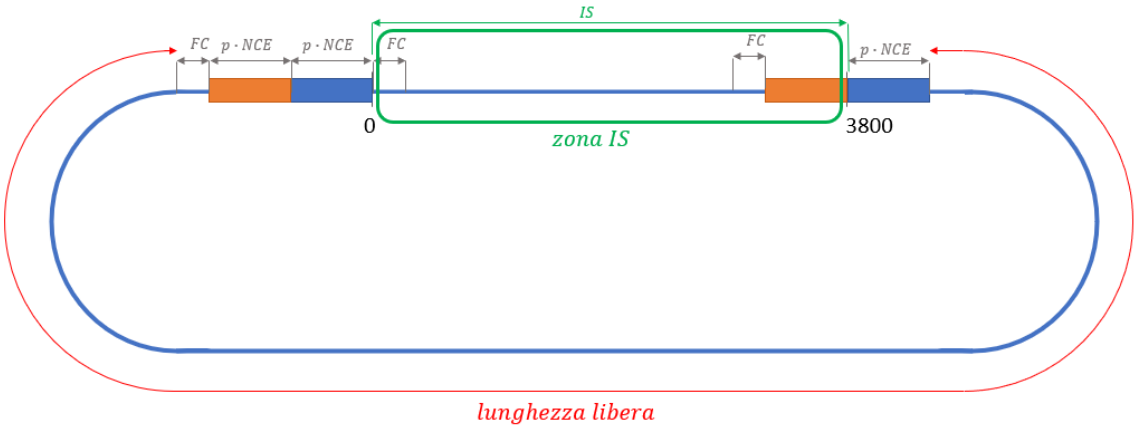


Figura 42 - Lunghezza libera per il calcolo delle M coppie di equipaggi nel caso $I < U$.

Le formule per il calcolo di M sono:

$$Se\ I < U: \begin{cases} lunghezza\ libera = L - p \cdot NCE - IS - (2 \cdot p \cdot NCE + FC) \\ lunghezza\ raggruppamento = 2 \cdot (p \cdot NCE) + FC \end{cases}$$

$$Se\ I > U: \begin{cases} lunghezza\ libera = L - p \cdot NCE - 2 \cdot p \cdot NCE \\ lunghezza\ raggruppamento = 2 \cdot (p \cdot NCE) + (FS - IS) \end{cases}$$

Per calcolare quante N coppie di equipaggi possono entrare nella zona IS basterà dividere la lunghezza libera disponibile della zona IS con la lunghezza occupata dal raggruppamento, mentre per calcolare quante M coppie di equipaggi possono entrare nella restante cinghia si divide la lunghezza libera disponibile fuori dalla zona IS con la lunghezza del raggruppamento.

Infine, il calcolo della distanza minima dipende esclusivamente da quante operazioni devono essere svolte per completare il carico o scarico di un equipaggio. Più sono le operazioni più la distanza minima deve essere ampia per garantire lo spazio a sufficienza all'equipaggio per poter avanzare durante tutte le operazioni.

La formula di calcolo dipende solo dal numero di cassetti in equipaggio NCE (dunque dalla lunghezza totale dell'equipaggio considerando il passo dei cassetti p) e dalla lunghezza di equipaggio che viene riempita (o svuotata) per singola operazione: cioè dalla quantità I di prodotti lavorati per singola operazione se siamo nel caso $I < U$ o dalla quantità U se siamo nel caso $I > U$.

Le formule per il calcolo sono le seguenti:

$$Se\ I < U: \quad FC = \left(\frac{NCE}{I} - 1\right) \cdot p \cdot I$$

$$Se\ I > U: \quad FS = \mathrm{IS} + \left(\frac{NCE}{U} - 1\right) \cdot p \cdot U$$

In questo modo abbiamo completato la divisione in sotto-problemi e abbiamo risolto i due sotto-problemi associati in maniera diretta.

Le formule implementate sul calcolatore sono mostrate in Figura 43.

```
I=10;                   %n° prodotti ingresso per singola operazione robot
U=20;                   %n° prodotti uscita per singola operazione robot
NCE=20;                 %n° cassetti in un equipaggio
IS=3800;                %distanza tra le stazioni di carico/scarico
L=10200;                %lunghezza totale cinghia
p=30;                   %passo dei cassetti su un equipaggio
pNCE=p*NCE;             %lunghezza di ogni equipaggio di cassetti

if I<=U
    FC=(NCE/I-1)*p*I;                   %lunghezza zona di fine caricamento
    N1=(IS-FC-pNCE)/(2*pNCE+FC);                %calcolo del numero di
                                                %coppie di equipaggi
                                                %possibili tra le due
                                                %stazioni di lavoro
                                                %considerando i vincoli

    M1=(L-pNCE-IS-(2*pNCE+FC))/(2*pNCE+FC);    %calcolo delle restanti
                                                %coppie di equipaggi
                                                %possibili tranne le due
                                                %coppie presenti ai vincoli
else
    FS=IS+(NCE/U -1)*p*U;               %lunghezza zona di fine scaricamento
    N1=(IS-(FS-IS)-pNCE)/(2*pNCE+(FS-IS));
    M1=(L-pNCE-FS-2*pNCE)/(2*pNCE+(FS-IS));
end
```

Figura 43 - Implementazione del metodo divide and conquer su MATLAB

Sommando la soluzione N con la soluzione M e le due coppie di equipaggi presenti all'interfaccia della zona IS (essendo stati di fatto esclusi dal calcolo di N ed M), si ottiene la soluzione al problema generale ovvero il numero totale di equipaggi NE:

$$NE = N + M + 2$$

Le soluzioni N ed M in generale non sono dei numeri interi e pertanto vanno approssimate per difetto.

Tuttavia, prima di unire le soluzioni dei sotto problemi è necessario verificare che queste siano compatibili con i vincoli definiti nel paragrafo precedente.

I vincoli sono definiti sul passo minimo DE_{min} e sul passo massimo DE_{max} che intercorre tra due equipaggi; per ottenere il passo DE della soluzione generale basta distribuire uniformemente gli equipaggi lungo tutta la lunghezza della cinghia:

$$DE = \frac{L}{NE}$$

Se DE è maggiore del vincolo superiore o minore del vincolo inferiore allora la soluzione non è compatibile con i vincoli.

Prima di applicare i vincoli si può fare una importante constatazione. La soluzione N così trovata rappresenta il numero massimo di coppie di equipaggi che possono entrare nella zona IS, ma questo valore non necessariamente è il valore di equipaggi da scegliere, si può infatti decidere di inserire un numero inferiore di equipaggi a quello massimo. Lo stesso ragionamento vale anche per la soluzione M, che a sua volta rappresenta il numero massimo di equipaggi possibili nel resto della cinghia. Così facendo si espandono le soluzioni N ed M in due vettori contenenti un insime di numeri interi che vanno da 0 al valore massimo trovato in precedenza.

La soluzione generale NE a sua volta non sarà più un unico valore, ma un ventaglio di possibili combinazioni tra gli elementi del vettore $\overline{N}$ e gli elementi del vettore $\overline{M}$

Per poter confrontare tutte le possibili combinazioni si utilizza l'exhaustive search method. La sua implementazione in MATLAB può essere fatta con una serie di cicli *for*, che sono tuttavia dispendiosi a livello computazionale se lo spazio delle variabili diventa grande. In questa tesi si ricorre ad un approccio più efficiente: la vettorizzazione [16]. Il vantaggio di questo approccio è di ottenere una velocità di calcolo del codice più rapida, inoltre il codice è più compatto, permettendo quindi una migliore gestione degli errori e dei bug.

La vettorizzazione sostituisce i cicli *for* con operazioni vettoriali e matriciali. Pertanto, per implementare il metodo in MATLAB è necessario precondizionare i dati per creare vettori e matrici omogenee per eseguire le operazioni.

I vettori $\bar{N}$ ed $\bar{M}$si creano semplicemente con l'uso dell'operatore ":" come mostrato in Figura 44 nelle righe 60 e 61, che provvede anche ad approssimare per difetto tutti i valori in numeri interi.

```
60 -    N=0:1:N1;                                %intervallo di valori possibili
61 -    M=0:1:M1;                                %intervallo di valori possibili
62
63 -    [X,Y]=meshgrid(N,M);
64 -    NE2=X+Y+2;                               %n° totale di coppie di equipaggi
```

Figura 44 - Creazione di matrici omogenee per l'exhaustive search

I vettori ottenuti sono i seguenti:

$$\bar{N} = [0 \quad 1]$$

$$\bar{M} = [0 \quad 1 \quad 2]$$

Nella riga 63 invece si utilizza la funzione nativa *meshgrid* per generare due matrici $\bar{\bar{X}}$ ed $\bar{\bar{Y}}$ a partire dai vettori $\bar{N}$ ed $\bar{M}$. Queste due matrici avranno la stessa dimensione e pertanto si prestano a tutti gli effetti alle operazioni matriciali, qual è la somma svolta nella riga 64.

La funzione meshgrid prende in input i due vettori $\bar{N}$ ed $\bar{M}$, che rispettivamente hanno dimensione (1x2) e (1x3) e creano due matrici $\bar{\bar{X}}$ ed $\bar{\bar{Y}}$ di dimensioni (3x2), ovvero ci saranno tante righe quanti sono gli elementi di $\bar{M}$ e ci saranno tante colonne quanti sono gli elementi di $\bar{N}$ (Figura 45).

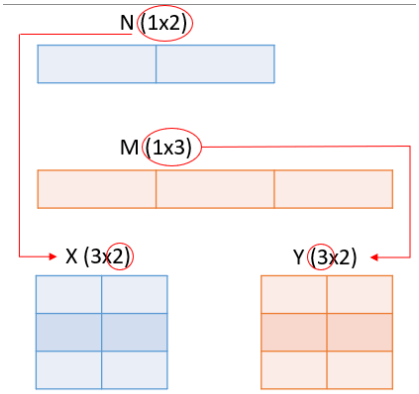


Figura 45 - Generazione matrici con meshgrid

La matrice $\bar{\bar{X}}$ è composta solo da elementi del vettore $\bar{N}$, ogni riga di $\bar{\bar{X}}$ è la ripetizione del vettore riga $\bar{N}$:

$$\bar{\bar{X}} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

La matrice $\bar{\bar{Y}}$ è composta invece solo da elementi del vettore $\bar{M}$, ogni colonna di $\bar{\bar{Y}}$ è la ripetizione del vettore colonna $\bar{M}^T$:

$$\bar{\bar{Y}} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$$

Si può vedere che sommando le due matrici $\bar{\bar{X}}$ ed $\bar{\bar{Y}}$ si ottiene una matrice cui ogni elemento è una combinazione diversa di somma tra gli elementi di $\bar{N}$ ed $\bar{M}$.

Per ogni combinazione otteniamo anche il passo degli equipaggi (Figura 46).

```
76 -    DE2=L./NE2;                    %passo degli equipaggi equidistribuito
77                                     %sulla lunghezza totale cinghia
```

Figura 46 - Calcolo del passo degli equipaggi

Eseguendo la riga 64 e la riga 76 si ottiene quindi l'insieme completo di soluzioni al problema generale, cioè il numero possibile di equipaggi in tutte le combinazioni e il loro relativo passo. Queste soluzioni vengono rappresentate in Figura 47 e costituiscono punti equidistanti di spigoli di una superficie.

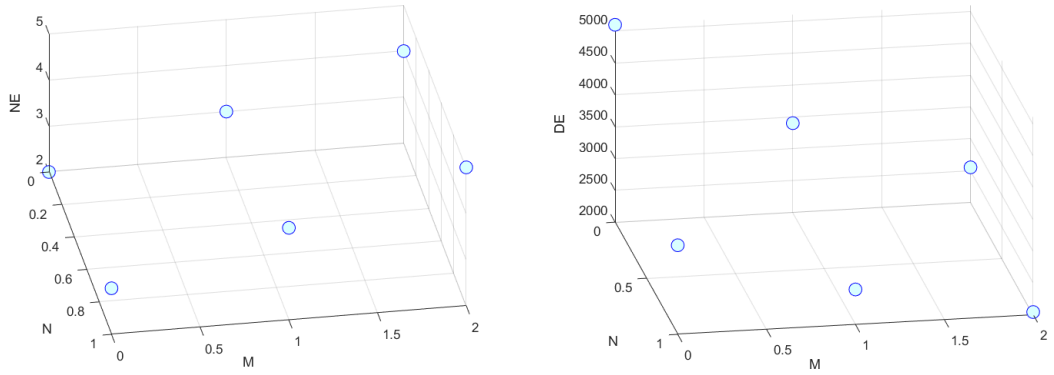


Figura 47 - Set di soluzioni NE e associati valori DE

Infine, non rimane che applicare i vincoli sul passo DE. Si inizializzano le matrici DE_{max} e DE_{min} di dimensioni compatibili con le matrici $\bar{\bar{X}}$ ed $\bar{\bar{Y}}$ e si verifica elemento per elemento che i vincoli siano rispettati secondo le equazioni definite nel paragrafo 3.2 (Figura 48).

In questa parte del codice è stato implementato un ciclo *for* per evitare eventuali divisioni per zero (riga 95).

```
89 -    DEmax=zeros(length(DE2(:,1)),length(DE2(1,:)));
90 -    DEmin=zeros(length(DE2(:,1)),length(DE2(1,:)));
91 -    DE1=[];
92      %vincoli sup. e inf.
93 -  for j=1:length(X(:,1))                    %iterazione sulle righe
94 -      for k=1:length(X(1,:))                %iterazione sulle colonne
95 -          if X(j,k)==0
96 -              if I<=U
97 -                  DEmax(j,k)=IS-pNCE-FC;
98 -                  DEmin(j,k)=(IS+pNCE)/2;
99 -              else
100 -                 DEmax(j,k)=(IS -pNCE);
101 -                 DEmin(j,k)=(pNCE+FS)/2;
102 -             end
103 -         else
104 -             if I<=U
105 -                 DEmax(j,k)=(IS -pNCE-FC)/(X(j,k));
106 -                 DEmin(j,k)=(IS+pNCE)/(X(j,k)+1);
107 -             else
108 -                 DEmax(j,k)=(IS -pNCE)/(X(j,k));
109 -                 DEmin(j,k)=(pNCE+FS)/(X(j,k)+1);
110 -             end
111 -         end
112         %se i vincoli sono rispettati aggiungi la soluzione ad un vettore
113 -         if DE2(j,k)<=DEmax(j,k) && DE2(j,k)>=DEmin(j,k)
114 -             DE1(end+1)=DE2(j,k);
115 -         end
116 -         if X(j,k)==0 && Y(j,k)==0
117 -             if L/2>IS+pNCE
118 -                 DE1(end+1)=L/2;
119 -             end
120 -         end
121 -     end
122 - end
```

Figura 48 - Confronto delle soluzioni con i vincoli

Nella riga 113 attraverso un confronto logico si verifica se il vincolo è soddisfatto e in caso positivo si inserisce il passo come soluzione in un nuovo vettore riga $\overline{DE1}$.

Nel vettore $\overline{DE1}$ potrebbero esserci più valori del passo uguali; pertanto, è necessario depurare i duplicati attraverso la funzione nativa *unique* e invertendo la formula del calcolo del passo si ottiene l'insieme di soluzioni al problema generale (Figura 49).

```
124 -   DE=unique(DE1);          %valori possibili in mm del passo tra gli equipaggi
125 -   NE=2*(L./DE);            %n° di equipggi totali possibili
```

Figura 49 - Set di soluzioni finali del problema

Nella riga 125 si è moltiplicato per 2 per ottenere il numero totale degli equipaggi e non il numero di coppie di equipaggi. I risultati del problema risultano:

$$\overline{NE} = [8 \quad 4]$$

i cui valori corrispondenti del passo (in mm) sono:

$$\overline{DE} = [2550 \quad 5100]$$

Cambiando il dato di input NCE e passando da 20 cassetti per equipaggio a 40 cassetti per equipaggio, l'algoritmo fornisce come risultato:

$$NE = 4$$

$$DE = 5100$$

Le soluzioni appena trovate generano tre possibili varianti del sistema, una variante di 8 equipaggi da 20 cassetti (8x20), una variante di 4 equipaggi da 20 cassetti (4x20) e una variante di 4 equipaggi da 40 cassetti (4x40).

Le configurazioni da confrontare per ottimizzare il sistema sono:

- 8x20;
- 4x20;
- 4x40.

Queste tre soluzioni verranno analizzate dal punto di vista cinematico per determinarne le prestazione sulla produttività (Figura 50, Figura 51 e Figura 52).

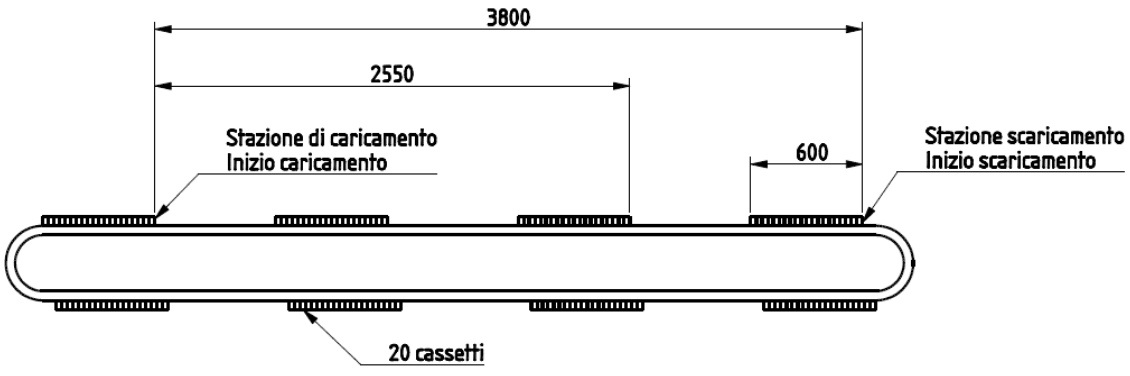


Figura 50 – Quote significative della configurazione 8x20

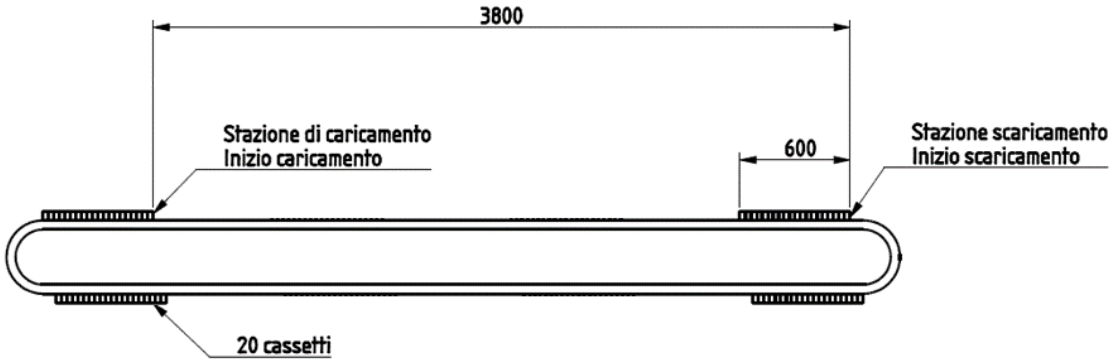


Figura 51 - Quote significative della configurazione 4x20

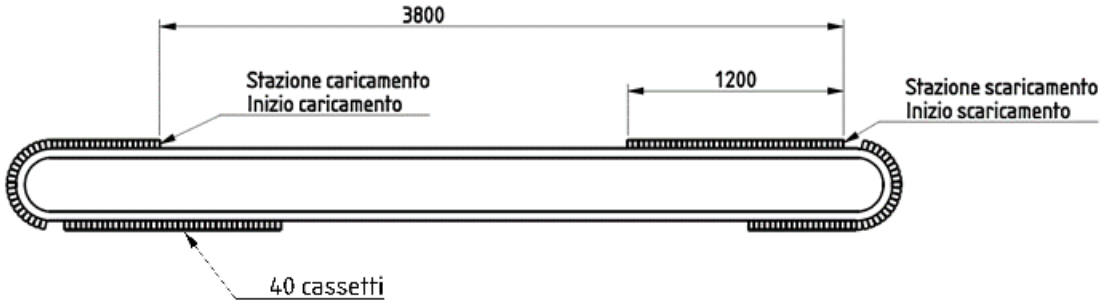


Figura 52 – Quote significative della configurazione 4x40

Oltre alle varianti generate dall'algoritmo si implementa nella simulazione anche il caso in cui il robot di caricamento preleva i prodotti con passo di 60 mm. In questo caso i caricamenti degli equipaggi vengono effettuati a cassetti alterni. Le configurazioni da simulare saranno in totale sei (Figura 53, Figura 54, Figura 55, Figura 56, Figura 57 e Figura 58).

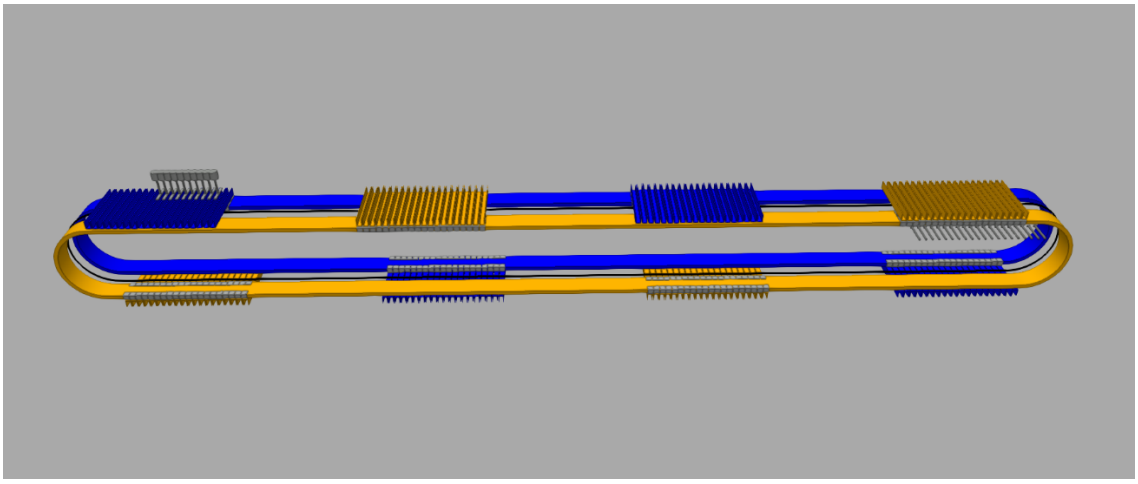


Figura 53 - Rappresentazione 3D della configurazione 8x20

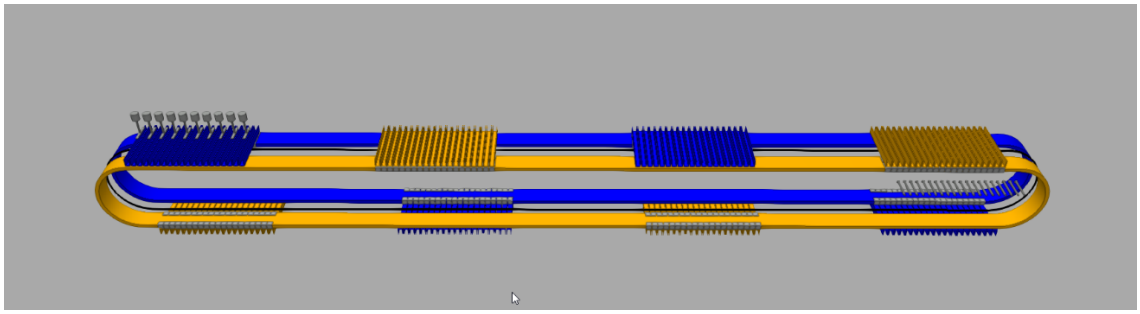


Figura 54 - Rappresentazione 3D della configurazione 8x20 con robot di caricamento a passo 60 mm

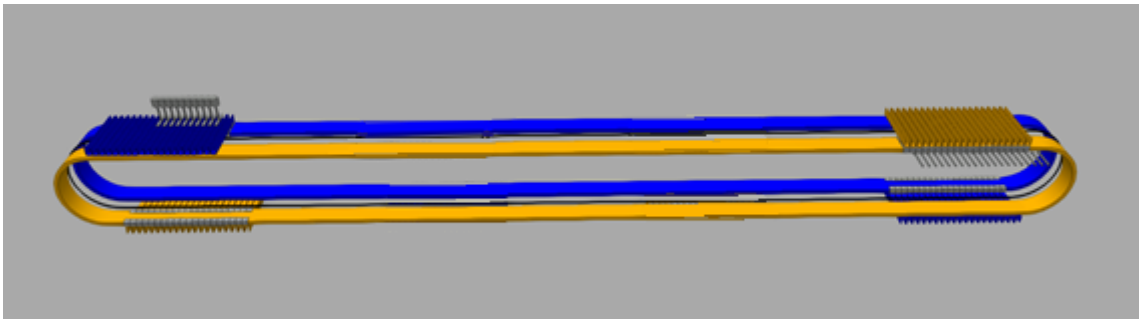


Figura 55 - Rappresentazione 3D della configurazione 4x20

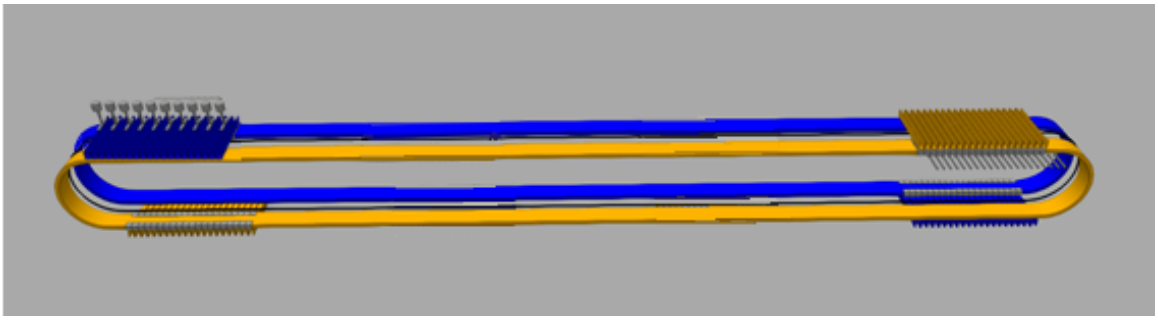


Figura 56 - Rappresentazione 3D della configurazione 4x20 con robot di caricamento a passo 60 mm

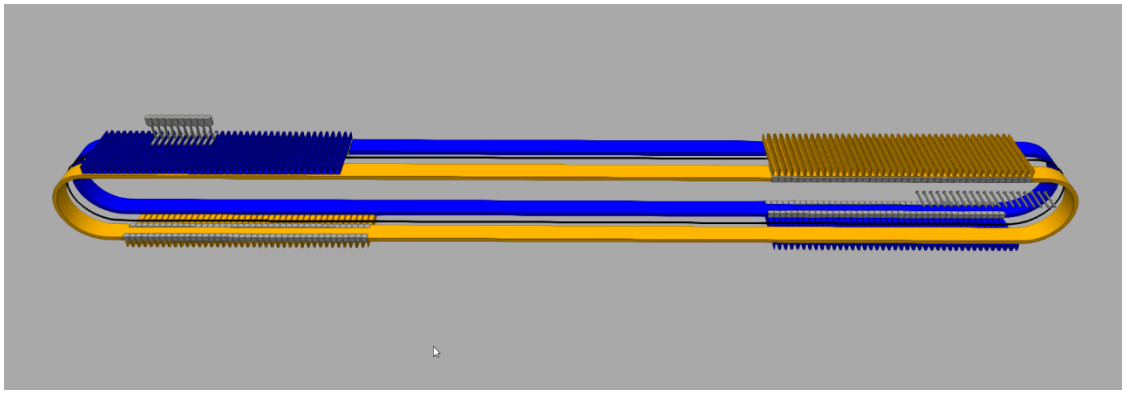


Figura 57 - Rappresentazione 3D della configurazione 4x40

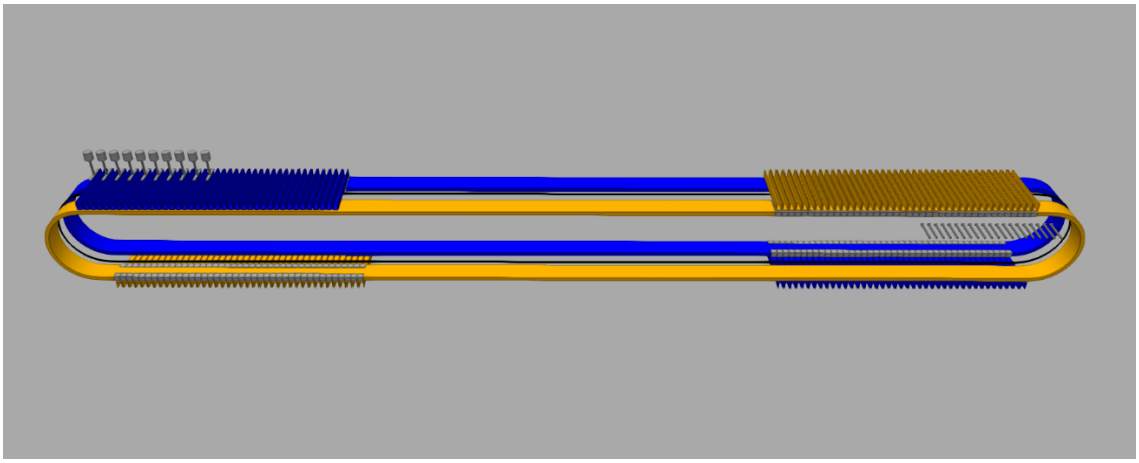


Figura 58 - Rappresentazione 3D della configurazione 4x40 con robot di caricamento a passo di 60 mm

3.4. Estensione dell'algoritmo per il calcolo delle varianti al variare della distanza tra le stazioni

L'algoritmo è in grado di calcolare le varianti del sistema considerando come vincoli la distanza tra le due stazioni e la distanza minima che gli equipaggi devono mantenere tra di loro.

Sulla distanza minima non è possibile apportare modifiche, ma è possibile eventualmente modificare la distanza tra le due stazioni IS. Se la distanza IS cambia, allora potrebbero cambiare anche le soluzioni fornite dall'algoritmo.

Se la distanza tra le stazione avesse una certa flessibilità di adattamento, si potrebbero confrontare il numero di configurazioni possibili per una distanza $(IS)_1$, per una distanza $(IS)_2$, ecc. Tuttavia, per fare questo confronto, si dovrebbe ripetere il calcolo delle configurazioni per ogni distanza IS-esima cambiando anche il dato di input NCE, cercando quindi la migliore configurazione fra tutte quelle possibili. Se il range di valori di IS risultasse sufficientemente ampio, le possibili combinazioni da calcolare diventano esageratamente grandi.

Attraverso una modifica all'algoritmo è possibile superare questo problema ed implementare il calcolo di tutte le configurazioni possibili al variare anche del parametro IS.

I dati di input rimangono esattamente gli stessi della versione precedente dell'algoritmo, in aggiunta si crea un vettore che contiene tutti i possibili valori che può assumere il parametro IS (Figura 59, riga 8).

```
 5 -    I=10;                  %n° prodotti ingresso per singola operazione robot
 6 -    U=20;                  %n° prodotti uscita per singola operazione robot
 7 -    NCE=20;                %n° cassetti in un equipaggio
 8 -    IS=1500:100:4400;      %distanza tra le stazioni di carico/scarico
 9 -    L=10200;               %lunghezza totale cinghia
10 -    p=30;                  %passo dei cassetti su un equipaggio
11 -    pNCE=p*NCE;            %lunghezza di ogni equipaggio di cassetti
```

Figura 59 - Input dell'algoritmo con il vettore IS

Il vettore $\overline{IS}$ è di dimensioni 1x30 (trenta possibili valori).

L'implementazione del metodo *divide and conquer* è uguale all'algoritmo precedente (Figura 60); tuttavia, in questo caso i parametri $\overline{N1}$ ed $\overline{M1}$ diventano dei vettori 1x30 poiché ereditano le dimensioni del vettore $\overline{IS}$ (Figura 61).

```
13 -    if I<=U
14 -        FC=(NCE/I-1)*p*I;                    %lunghezza zona di fine caricamento
15 -        N1=(IS-FC-pNCE)/(2*pNCE+FC);                %calcolo del numero di
16                                                      %coppie di equipaggi
17                                                      %possibili tra le due
18                                                      %stazioni di lavoro
19                                                      %considerando i vincoli
20
21 -        M1=(L-pNCE-IS-(2*pNCE+FC))/(2*pNCE+FC);     %calcolo delle restanti
22                                                      %coppie di equipaggi
23                                                      %possibili tranne le due
24                                                      %coppie presenti ai vincoli
25 -    else
26 -        FS=IS+(NCE/U -1)*p*U;               %lunghezza zona di fine scaricamento
27 -        N1=(IS-(FS-IS)-pNCE)/(2*pNCE+(FS-IS));
28 -        M1=(L-pNCE-FS-2*pNCE)/(2*pNCE+(FS-IS));
29 -    end
```

Figura 60 - Implementazione del metodo divide and conquer all'estensione dell'algoritmo

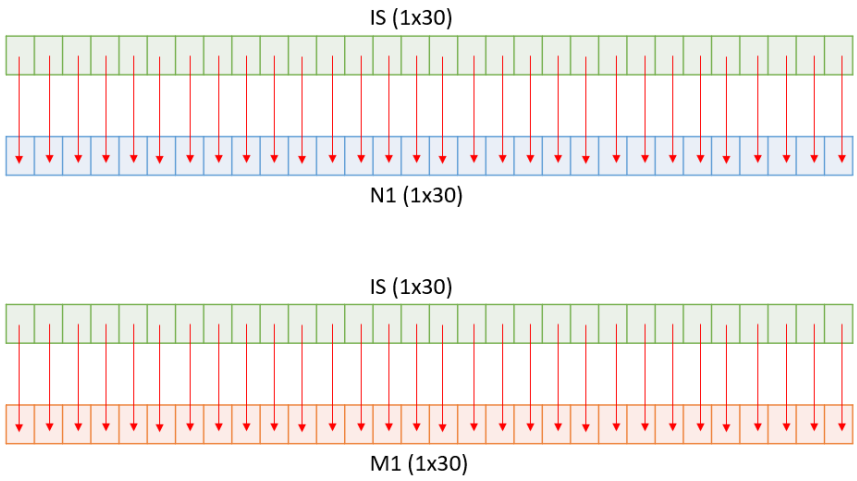


Figura 61 - Creazione dei vettori $N1$ ed $M1$

Nel caso precedente dell'algoritmo, era sufficiente creare dei vettori $\overline{N}$ ed $\overline{M}$ a partire dai valori massimi $N1$ ed $M1$, creando una serie di numeri interi dal valore 0 al valore $N1$ o $M1$.

In questo caso invece, i parametri $\overline{N1}$ ed $\overline{M1}$ sono già dei vettori; pertanto, per valutare tutte le possibili combinazioni è necessario creare delle matrici $\overline{\overline{N}}$ ed $\overline{\overline{M}}$. La matrice $\overline{\overline{N}}$ avrà tante righe quanti sono i valori della sequenza di numeri interi che vanno da 0 a $\max(\overline{N1})$ e la matrice $\overline{\overline{M}}$ avrà tante righe quanti sono i valori della sequenza di numeri interi che vanno da 0 a $\max(\overline{M1})$. Il numero di colonne di $\overline{\overline{N}}$ ed $\overline{\overline{M}}$ corrisponde al numero di colonne del vettore $\overline{IS}$ (Figura 62).

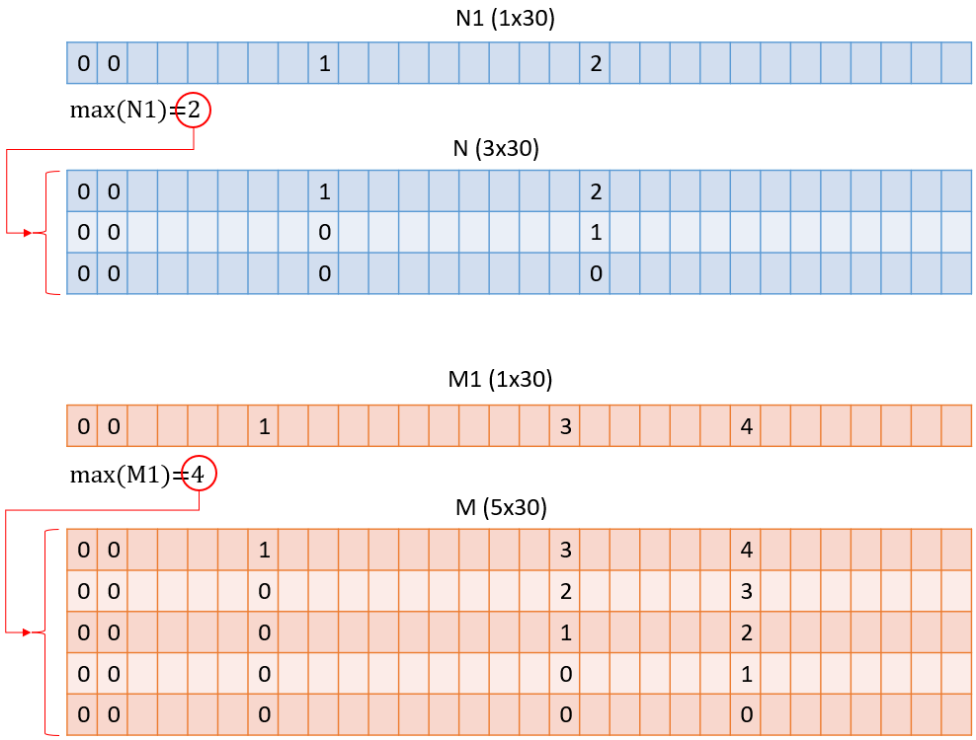


Figura 62 - Rappresentazione schematica della creazione delle matrici N ed M

Per creare queste matrici, come si può vedere dalla Figura 63, si creano due coppie di vettori ausiliari $\overline{N2}, \overline{N3}$ e $\overline{M2}, \overline{M3}$. I vettori $\overline{N2}, \overline{M2}$ (dimensione 1x30) servono ad arrotondare per difetto tutti i valori dei vettori $\overline{N1}, \overline{M1}$ in modo tale che si passi ad un dominio di numeri interi (righe 32-33). I vettori $\overline{N3}, \overline{M3}$ sono dei vettori che cambiano dimensione ad ogni iterazione di un ciclo *for*: per ogni elemento i-esimo di $\overline{N2}$ si crea un vettore riga $\overline{N3}_i$, i cui valori sono una sequenza di numeri interi che vanno da 0 al valore scalare $(\overline{N2})_i$ (riga 44). Attraverso un altro vettore ausiliario $\overline{N4}_i$ si esegue l'operazione di trasposizione del vettore $\overline{N3}_i$ (riga 45) che verrà usato per riempire ogni colonna i-

esima della matrice $\overline{\overline{N}}$ (riga 46). Per poter costruire la matrice complessiva $\overline{\overline{N}}$, ogni vettore $\overline{N3}_i$ deve avere la stessa dimensione, ma ogni vettore $\overline{N3}_i$ ha dimensioni diverse dagli altri. Per risolvere questo problema si inizializza la matrice $\overline{\overline{N}}$ come una matrice di zeri (riga 38) con il numero di colonne pari a $\max(\overline{N2})$, ed ogni colonna i-esima verrà rimpiazzata dal vettore $\overline{N3}_i$, che viene creato a partire dal valore $(\overline{N2})_i$ e discendendo fino al valore 0 (riga 44). Se il vettore $\overline{N3}_i$ ha dimensioni più piccole delle colonne della matrice $\overline{\overline{N}}$, i restanti elementi rimarranno pari a zero.

Analogamente viene fatto per la matrice M (righe 48-52).

```
32 -    N2=floor(N1);              %si approssima per difetto all'intero più vicino
33 -    M2=floor(M1);
34 -    szN1=size(N1);
35 -    szM1=size(M2);
36 -    n=max(N2);            %massimo n° di equipaggi possibili tra le due stazioni
37 -    m=max(M2);            %massimo n° di equipaggi possibili restanti
38 -    N=zeros(n,szN1(2));       %inizializzazione matrice di tutte le possibili
39                                %combinazioni di coppie di equipaggi di N2 con M2
40 -    M=zeros(m,szM1(2));       %inizializzazione matrice di tutte le possibili
41                                %combinazioni di coppie di equipaggi di M2 con N2
42
43 -  for i=1:szN1(1,2)
44 -      N3=max(N2(1,i)):-1:0;
45 -      N4=N3';
46 -      N(1:length(N3),i)=N4;
47 -  end
48 -  for i=1:szM1(1,2)
49 -      M3=max(M2(1,i)):-1:0;
50 -      M4=M3';
51 -      M(1:length(M3),i)=M4;
52 -  end
```

Figura 63 – Creazione delle matrici N ed M

Per eseguire l'*exhaustive search* in modo efficiente si creano le matrici $\overline{\overline{X}}$ ed $\overline{\overline{Y}}$ (Figura 64). Sono matrici omogenee in modo tale da poter effettuare le operazioni algebriche tenendo conto di tutte le combinazioni di valori.

```
54 -    [X,Y]=meshgrid(N,M);
55 -    szX=size(X);
56 -    szN=size(N);
```

Figura 64 - Creazione delle matrici omogenee X ed Y

La funzione *meshgrid* prende in input le due matrici $\bar{\bar{N}}$ ed $\bar{\bar{M}}$, che rispettivamente hanno dimensione (3x30) e (5x30) e creano due matrici $\bar{\bar{X}}$ ed $\bar{\bar{Y}}$ di dimensioni (150x90), ovvero ci saranno tante righe quanti sono gli elementi di $\bar{\bar{M}}$ (Figura 65) e ci saranno tante colonne quanti sono gli elementi di $\bar{\bar{N}}$ (Figura 66).

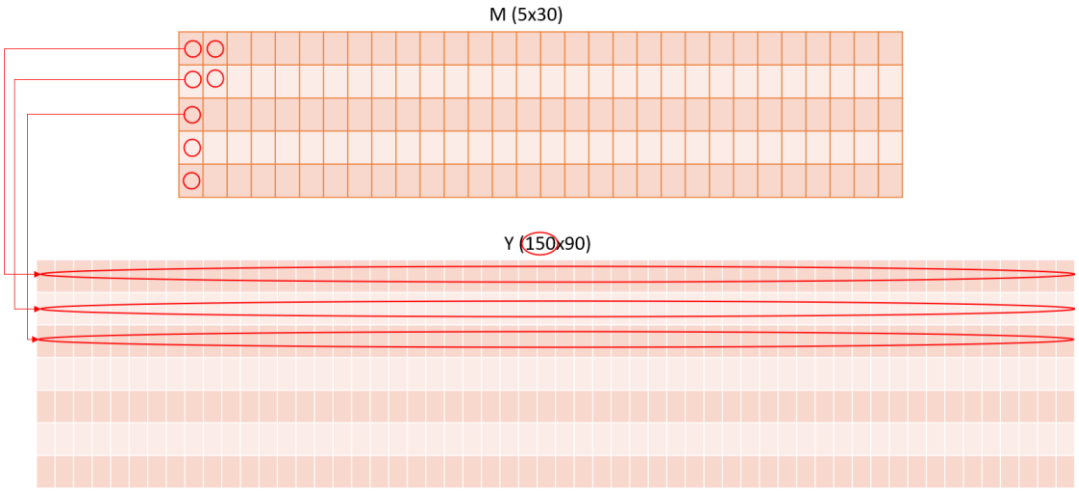


Figura 65 - Creazione delle matrici X e Y, numero di righe

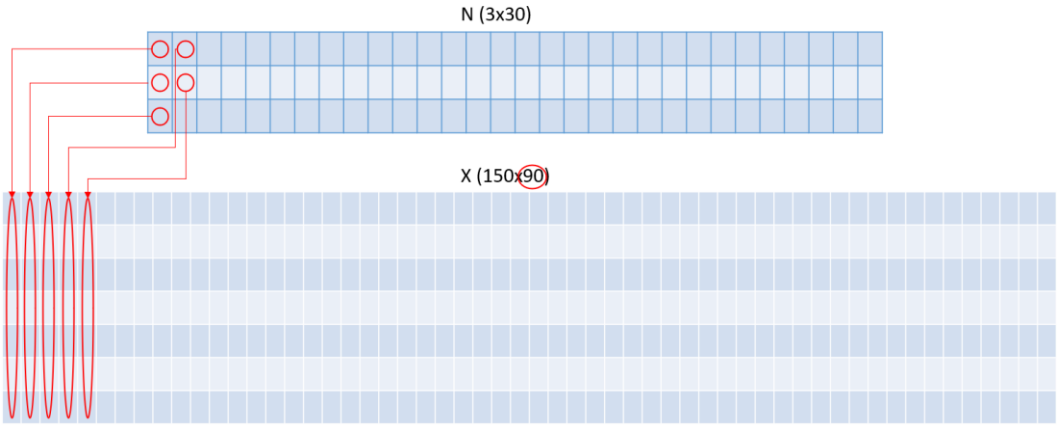


Figura 66 - Creazione delle matrici X e Y, numero di colonne

La matrice $\bar{\bar{X}}$ è composta solo da elementi della matrice $\bar{\bar{N}}$, ogni colonna di $\bar{\bar{X}}$ è la ripetizione di un elemento della matrice $\bar{\bar{N}}$, seguendo la sequenza:

$$N(1,1); N(2,1); N(3,1); N(1,2); N(2,2); \dots$$

Essendo $\max(\bar{\bar{N}}) = 1$, alcuni elementi della matrice $\bar{\bar{X}}$ sono:

$$\bar{\bar{X}} = \begin{bmatrix} 0 & \dots & 0 & 1 & \dots & 0 & \dots & 1 \\ \vdots & \ddots & \dots & \dots & \dots & \dots & \dots & \vdots \\ 0 & \dots & 0 & 1 & \dots & 0 & \dots & 1 \\ \vdots & \dots & \dots & \dots & \dots & \dots & \dots & \vdots \end{bmatrix}$$

La matrice $\bar{\bar{Y}}$ è composta invece solo da elementi della matrice $\bar{\bar{M}}$, ogni riga di $\bar{\bar{Y}}$ è la ripetizione di un elemento della matrice $\bar{\bar{M}}$, seguendo la sequenza:

$$M(1,1); M(2,1); M(3,1); M(5,1); M(5,1); M(1,2); M(2,2); \dots$$

Essendo $\max(\bar{\bar{M}}) = 4$, alcuni elementi della matrice $\bar{\bar{Y}}$ sono:

$$\bar{\bar{Y}} = \begin{bmatrix} 4 & \dots & 4 & \dots & \dots & 4 \\ 3 & \ddots & 3 & \dots & \dots & 3 \\ 2 & \dots & 2 & \dots & \dots & 2 \\ 1 & \dots & 1 & \dots & \dots & 1 \\ 0 & \dots & 0 & \dots & \dots & 0 \\ 4 & \dots & 4 & \dots & \dots & 4 \\ \vdots & \dots & \dots & \dots & \dots & \vdots \\ \vdots & \dots & \dots & \dots & \dots & \vdots \\ 3 & \dots & 3 & \dots & \dots & 3 \\ 2 & \dots & 2 & \dots & \dots & 2 \\ \vdots & \dots & \dots & \dots & \dots & \vdots \end{bmatrix}$$

Sommando le due matrici $\bar{\bar{X}}$ ed $\bar{\bar{Y}}$ si ottiene una matrice $\overline{\overline{NE1}}$ di dimensione (150x90), cui ogni elemento è una combinazione diversa di somma tra gli elementi di $\bar{\bar{N}}$ ed $\bar{\bar{M}}$.

Per ogni combinazione otteniamo anche la matrice $\overline{\overline{DE1}}$ cui ogni elemento costituisce il passo degli equipaggi (Figura 67).

```
86 -    NE1=X+Y+2;                                  %n° totale di coppie di equipaggi
95 -    DE1=L./NE1;                                 %passo degli equipaggi equidistri
```

Figura 67 – Calcolo di tutti gli equipaggi (possibili e non) e il loro rispettivo passo

Per poter valutare quali sono gli elementi della matrice $\overline{\overline{NE1}}$ che soddisfano i requisiti di progetto, è necessario confrontarli con i vincoli. Nel confrontare gli elementi di $\overline{\overline{NE1}}$ con i vincoli, è utile tenere traccia di quale elemento è associato all'elemento IS-esimo.

In questo modo una volta ottenuti i risultati accettabili, si mantiene l'informazione che associa i risultati al valore di distanza $(IS)_i$ sul quale è stato calcolato.

Per adoperare questo metodo in modo efficiente, allo stesso modo di prima si crea una matrice ausiliaria $\overline{\overline{IS1}}$ delle stesse dimensioni di $\bar{\bar{X}}$ e $\bar{\bar{Y}}$, ovvero di dimensioni (150x90).

Attraverso la funzione nativa *repmat* si riempie ogni elemento di una colonna di $\overline{\overline{IS1}}$ ripetendo in sequenza ognuno degli elementi di $\overline{IS}$. La ripetizione degli elementi segue una precisa logica (Figura 68).

```
73 -    b=1;
74 -    c=1;
75 -    IS1=zeros(szX(1,1),szX(1,2));
76 -    for a=1:szX(1,2)
77 -        IS1(:,a)=repmat(IS(c),szX(1,1),1);
78 -        b=b+1;
79 -        if b>szN(1,1)
80 -            c=c+1;
81 -            b=1;
82 -        end
83 -    end
```

Figura 68 - Creazione matrice $IS1$

Tale logica si basa sulla struttura della matrice $\bar{\bar{X}}$, poiché nell'applicare i vincoli si fa affidamento proprio alla matrice $\bar{\bar{X}}$. La matrice $\bar{\bar{X}}$ viene generata a partire dalla matrice $\bar{\bar{N}}$ che a sua volta viene generata a partire dal vettore $\overline{N1}$ e dal vettore $\overline{IS}$. Seguendo il workflow mostrato in Figura 69, la matrice $\bar{\bar{N}}$ ha dimensioni (3x30) e ogni colonna di $\bar{\bar{N}}$ è correlata ad un elemento del vettore $\overline{IS}$; inoltre, ogni elemento della colonna i-esima di $\bar{\bar{N}}$ viene utilizzato per riempire una colonna della matrice $\bar{\bar{X}}$. In sostanza, le prime tre colonne della matrice $\bar{\bar{X}}$ sono correlate con il primo elemento di $\overline{IS}$, le successive tre colonne della matrice $\bar{\bar{X}}$ saranno correlate con il secondo elemento di $\overline{IS}$ e così via.

Attraverso il codice mostrato in Figura 68 si riempie la matrice $\overline{\overline{IS1}}$ inserendo l'elemento $(IS)_1$ in ogni riga delle colonne $IS1(:,1)$; $IS1(:,2)$; $IS1(:,3)$, successivamente si inserisce l'elemento $(IS)_2$ in ogni riga delle colonne $IS1(:,4)$; $IS1(:,5)$; $IS1(:,6)$ e così proseguendo.

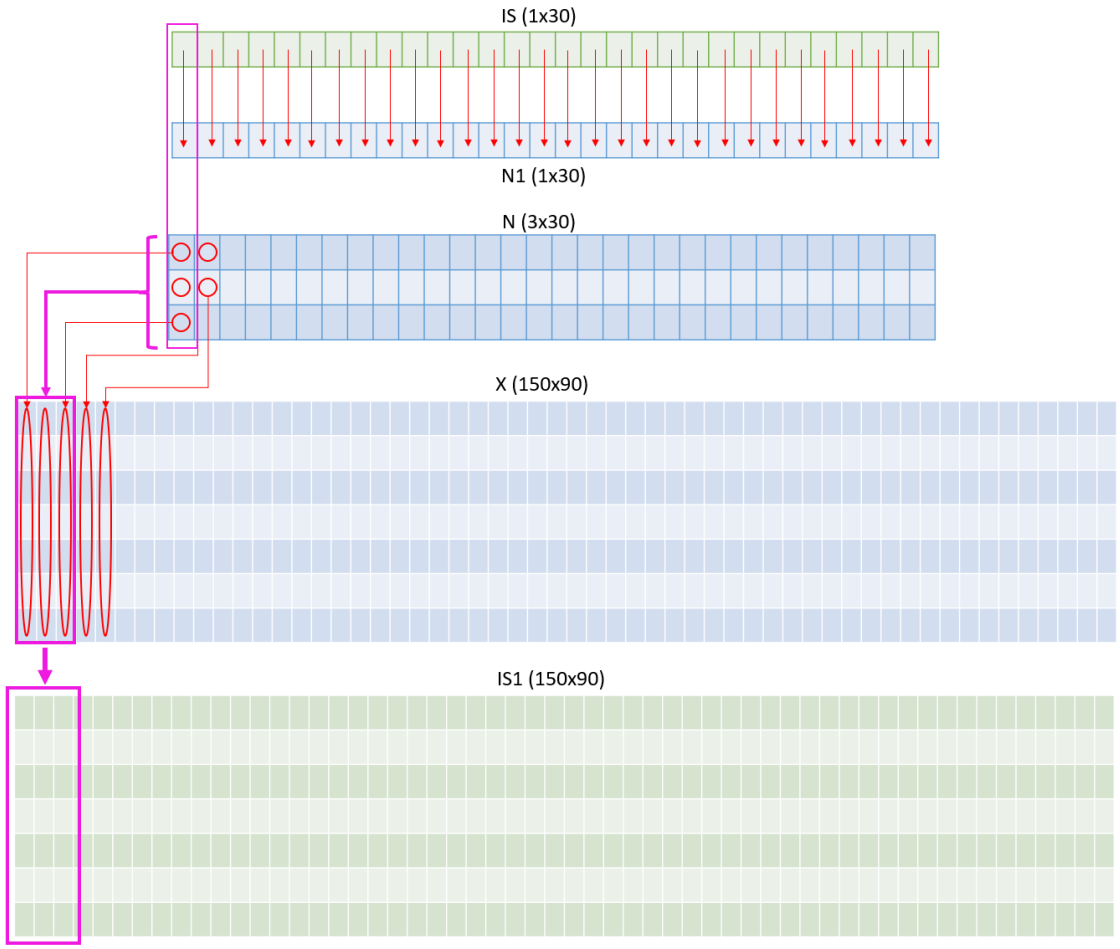


Figura 69 - Workflow di creazione della matrice $IS1$

Per scremare l'insieme delle soluzioni e ottenere quelle accettabili si applicano i vincoli. Questa operazione viene eseguita sulla matrice $\overline{\overline{DE1}}$ (Figura 70).

Ad ogni ciclo di calcolo vengono memorizzati i valori delle soluzioni accettabili nei vettori $\overline{DE2}, \overline{IS2}$. In questo modo, ad ogni elemento j-esimo di $\overline{DE2}$ conosciamo l'elemento j-esimo $\overline{IS2}$ a cui è associato.

```
104 -    DEmax=zeros(length(DE1(:,1)),length(DE1(1,:)));
105 -    DEmin=zeros(length(DE1(:,1)),length(DE1(1,:)));
106 -    DE2=[];
107 -    IS2=[];
108      %vincoli sup. e inf.
109 -  for j=1:length(X(:,1))                    %iterazione sulle righe
110 -      for k=1:length(X(1,:))                %iterazione sulle colonne
111 -          if X(j,k)==0
112 -              if I<=U
113 -                  DEmax(j,k)=IS1(j,k)-pNCE-FC;
114 -                  DEmin(j,k)=(IS1(j,k)+pNCE)/2;
115 -              else
116 -                  DEmax(j,k)=(IS1(j,k)-pNCE);
117 -                  DEmin(j,k)=(FS+pNCE)/2;
118 -              end
119 -          else
120 -              if I<=U
121 -                  DEmax(j,k)=(IS1(j,k)-pNCE-FC)/(X(j,k));
122 -                  DEmin(j,k)=(IS1(j,k)+pNCE)/(X(j,k)+1);
123 -              else
124 -                  DEmax(j,k)=(IS1(j,k)-pNCE)/(X(j,k));
125 -                  DEmin(j,k)=(FS+pNCE)/(X(j,k)+1);
126 -              end
127 -          end
128          %se i vincoli sono rispettati aggiungi la soluzione ad un vettore
129 -          if DE1(j,k)<=DEmax(j,k) && DE1(j,k)>=DEmin(j,k)
130 -              DE2(end+1)=DE1(j,k);
131 -              IS2(end+1)=IS1(j,k);
132 -          end
133 -          if X(j,k)==0 && Y(j,k)==0
134 -              if L/2>IS1(j,k)+pNCE
135 -                  DE2(end+1)=L/2;
136 -                  IS2(end+1)=IS1(j,k);
137 -              end
138 -          end
139 -      end
140 -  end
```

Figura 70 - Confronto degli elementi della matrice $DE1$ con i vincoli

L'insieme di soluzioni accettabili è mostrato in Figura 71. Nel vettore $\overline{DE2}$ potrebbero esserci più valori del passo uguali; pertanto, è necessario depurare i duplicati attraverso la funzione nativa *unique* (riga 142) e invertendo la formula del calcolo del passo (riga 143), si ottiene l'insieme delle soluzioni accettabili del problema generale.

```
142 -    DE=unique(DE2);     %valori possibili dei passi tra gli equipaggi
143 -    NE=2*(L./DE);      %n° di equipggi totali possibili
```

Figura 71 - Set di soluzioni accettabili del problema

Con gli input di Figura 59, si ottengono le soluzioni mostrate in Figura 72 e Figura 73.

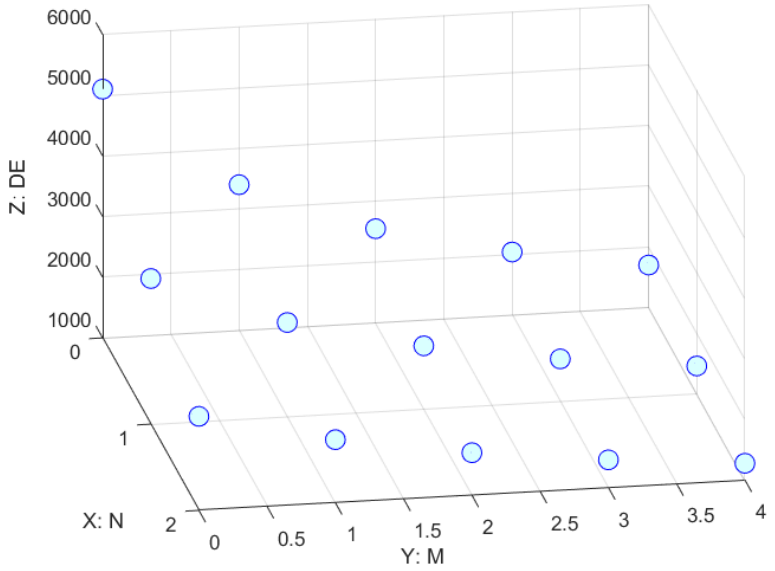


Figura 72 - Soluzioni accettabili del problema: valori del passo al variare dei parametri N e M

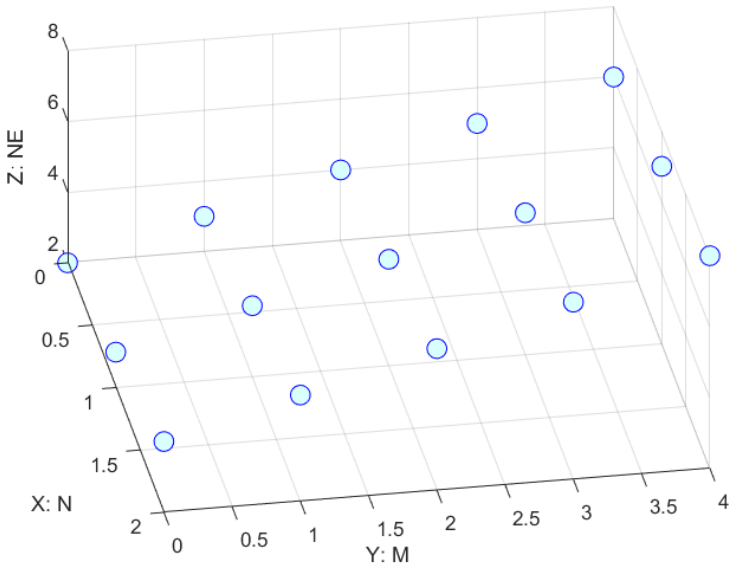


Figura 73 - Soluzioni accettabili del problema: valori del numero totale di equipaggi al variare dei parametri N e M

Infine, esplicitando l'informazione che correla le soluzioni al parametro $\overline{IS}$ (Figura 74), possiamo creare una matrice $\overline{\overline{Comb1}}$ che raccoglie ogni soluzione, associando ad ogni elemento di $\overline{NE}$ il suo elemento $\overline{DE}$ e il suo elemento $\overline{IS}$ (Figura 75).

```
145 -    w=size(DE2);
146 -    Comb1=zeros(w(1,2),3);
147 -    Comb1(:,1)=IS2';
148 -    Comb1(:,2)=2*(L./DE2);
149 -    Comb1(:,3)=DE2';
150 -    Comb=unique(Comb1,'rows');  %n° totale di combinazioni soluzione del problema
```

Figura 74 - Set di soluzioni accettabili al variare del parametro IS

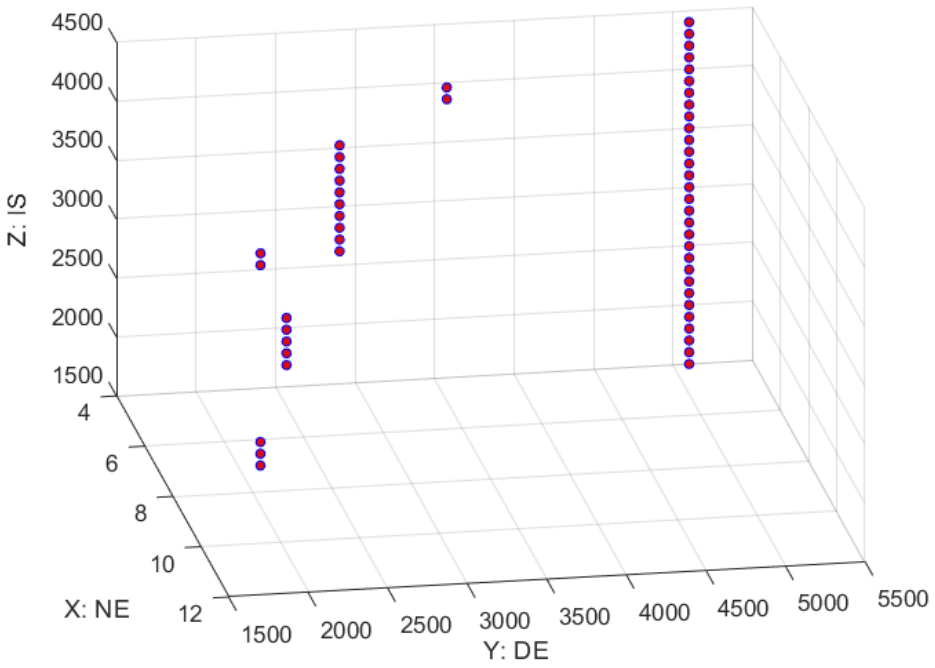


Figura 75 - Combinazioni possibili delle configurazioni del sistema

In Figura 76 si riassume in breve il numero di equipaggi accettabili al variare del parametro $\overline{IS}$.

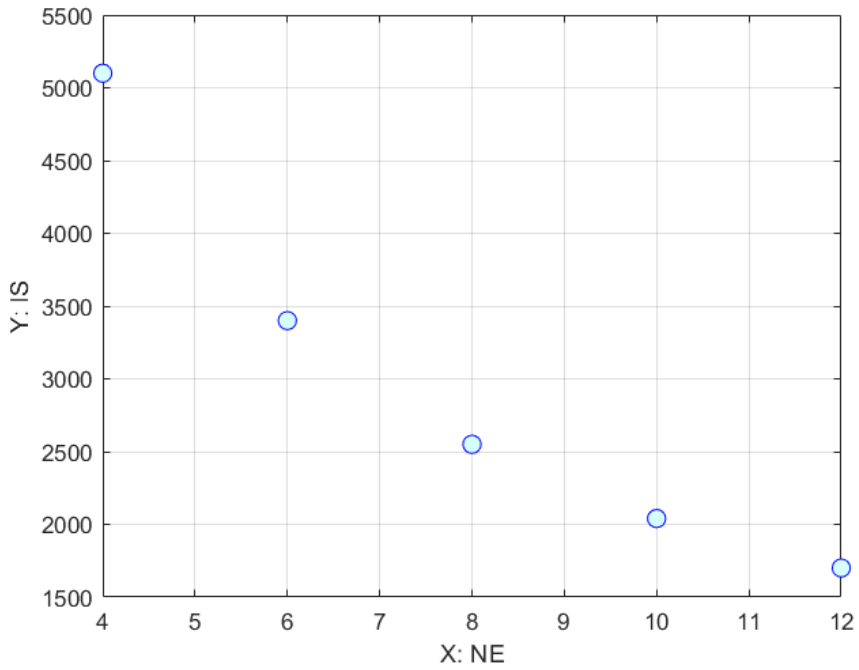


Figura 76 - Soluzioni accettabili al variare del parametro IS

4. Analisi cinematica

4.1. Carta delle operazioni

Per poter valutare le prestazione di una configurazione è necessario valutare se essa è in grado di soddisfare la produttività richiesta o eventualmente se è in grado di fornire una produttività maggiore di quella di progetto.

Un altro parametro da valutare per definire la bontà di una configurazione è lo sforzo richiesto ai motori per poter svolgere il trasferimento di un equipaggio da una stazione ad un'altra, la condizione migliore è ovviamente quella in cui le leggi di moto degli equipaggi hanno velocità e accelerazione massime che siano le più basse possibile. Un metodo per affrontare questa valutazione è quello di utilizzare la carta delle operazioni [17] [18].

Questo strumento permette di organizzare in maniera cronologica il susseguirsi di operazioni che sono necessarie per completare un'unità produttiva. Per ogni operazione viene annotato il tempo impiegato per il completamento di essa e viene rappresentato graficamente, si impilano assieme tutti i tempi e si ottiene il tempo totale impiegato per il completamento di una unità produttiva.

Questo strumento è impiegato solitamente per saturare il lavoro da svolgere di un operatore all'interno di una cella produttiva. Se ad esempio una cella produttiva è costituita da due macchine, l'operatore dovrà rifornire in tempo la materia prima alle due macchine per evitare che queste rimangano in ozio. Se le due macchine hanno tempo di lavorazione differente allora l'operatore dovrà spostarsi tra le due macchine seguendo una precisa organizzazione in modo tale che egli sia sincronizzato perfettamente con almeno una delle due macchine riducendo i tempi di ozio delle macchine di cella al minimo possibile. Così facendo, dopo un periodo iniziale di warm-up, si determina una sequenza di operazioni ripetitiva, sempre uguale nel tempo, che costituirà il tempo ciclo della cella.

In questo lavoro di tesi, si adatta questo strumento sostituendo il sistema sincro-dinamico all'operatore. Infatti, il compito del sistema sincro-dinamico è quello di sincronizzare la macchina a monte e la macchina a valle attraverso una ben congegnata

sequenza di operazioni. Infatti, queste due macchine presentano un tempo di lavorazione diverso ed è necessario adottare una corretta sequenza di movimenti per sincronizzarle.

Questo strumento è una simulazione del processo produttivo basata esclusivamente sui tempi di lavorazione delle macchine. Il vantaggio di questo strumento è quello di procurare una visione di insieme del processo, in modo da poter valutare le prestazione del processo e gli aspetti migliorabili. D'altra parte, lo svantaggio di questo strumento è che costituisce una operazione manuale e tediosa, spesso suscettibile ad errori di disattenzione.

Per migliorare l'utilizzo di questo strumento si utilizza un software di emulazione virtuale del processo che permette la realizzazione automatica della carta delle operazioni. Questa simulazione permetterà di rendere più veloce la realizzazione della carta, soprattutto se è necessario realizzarla per diverse configurazioni di macchina.

In questa fase del progetto le informazioni in nostro possesso ci permettono solo di stabilire i tempi di lavorazione totali delle stazioni di carico e scarico e i tempi di sosta degli equipaggi nelle stazioni.

Ricordiamo che la stazione di carico impiega 1,2 secondi come tempo di lavorazione per caricare 10 prodotti, di cui 0,8 secondi per depositare il prodotto nell'equipaggio e 0,4 secondi per prelevare il prodotto e movimentare il robot. La stazione di scarico impiega 0,8 secondi per scaricare 20 prodotti.

Sia in caricamento che in scaricamento l'equipaggio presenta un tempo di sosta di 0,8 secondi.

Scegliamo le configurazioni generate dall'algoritmo per la distanza $IS = 3800\ mm$.

- Configurazione 8x20: ogni equipaggio è costituito da 20 cassetti per una lunghezza totale di 600 mm; pertanto, sono necessarie due operazioni di caricamento per completare un equipaggio e soltanto una operazione di scaricamento per svuotarlo.

  Per garantire una produttività di progetto di 500 prodotti al minuto, è necessario eseguire lo scarico di 20 prodotti ogni 2,4 secondi:

$$Produttività = 500\ \left[\frac{prodotti}{min}\right] = 8{,}3\overline{3}\ \left[\frac{prodotti}{s}\right]$$

$$N°\ tot\ prodotti\ da\ scaricare = 1 \cdot 20\ \left[\frac{prodotti}{operazione}\right] = 20\ \left[\frac{prodotti}{operazione}\right]$$

$$T_{sc} = \frac{Scarico}{8{,}3\overline{3}} \frac{\left[\frac{prodoti}{operazione}\right]}{\left[\frac{prodotti}{s}\right]} = 2{,}4 \left[\frac{s}{operazione}\right]$$

In altre parole, ogni 2,4 secondi bisogna eseguire almeno una operazione di scarico. Se ogni 2,4 secondi bisogna scaricare completamente un equipaggio, per mantenere la continuità del flusso di prodotti negli equipaggi è necessario che ogni 2,4 secondi si riempia un altro equipaggio. Di conseguenza, ogni 2,4 secondi è necessario eseguire almeno due operazioni di carico per completare un equipaggio; essendo il tempo totale di una operazione di carico pari a 1,2 secondi, si deduce che la stazione di carico deve lavorare in completa saturazione per poter garantire la produttività di progetto.

Con queste informazioni possiamo già stilare una bozza della carta operazioni come mostrato in Figura 77.

In figura si può vedere come la stazione di caricamento lavora senza sosta mentre la stazione di scaricamento ha diversi spazi vuoti che permettono una flessibilità notevole su quando eseguire l'operazione di scarico nella scala temporale. Dopo aver caricato completamente un equipaggio (due operazioni) si procedere nel caricare un equipaggio appartenente all'altra cinghia di movimentazione.

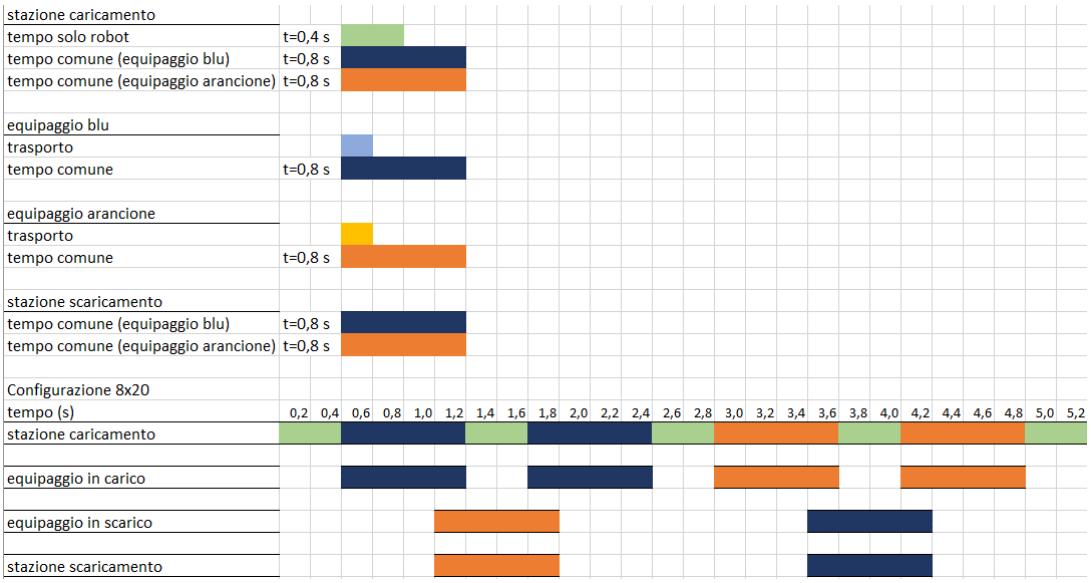


Figura 77 - Bozza carta delle operazioni della configurazione 8x20

Nella bozza della carta, la prima operazione di scarico dell'equipaggio arancione è stata arbitrariamente posizionata all'istante $t = 1{,}0\ s$, mentre la prima operazione di carico dell'equipaggio blu è obbligatoriamente da posizionarsi all'istante $t = 0{,}4\ s$. La scelta di caricare prima l'equipaggio blu e di scaricare prima l'equipaggio arancione non ha nessun effetto sul redigere la carta delle operazioni e può essere del tutto arbitraria.

Questa condizione di saturazione purtroppo è svantaggiosa per il sistema sincro-dinamico. Se ci fossero dei ritardi a monte, che comporterebbero lo svolgersi di una operazione di carico in un tempo maggiore di 1,2 secondi, il sistema sincro-dinamico non può fare nulla per recuperare tale ritardo visto la saturazione completa della stazione.

- Configurazione 4x20: dal punto di vista dei tempi di carico/scarico, questa configurazione è uguale alla configurazione 8x20.

  Per garantire una produttività di progetto di 500 prodotti al minuto, è necessario eseguire lo scarico di 20 prodotti ogni 2,4 secondi:

$$Produttività = 500\ \left[\frac{prodotti}{min}\right] = 8{,}3\bar{3}\ \left[\frac{prodotti}{s}\right]$$

$$N°\ tot\ prodotti\ da\ scaricare = 1 \cdot 20\ \left[\frac{prodotti}{operazione}\right] = 20\ \left[\frac{prodotti}{operazione}\right]$$

$$T_{sc} = \frac{Scarico}{8{,}3\bar{3}}\frac{\left[\frac{prodoti}{operazione}\right]}{\left[\frac{prodotti}{s}\right]} = 2{,}4\ \left[\frac{s}{operazione}\right]$$

  Anche in questo caso la stazione di carico deve lavorare in completa saturazione per poter garantire la produttività di progetto.

  In Figura 78 è mostrata la bozza della carta delle operazioni della configurazione 4x20.

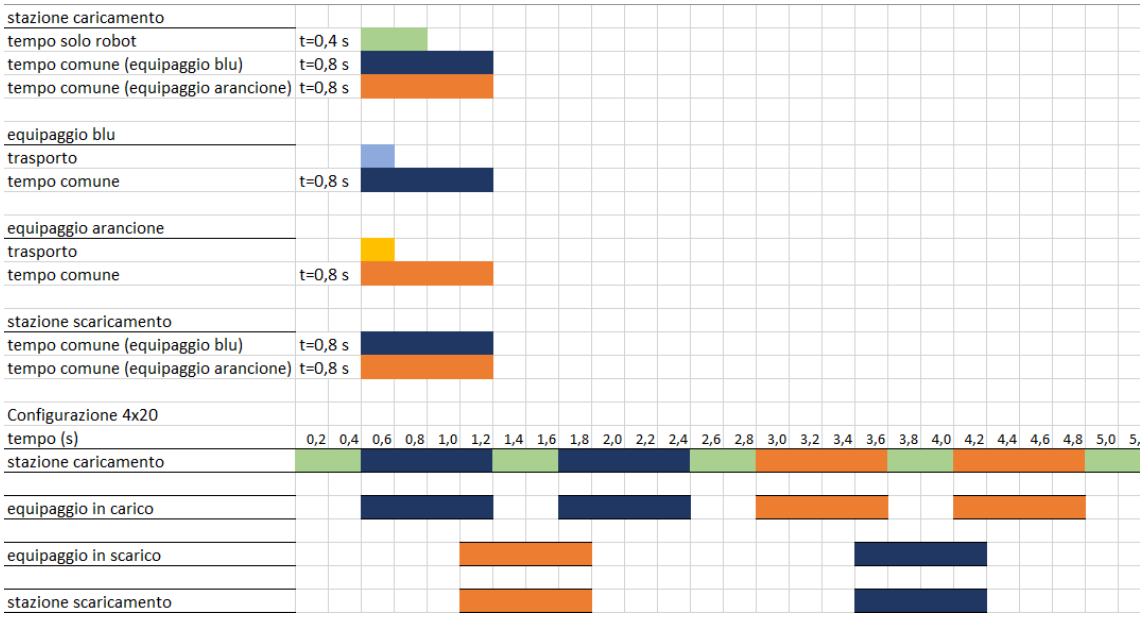


Figura 78 - Bozza carta delle operazioni della configurazione 4x20

- Configurazione 4x40: ogni equipaggio è costituito da 40 cassetti per una lunghezza totale di 1200 mm; sono necessarie quattro operazioni di carico e due operazioni di scarico per completare ogni equipaggio.
  Per garantire una produttività di progetto di 500 prodotti al minuto, è necessario eseguire lo scarico di 40 prodotti ogni 4,8 secondi:

$$Produttività = 500 \left[\frac{prodotti}{min}\right] = 8{,}3\overline{3} \left[\frac{prodotti}{s}\right]$$

$$N° \, tot \, prodotti \, da \, scaricare = 2 \cdot 20 \left[\frac{prodotti}{operazione}\right] = 40 \left[\frac{prodotti}{operazione}\right]$$

$$T_{sc} = \frac{Scarico}{8{,}3\overline{3}} \frac{\left[\frac{prodoti}{operazione}\right]}{\left[\frac{prodotti}{s}\right]} = 4{,}8 \left[\frac{s}{operazione}\right]$$

  In altre parole, ogni 4,8 secondi bisogna eseguire almeno due operazioni di scarico. Se ogni 4,8 secondi bisogna scaricare completamente un equipaggio, per mantenere la continuità del flusso di prodotti negli equipaggi è necessario che ogni 4,8 secondi si riempia un altro equipaggio. Di conseguenza, ogni 4,8 secondi è necessario eseguire almeno quattro operazioni di carico per

completare un equipaggio. Essendo il tempo totale di una operazione di carico pari a 1,2 secondi, si deduce che la stazione di carico anche in questa configurazione deve lavorare in completa saturazione per poter garantire la produttività di progetto.

In Figura 79 è stata stilata la carta operazioni della configurazione 4x40. In questa configurazione le operazioni da svolgere su un equipaggio in caricamento o scaricamento raddoppiano rispetto alla configurazione precedente, ma dal confronto di queste due bozze non si riesce a determinare differenze sostanziali tali da far preferire una configurazione piuttosto che un'altra.

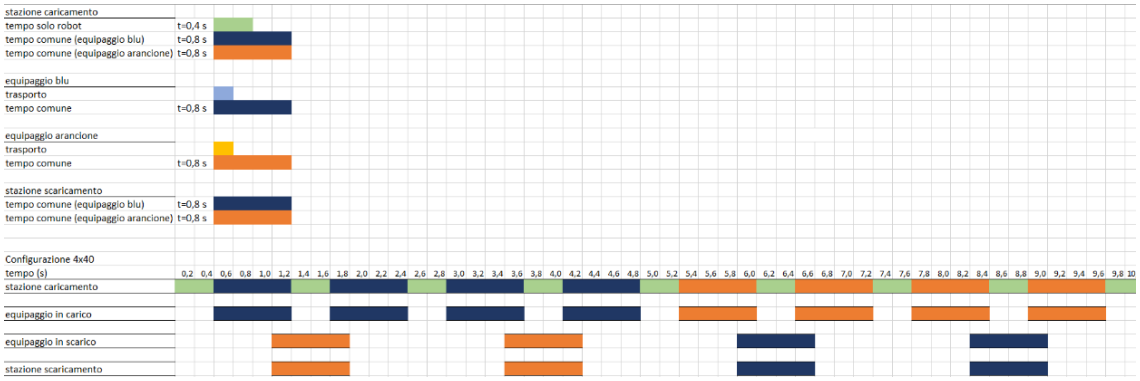


Figura 79 - Bozza carta operazioni della configurazione 4x40

Le altre tre configurazioni del sistema con il robot di caricamento a passo di 60 mm non presentano differenze nelle carte delle operazioni delle rispettive configurazioni.

Queste tre carte delle operazioni vanno completate inserendo i tempi impiegati dagli equipaggi per spostarsi da una stazione ad un'altra. Le sei configurazioni da questo punto di vista presenteranno delle differenze sostanziali, poiché le lunghezze degli equipaggi e il loro numero sono differenti, oppure sono differenti le distanze da percorrere durante le operazioni di caricamento.

I tempi di spostamento andranno a riempire gli spazi vuoti lasciati nella carta operazioni e di conseguenza trasleranno le operazioni di scaricamento in maniera coniugata. Dopo un periodo iniziale di warm-up si arriverà ad una sequenza di operazioni ripetitive uguali in cui si sarà determinato un tempo ciclo di cella e tale calcolo verrà affrontato nella simulazione virtuale del sistema sincro-dinamico.

Per calcolare i tempi di spostamento è necessario svolgere un'analisi cinematica della macchina in tutte le sue configurazioni, andando a determinare quali sono le leggi di moto degli equipaggi e implementando tali leggi nella simulazione virtuale del processo.

4.2. Scelta della legge di moto

Il sistema che si sta progettando, per quanto riguarda gli azionamenti, è costituito da due motori uguali, ognuno montato su una puleggia che aziona una delle due cinghie. Da prove sperimentali in azienda si è notato che se la variazione delle accelerazioni è troppo repentina e se l'accelerazione della cinghia è troppo ampia, la forza di inerzia dei cassetti risulta sufficientemente elevata da provocare dei fenomeni vibratori che si concludono con l'impuntamento dei cassetti o in alcuni casi si è potuto constatare che i prodotti contenuti nei cassetti rischiano di sollevarsi dal piano in cui sono ospitati, perdendo quindi il controllo sulla loro posizione che questo sistema ci deve garantire.

Per tali motivi sono stati imposti come limite progettuale i seguenti valori:

- Velocità massima: $1{,}5\,\frac{m}{s}$
- Accelerazione massima: $15\,\frac{m}{s^2}$
- Jerk massimo: $150\,\frac{m}{s^3}$

Sulla base di questi dati serve scegliere la legge di moto più adatta. In particolare, si consideri, come esempio, l'esecuzione di una alzata unitaria in un periodo T unitario per poter confrontare in maniera normalizzata l'andamento di diverse leggi di moto.

Il diagramma delle alzate ottenuto applicando le leggi di moto polinomiale del 3° e 5° ordine, cicloidale, trapezoidale modificata e a doppia "esse" è mostrato in Figura 80.

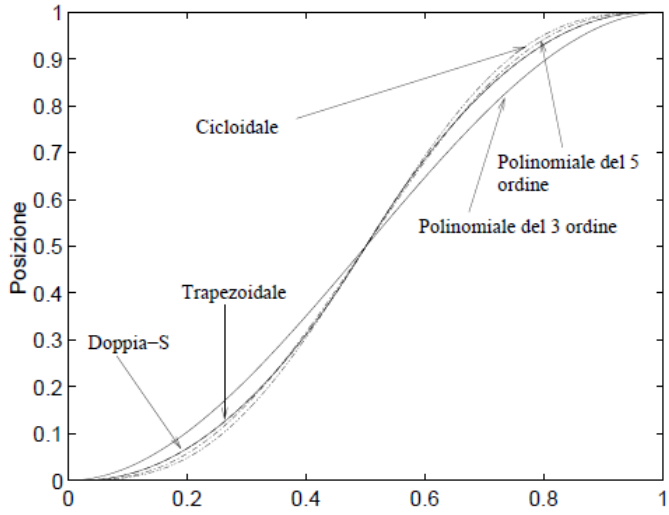


Figura 80 - Diagramma normalizzato di posizione per leggi di moto polinomiale del 3° e 5° ordine, cicloidale, trapezoidale modificata e trapezoidale a doppia "esse"

Dal diagramma delle alzate è importante osservare che tutte le leggi di moto $f(p)$ hanno un andamento abbastanza simile tra di loro e quindi questo diagramma non aiuta sulla scelta della particolare legge di moto.

I criteri su cui basarsi per scegliere una legge di moto sono i seguenti:

- Limitare la velocità massima.
- Limitare le accelerazioni massime e RMS. Infatti, siccome la coppia dipende in modo proporzionale dalla accelerazione, bassi valori di accelerazione di picco e RMS conducono a scegliere motori di taglia più piccola (meno costosi).
- Continuità fino all'accelerazione o fino al jerk. Le discontinuità nelle accelerazioni comportano discontinuità nelle forze e nelle coppie, che conducono a sollecitazioni ed urti nella macchina controllata. Infatti, discontinuità nelle accelerazioni producono movimenti irregolari nei meccanismi che, in presenza di giochi meccanici, comportano oscillazioni e collisioni delle parti a contatto negli organi di trasmissione del moto, con conseguenze usura e distruzione nel lungo periodo. Spesso il prodotto stesso non sopporta accelerazioni molto elevate. Ad esempio, un prodotto trasportato su di un nastro tende a scivolare se sollecitato da accelerazioni discontinue.
- Limitare la banda dello spettro del segnale di accelerazione. Questo si traduce in un limite di banda sulla corrente nel motore. Occorre infatti tenere presente

che il sistema azionamento elettrico-motore è un sistema con comportamento di tipo passa-basso, e quindi non può seguire fedelmente un segnale con banda spettrale troppo elevata. Quindi accelerazioni/correnti con banda troppo elevata producono maggiori errori di inseguimento durante i transitori ed eccitazione di frequenze di risonanza che possono innescare vibrazioni rumorose e dannose.

I diagrammi della velocità e accelerazione sono mostrati in Figura 81 e Figura 82.

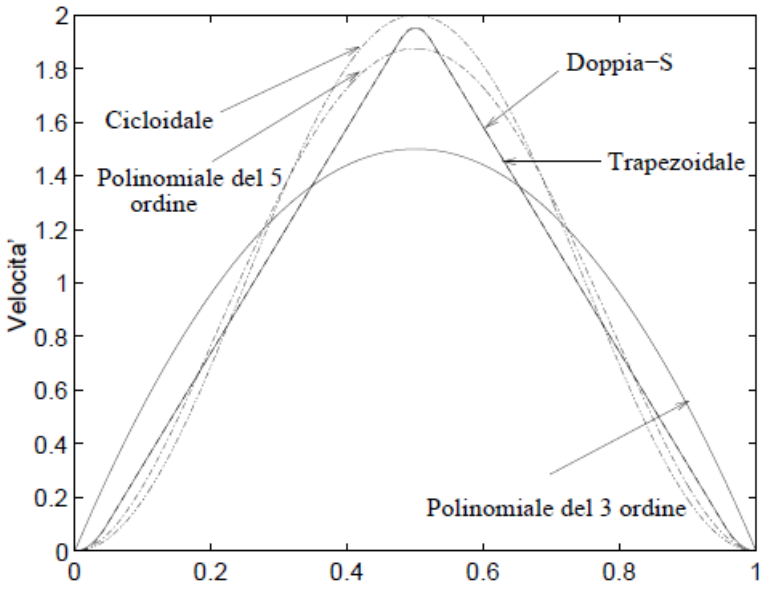


Figura 81 - Diagramma normalizzato di velocità per leggi di moto polinomiale del 3° e 5° ordine, cicloidale, trapezoidale modificata e trapezoidale a doppia "esse"

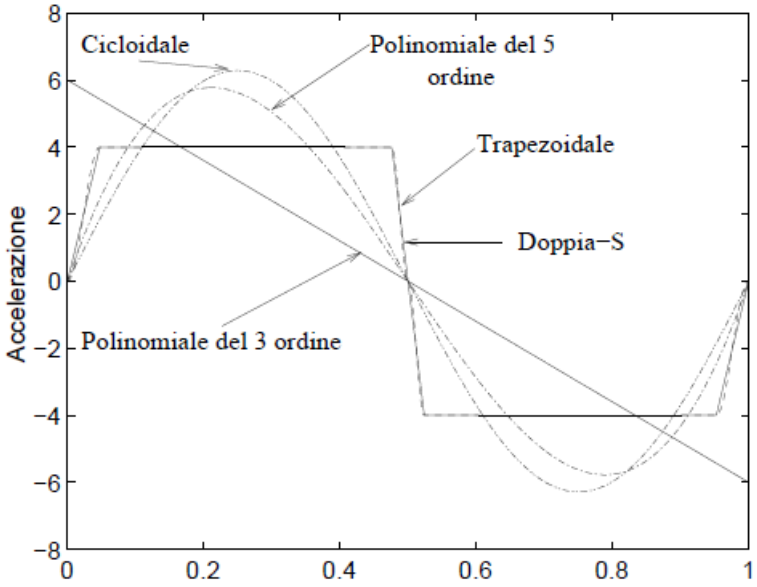


Figura 82 - Diagramma normalizzato di accelerazione per leggi di moto polinomiale del 3° e 5° ordine, cicloidale, trapezoidale modificata e trapezoidale a doppia "esse"

Tra i suddetti criteri, il sistema sincro-dinamico ha maggior necessità di mantenere una accelerazione massima limitata e di garantire un'ottima continuità dell'accelerazione.

Dalla Figura 82 risulta evidente che la legge di moto di tipo polinomiale del terzo ordine presenta discontinuità nell'accelerazione all'inizio ed alla fine della traiettoria e quindi è da escludere poiché potrebbe introdurre sollecitazioni di tipo dinamiche non desiderabili.

Le leggi cicloidale e polinomiale del quinto ordine sono molto simili e presentano una buona dolcezza del diagramma delle accelerazioni, mentre le leggi trapezoidale e a doppia-S sono simili tra loro e presentano un valore di accelerazione massima minima tra le leggi considerate. Inoltre, il polinomio di 5° grado, la legge cicloidale, trapezoidale modificata e a doppia "esse" sono continue nell'accelerazione e hanno jerk limitato [19].

Da queste considerazioni si deduce che la legge di moto trapezoidale è la più adatta per la generazione di traiettorie degli equipaggi e in particolare si sceglie di adottare la legge trapezoidale a doppia "esse".

4.3. Generazione della traiettoria

In Figura 83 è mostrato l'andamento tipico di una legge di moto trapezoidale a doppia "esse". La forma del profilo di velocità di questa traiettoria è la ragione da cui prende il nome questa legge di moto, conosciuta anche come traiettoria a campana o traiettoria a sette segmenti poiché è composta da sette diversi tratti a jerk costante.

La legge di moto a doppia "esse" è analoga alla legge di moto trapezoidale modificata, con la sola differenza che i tratti ad accelerazione costante T_a, T_v e T_d sono raccordati con tratti polinomiali del terzo ordine, che derivati due volte, forniscono tratti ad accelerazione lineare.

Negli intervalli T_{j1} e T_{j2} è utilizzata una legge di moto di tipo polinomiale cubica, mentre nei rimanenti tratti viene utilizzata una legge di moto ad accelerazione costante [20].

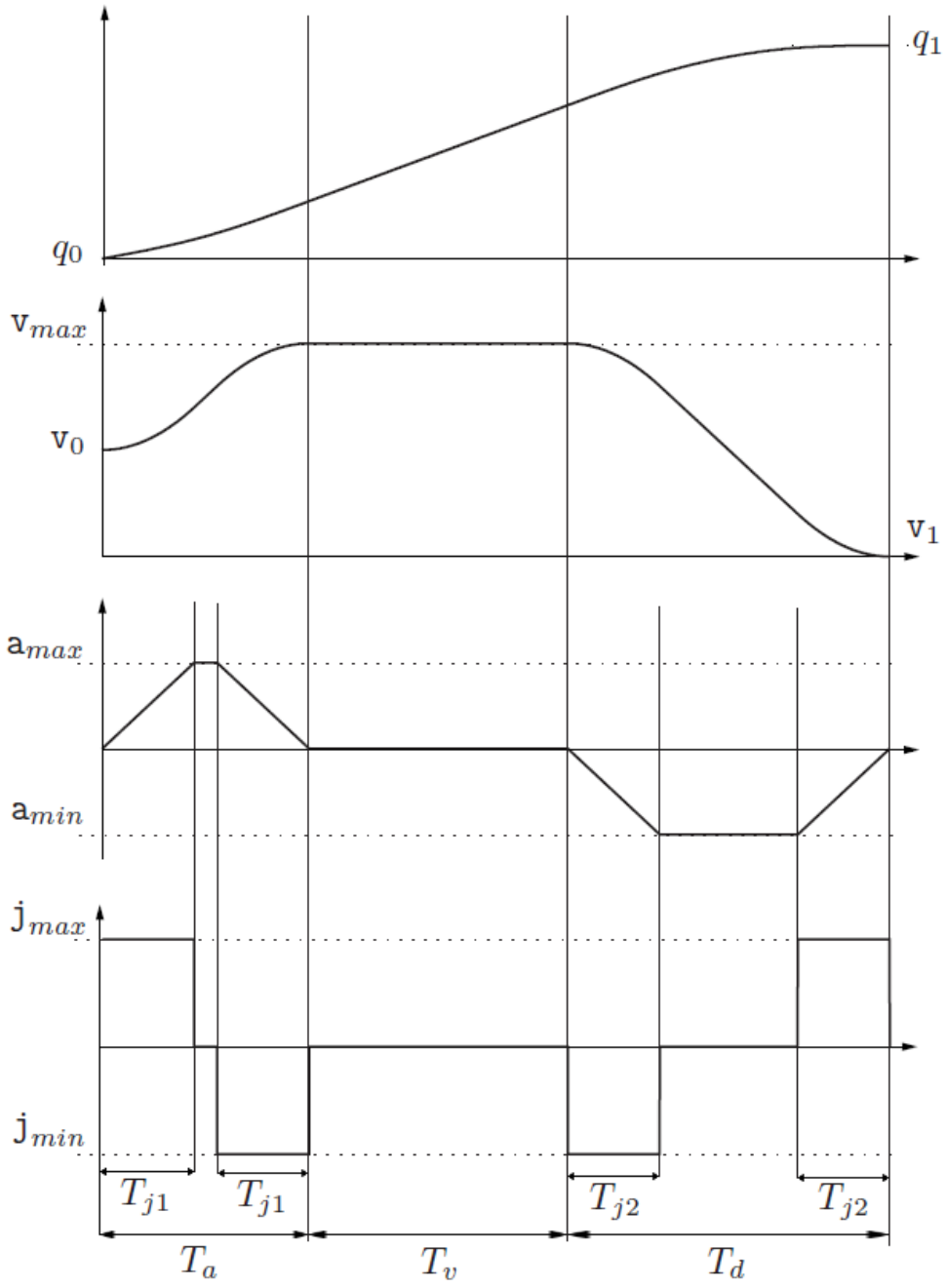


Figura 83 - Andamento tipico del profilo di posizione, velocità, accelerazione e jerk di una legge di moto trapezoidale a doppia "esse"

Definiamo i seguenti parametri della traiettoria:

T_{j1} : intervallo di tempo in cui il jerk è costante (j_{max} o j_{min}) durante la fase di accelerazione;

T_{j2} : intervallo di tempo in cui il jerk è costante (j_{max} o j_{min}) durante la fase di decelerazione;

T_a : periodo di accelerazione;

T_v : periodo di velocità costante;

T_d : periodo di decelerazione;

T : durata totale della traiettoria ($T = T_a + T_v + T_d$).

Come ipotesi semplificative assumiamo che:

$$a_{min} = -a_{max}\,, \qquad j_{min} = -j_{max}$$

$$T_{j1} = T_{j2}\,, T_a = T_d$$

In questo modo otteniamo una legge di moto simmetrica.

La strategia che verrà assunta per generare le traiettorie sarà quella di minimizzare la durata T della corsa (minimum time trajectory). Il fine ultimo è quello di redigere una carta delle operazioni determinando i tempi di spostamento degli equipaggi; se si completa la carta operazioni con i tempi minimi di spostamento si otterrà una carta con il massimo tempo di ozio possibile degli equipaggi. Nella carta delle operazioni, se accanto ad alcuni spostamenti è presente del tempo di ozio, vuol dire che si può eseguire quello spostamento con una legge di moto meno spinta dal punto di vista cinematico e questo parametro potrà essere utilizzato come determinante per la valutazione delle diverse configurazioni.

Le condizioni al contorno utilizzate sono:

- Posizione iniziale e finale generica: q_0, q_1
- Valore iniziale del tempo generico: $t_0 \neq 0$
- Valore inziale e finale della velocità generico: v_0, v_1
- Accelerazione iniziale e finale pari a zero: $a_0 = 0, a_1 = 0$

Nella traiettoria è possibile distinguere tre fasi distinte:

- Fase di accelerazione $t \in [t_0, t_0 + T_a]$: in questa fase l'accelerazione ha un andamento lineare dal suo valore iniziale al suo massimo e di nuovo al suo valore iniziale.
- Fase a velocità costante $t \in [t_0 + T_a, t_0 + T_a + T_v]$: questa fase ha un profilo di velocità costante.

- Fase di decelerazione $t \in [t_0 + T_a + T_v, t_0 + T]$: stesso profilo di accelerazione della prima fase, ma con verso opposto.

Dati i vincoli sui valori massimi di jerk, accelerazione e velocità, e dato lo spostamento desiderato $h = q_1 - q_0$, si calcola la traiettoria a durata minima. Tuttavia, è prima necessario verificare se una traiettoria può essere effettivamente eseguita o meno, poiché possono esserci diversi casi in cui una traiettoria non può essere calcolata con i vincoli dati.

Il caso limite è rappresentato da una singola fase di accelerazione (se $v_0 < v_1$) o fase di decelerazione (se $v_0 > v_1$). Pertanto, è necessario prima verificare se è possibile eseguire la traiettoria con un doppio gradino di jerk (uno positivo e uno negativo).

A questo scopo, definiamo:

$$T_j^* = \min\left\{\sqrt{\frac{|v_1 - v_o|}{j_{max}}}, \frac{a_{max}}{j_{max}}\right\}$$

Se $T_j^* = \frac{a_{max}}{j_{max}}$, l'accelerazione raggiunge il suo valore massimo e può esistere un segmento con zero jerk.

Allora, la traiettoria è fattibile se:

$$q_1 - q_0 > \begin{cases} T_j^*(v_0 + v_!), & Se \quad T_j^* < \dfrac{a_{max}}{j_{max}} \\ \dfrac{1}{2}(v_0 + v_1)\left[T_j^* + \dfrac{|v_1 - v_0|}{a_{max}}\right], & Se \quad T_j^* = \dfrac{a_{max}}{j_{max}} \end{cases}$$

Se una di queste due diseguaglianze è vera, allora è possibile calcolare i parametri della traiettoria, ovvero i tempi T_j, T_a, T_v, ma in base alla velocità realizzata durante il movimento ci possono essere due casi. Definendo il valore massimo della velocità durante il movimento come $v_{lim} = \max(\dot{q}(t))$, ci possono essere due possibilità:

$$Caso\ 1: \quad v_{lim} = v_{max}$$

$$Caso\ 2: \quad v_{lim} < v_{max}$$

Nel caso 2 la traiettoria non raggiunge la velocità massima delle specifiche di progetto e pertanto il profilo presenterà soltanto una fase di accelerazione e una di decelerazione con assenza di segmento a velocità costante ($T_v = 0$).

Sia nel caso 1 che nel caso 2 è possibile che non si raggiunga l'accelerazione massima delle specifiche di progetto e questo potrebbe accadere se la massima accelerazione possibile è molto alta ("alta dinamica" del sistema), se lo spostamento è molto piccolo o se la velocità finale o iniziale è molto vicina alla velocità massima possibile. Se si è in questi casi allora non sarà presente il segmento ad accelerazione costante.

La possibilità di raggiungere o meno l'accelerazione massima possibile determina per ciascuno dei due casi altri due sottocasi.

- $Caso\ 1$: $v_{lim} = v_{max}$.
  In questo caso è possibile verificare se si raggiunge la massima accelerazione possibile attraverso le seguenti condizioni, andando a determinare i due sottocasi.

  - $1.a : a_{lim} = a_{max}$;
    a_{max} è raggiunta se: $(v_{max} - v_0)\, j_{max} \geq a_{max}^2$;
    In questo caso i parametri della traiettoria si calcolano in questo modo:

    $$T_j = \frac{a_{max}}{j_{max}}, \qquad T_a = T_j + \frac{v_{max} - v_0}{a_{max}}$$

  - $1.b : a_{lim} < a_{max}$;
    a_{max} non è raggiunta se: $(v_{max} - v_0)\, j_{max} < a_{max}^2$;
    In questo caso i parametri della traiettoria si calcolano in questo modo:

    $$T_j = \sqrt{\frac{v_{max} - v_0}{j_{max}}}, \qquad T_a = 2T_j$$

  Conoscendo T_j e T_a si può calcolare anche il parametro T_v:

  $$T_v = \frac{q_1 - q_0}{v_{max}} - \frac{T_a}{2}\left(1 + \frac{v_0}{v_{max}}\right) - \frac{T_d}{2}\left(1 + \frac{v_1}{v_{max}}\right)$$

  Se $T_v > 0$ allora la massima velocità possibile è raggiunta e pertanto il caso 1 è corretto, si utilizzerà il valore $v_{lim} = v_{max}$ nelle equazioni del moto e il valore $a_{lim} = a_{max}$ se si è nel caso $1.a$ o il valore a_{lim} se si è nel caso $1.b$.
  Se $T_v \leq 0$ allora vuol dire che il valore v_{lim} è più piccolo della specifica v_{max} e pertanto, imponendo $T_v = 0$, bisogna passare al caso 2.

- $Caso\ 2: v_{lim} < v_{max}$.
  In questo caso si ha $T_v = 0$. Analogamente si distinguono i due sottocasi in base all'accelerazione ottenibile nella traiettoria.

  - $2.a: a_{lim} = a_{max}$;
    Per determinare se l'accelerazione massima possibile è raggiunta si calcolano i parametri nel seguente modo:

    $$T_j = \frac{a_{max}}{j_{max}}, \qquad T_a = \frac{a_{max}^2 - 2v_0 + \sqrt{\Delta}}{a_{max}}$$

    dove:

    $$\Delta = \frac{a_{max}^4}{j_{max}^2} + 2(v_0^2 + v_1^2) + a_{max}\left[4(q_1 - q_0) - 2\frac{a_{max}}{j_{max}}(v_0 + v_1)\right]$$

    Se $T_a \geq 2T_j$ allora l'accelerazione massima è raggiunta e i parametri si calcolano come mostrato nel caso $2.a$. Se invece $T_a < 2T_j$ allora l'accelerazione massima non è raggiunta e si passa al caso $2.b$.

  - $2.b: a_{lim} < a_{max}$;
    In questo caso (piuttosto raro) i parametri della traiettoria sono difficili da calcolare e sarebbe più conveniente cercare una soluzione approssimata, sebbene non ottimale, che sia accettabile dal punto di vista computazionale. Una via possibile è quella di diminuire l'accelerazione massima assumendo $a_{max} = \gamma a_{max}$ $(0 < \gamma < 1)$ e calcolare la durata dei segmenti T_a, T_j con le stesse formule del caso $2.a$ fino a quando non si soddisfa la condizione $T_a > 2T_j$.

Una volta calcolati i parametri della traiettoria T_j, T_a e T_v e determinato se $a_{lim} = a_{max}$ (o $a_{lim} < a_{max}$), $v_{lim} = v_{max}$ (o $v_{lim} < v_{max}$), si può proseguire con il calcolo delle equazioni del moto.

Queste equazioni saranno definite in base alla fase del moto, ovvero ai segmenti temporali che suddividono la traiettoria.

- **Fase di accelerazione**

  a) $t \in [t_0, t_0 + T_j]$

$$\begin{cases} q(t) = q_0 + v_0(t - t_0) + j_{max}\dfrac{(t - t_0)^3}{6} \\ \dot{q}(t) = v_0 + j_{max}\dfrac{(t - t_0)^2}{2} \\ \ddot{q}(t) = j_{max}(t - t_0) \\ \dddot{q}(t) = j_{max} \end{cases}$$

  b) $t \in [t_0 + T_j, t_0 + T_a - T_j]$

$$\begin{cases} q(t) = q_0 + v_0(t - t_0) + \dfrac{a_{lim}}{6}\left(3(t - t_0)^2 - 3T_j(t - t_0) + T_j^2\right) \\ \dot{q}(t) = v_0 + a_{lim}\left(t - \dfrac{T_j}{2} - t_0\right) \\ \ddot{q}(t) = j_{max}T_j \\ \dddot{q}(t) = 0 \end{cases}$$

  c) $t \in [t_0 + T_a - T_j, t_0 + T_a]$

$$\begin{cases} q(t) = q_0 + (v_{lim} + v_0)\dfrac{T_a}{2} - v_{lim}(T_a - t + t_0) + j_{max}\dfrac{(T_a - t - t_0)^3}{6} \\ \dot{q}(t) = v_{lim} - j_{max}\dfrac{(T_a - t - t_0)^2}{2} \\ \ddot{q}(t) = j_{max}(T_a - t - t_0) \\ \dddot{q}(t) = j_{max} \end{cases}$$

- **Fase a velocità costante**

  a) $t \in [t_0 + T_a, t_0 + T_a + T_v]$

$$\begin{cases} q(t) = q_0 + (v_{lim} + v_0)\dfrac{T_a}{2} + v_{lim}(t - T_a - t_0) \\ \dot{q}(t) = v_{lim} \\ \ddot{q}(t) = 0 \\ \dddot{q}(t) = 0 \end{cases}$$

- **Fase di decelerazione**

a) $t \in [t_1 - T_a, t_1 - T_a + T_j]$

$$\begin{cases} q(t) = q_1 - (v_{lim} + v_1)\dfrac{T_a}{2} + v_{lim}(t - t_1 + T_a) - j_{max}\dfrac{(t - t_1 + T_a)^3}{6} \\ \dot{q}(t) = v_{lim} - j_{max}\dfrac{(t - t_1 + T_a)^2}{2} \\ \ddot{q}(t) = -j_{max}T_j \\ \dddot{q}(t) = -j_{max} \end{cases}$$

b) $t \in [t_1 - T_a + T_j, t_1 - T_j]$

$$\begin{cases} q(t) = q_1 - (v_{lim} + v_1)\dfrac{T_a}{2} + v_{lim}(t - t_1 + T_a) + \\ \quad -\dfrac{a_{lim}}{6}\left(3(t - t_1 + T_a)^2 - 3T_j(t - t_1 + T_a) + T_j^2\right) \\ \dot{q}(t) = v_{lim} + a_{lim}\left(t - t_1 + T_a - \dfrac{T_j}{2}\right) \\ \ddot{q}(t) = -j_{max}T_j \\ \dddot{q}(t) = 0 \end{cases}$$

c) $t \in [t_1 - T_j, t_1]$

$$\begin{cases} q(t) = q_1 - v_1(t_1 - t) - j_{max}\dfrac{(t_1 - t)^3}{6} \\ \dot{q}(t) = v_1 + j_{max}\dfrac{(t_1 - t)^2}{2} \\ \ddot{q}(t) = -j_{max}(t_1 - t) \\ \dddot{q}(t) = j_{max} \end{cases}$$

Per il calcolo delle traiettorie degli equipaggi, queste equazioni vengono implementate su MATLAB in una funzione `fastest_alzata`, che prende in input i parametri $[v_{max},\ a_{max},\ j_{max},\ q_0,\ t_0, v_0,\ v_1,\ h,\ t_{step}]$ e restituisce in output la traiettoria e il periodo T (minimo possibile) $[\dot{q},\ \ddot{q},\ \dddot{q},\ T]$.

t_{step} è un parametro che determina in quanti intervalli viene suddiviso il periodo di calcolo: più è piccolo il suo valore, più sono numerosi gli intervalli e quindi maggiore è la precisione di calcolo. Viene impostato $t_{step} = 0{,}0001$.

Infine, su MATLAB si scrive un semplicissimo codice che imposta il valore delle specifiche massime su velocità, accelerazione, jerk, condizioni al contorno e alzata richiesta. Questi dati vengono forniti come input per la funzione `fastest_alzata`, e si plottano i profili di posizione, velocità, accelerazione e jerk per ogni alzata di ogni configurazione di macchina.

Per ogni configurazione ci sono diverse distanze da percorrere per spostare un equipaggio tra una stazione e l'altra e per completare i caricamenti e scaricamenti, ognuna di queste alzate verrà identificata da un pedice alfabetico.

- Si esegue il calcolo delle traiettorie per la configurazione 8x20 (Figura 84).

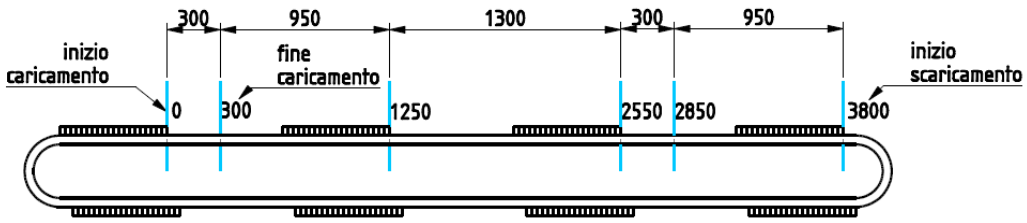


Figura 84 - Distanze da percorrere per una configurazione 8x20

```
%initial data
t_step=0.0001;
v0=0;            %mm/s
v1=0;            %mm/s
j_max=150000;    %mm/s^3
a_max=15000;      %mm/s^2
v_max=1500;      %mm/s
q0=0;           %mm
t0=0;           %s

%trattoA
hA=300;          %mm
[qA,qA_dot,qA_2dot,qA_3dot,TA,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hA,t_s
tep);

%trattoB
hB=950;          %mm
[qB,qB_dot,qB_2dot,qB_3dot,TB,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hB,t_s
tep);

%trattoC
hC=1300;          %mm
[qC,qC_dot,qC_2dot,qC_3dot,TC,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hC,t_s
tep);
```

Si ottengono le leggi di moto rappresentate in Figura 85, Figura 86 e Figura 87.

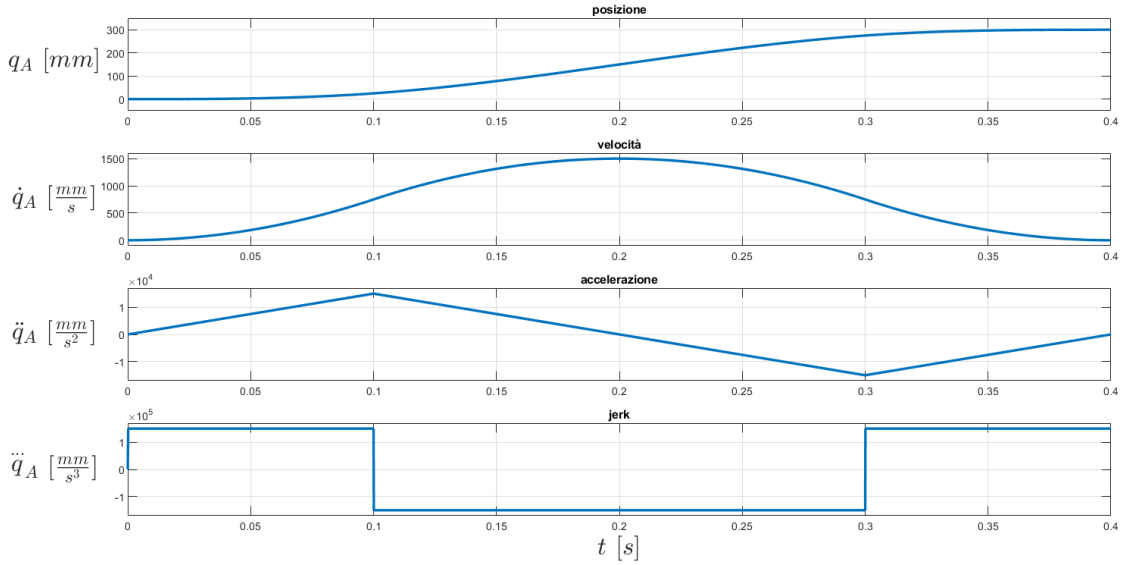


Figura 85 – Configurazione 8x20: alzata di 300 mm

$$T_A = 0{,}4\ s$$

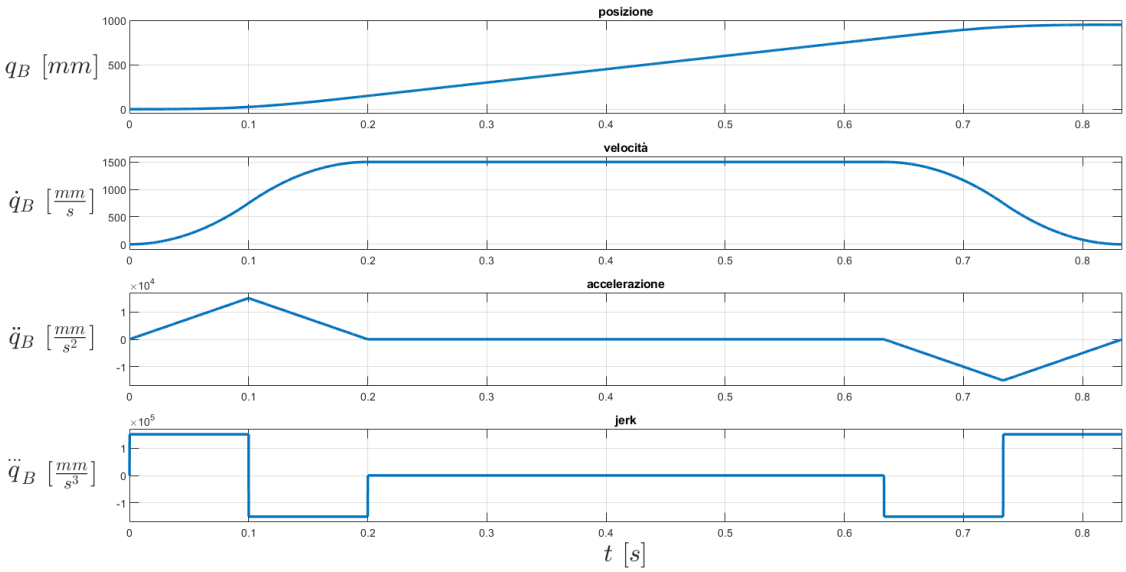


Figura 86 – Configurazione 8x20: alzata di 950 mm

$$T_B = 0{,}8333\ s$$

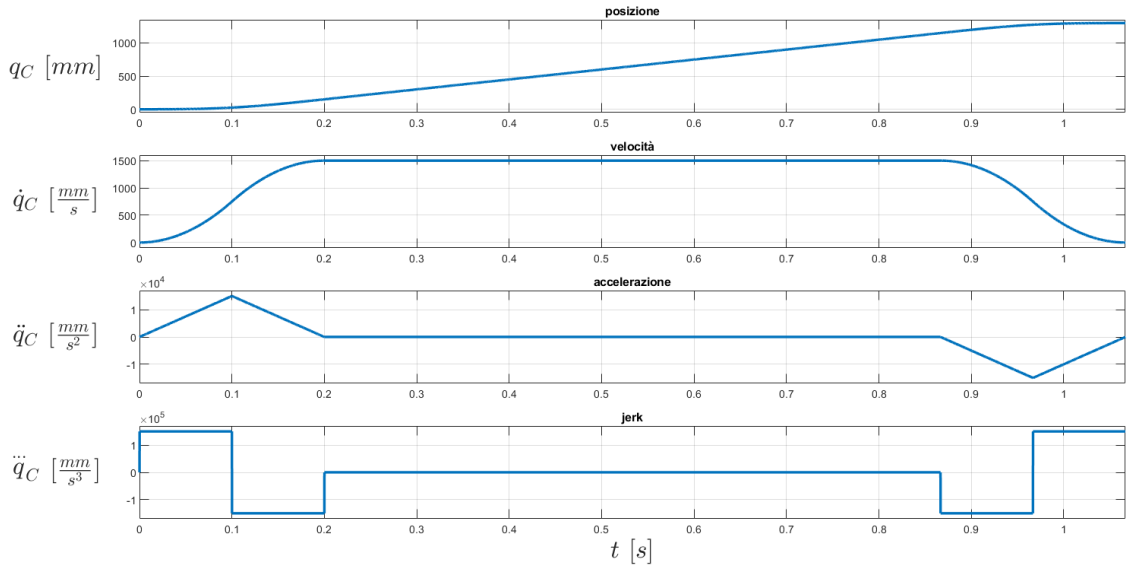


Figura 87 – Configurazione 8x20: alzata di 1300 mm

$$T_C = 1{,}0667\ s$$

- Si esegue il calcolo delle traiettorie per la configurazione 8x20 con il robot di caricamento che preleva a passo di 60 mm (Figura 88).

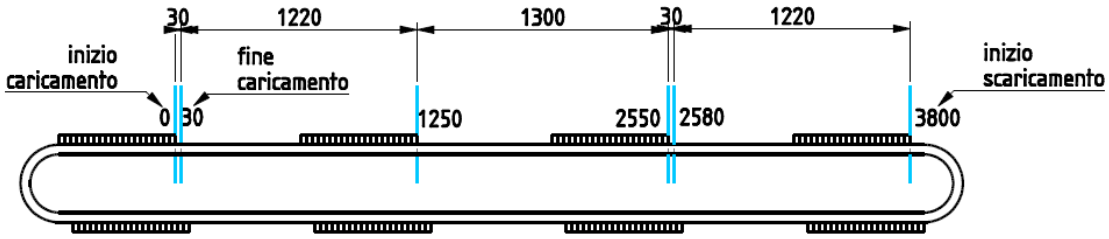


Figura 88 - Distanze da percorrere nella configurazione 8x20 con robot di caricamento a passo 60 mm

```
%initial data
t_step=0.0001;
v0=0;            %mm/s
v1=0;            %mm/s
j_max=150000;    %mm/s^3
a_max=15000;      %mm/s^2
v_max=1500;      %mm/s
q0=0;           %mm
t0=0;           %s

%trattoA
hA=30;          %mm
[qA,qA_dot,qA_2dot,qA_3dot,TA,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hA,t_s
    tep);

%trattoB
hB=1220;          %mm
[qB,qB_dot,qB_2dot,qB_3dot,TB,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hB,t_s
    tep);

%trattoC
hC=1300;          %mm
[qC,qC_dot,qC_2dot,qC_3dot,TC,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hC,t_s
    tep);
```

Si ottengono le leggi di moto rappresentate in Figura 89, Figura 90 e Figura 91.

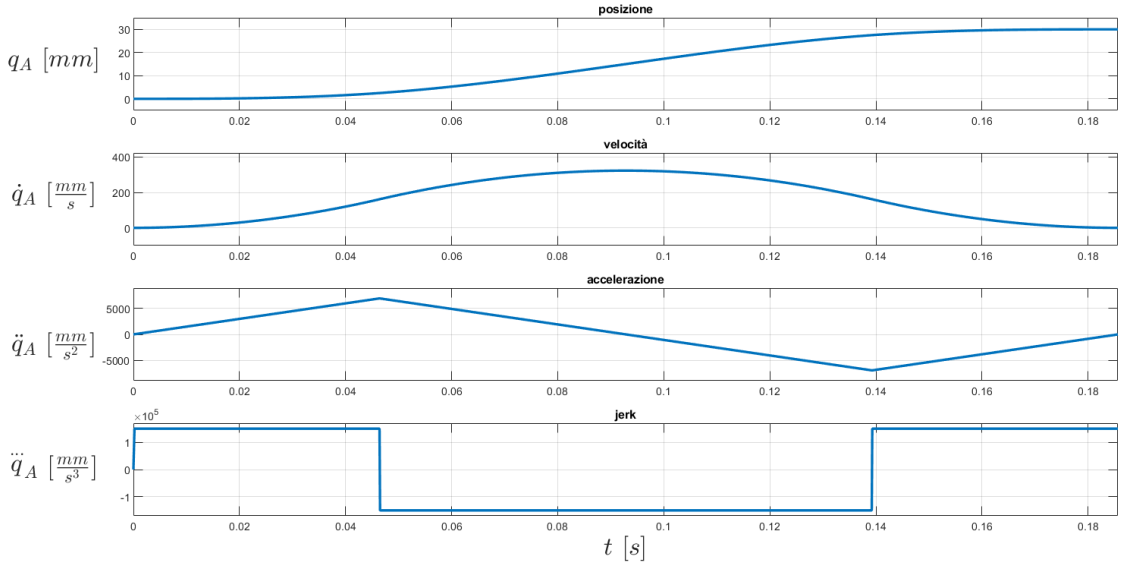


Figura 89 - Configurazione 8x20 con caricamento a passo 60 mm: alzata di 30 mm

$$T_A = 0{,}186\ s$$

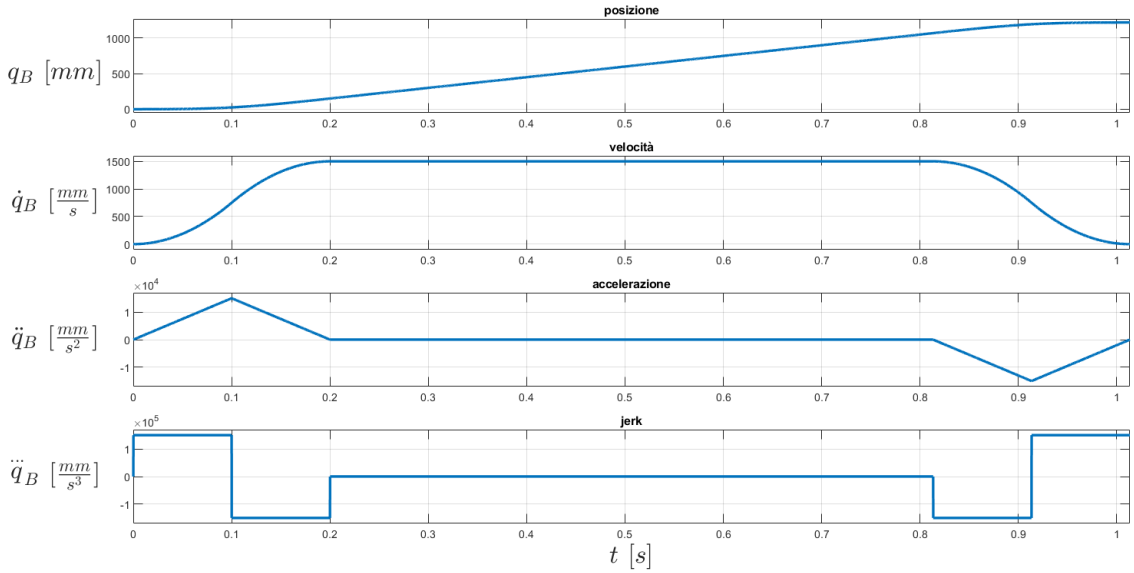


Figura 90 - Configurazione 8x20 con caricamento a passo 60 mm: alzata di 1220 mm

$$T_B = 1{,}013\ s$$

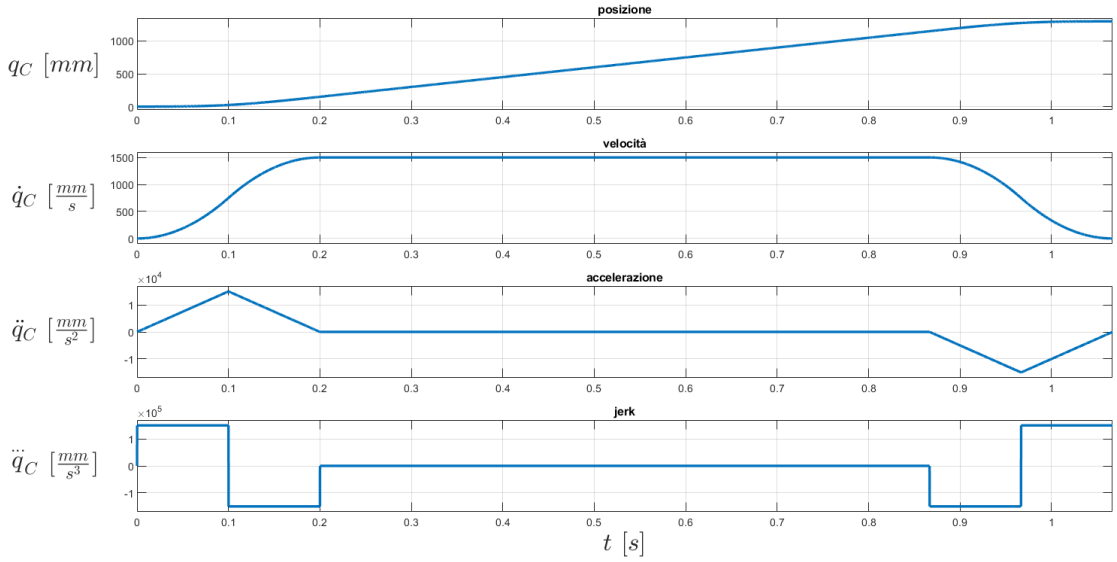


Figura 91 - Configurazione 8x20 con caricamento a passo 60 mm: alzata di 1300 mm

$$T_C = 1{,}067\ s$$

- Si esegue il calcolo delle traiettorie per la configurazione 4x20 (Figura 92).

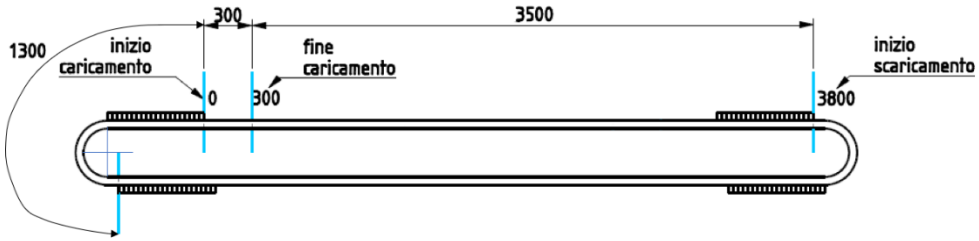


Figura 92 - Distanze da percorrere per una configurazione 4x20

```
%initial data
t_step=0.0001;
v0=0;            %mm/s
v1=0;            %mm/s
j_max=150000;    %mm/s^3
a_max=15000;      %mm/s^2
v_max=1500;      %mm/s
q0=0;           %mm
t0=0;           %s

%trattoA
hA=300;          %mm
[qA,qA_dot,qA_2dot,qA_3dot,TA,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hA,t_s
tep);

%trattoB
hB=3500;          %mm
[qB,qB_dot,qB_2dot,qB_3dot,TB,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hB,t_s
tep);

%trattoC
hC=1300;          %mm
[qC,qC_dot,qC_2dot,qC_3dot,TC,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hC,t_s
tep);
```

Si ottengono le leggi di moto rappresentate in Figura 93, Figura 94 e Figura 95.

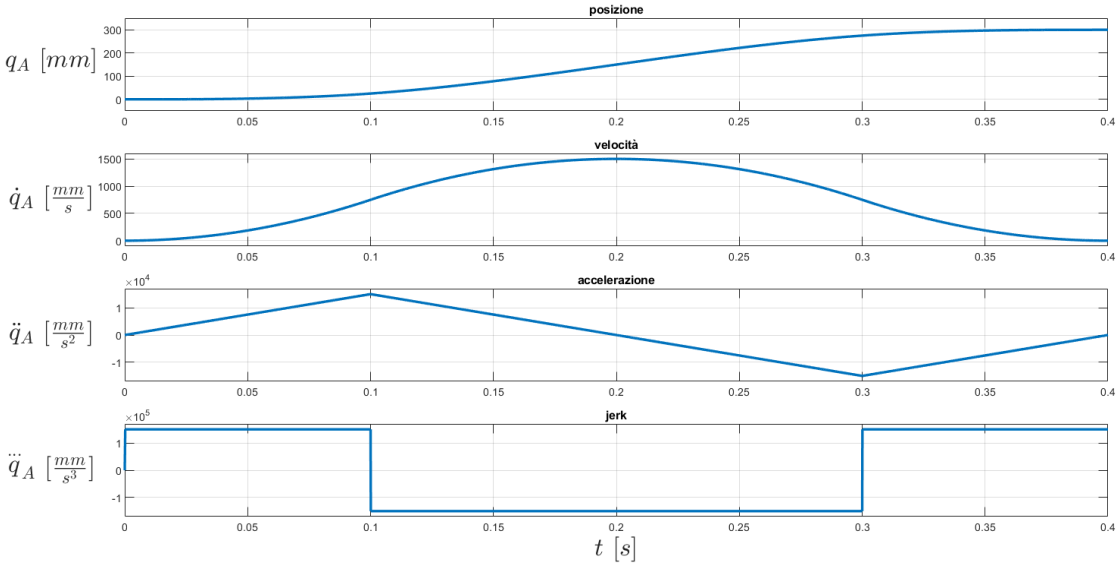


Figura 93 – Configurazione 4x20: alzata di 300 mm

$$T_A = 0{,}4\ s$$

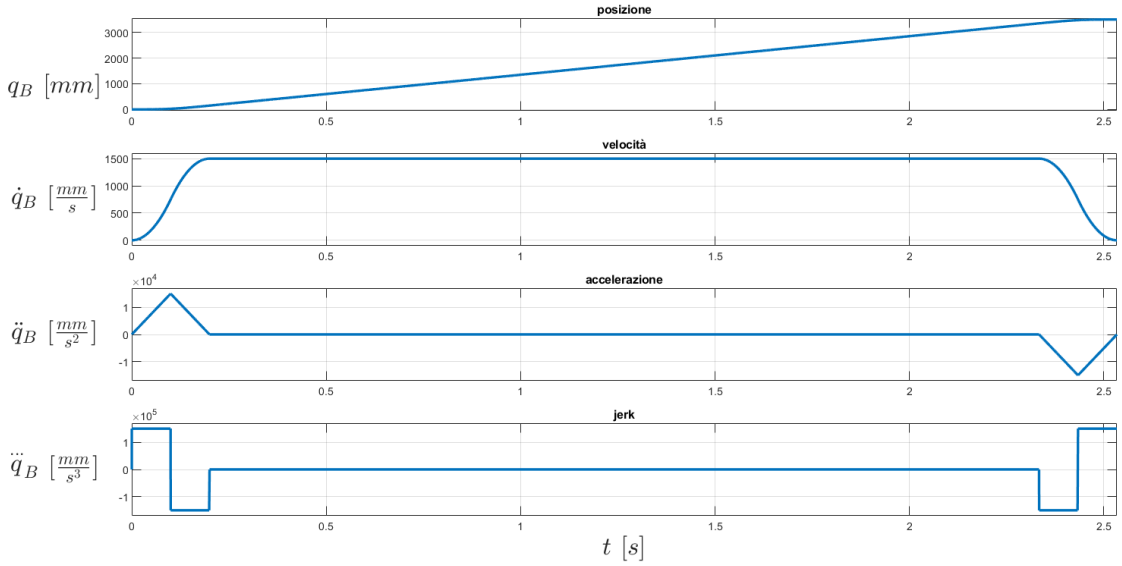


Figura 94 – Configurazione 4x20: alzata di 3500 mm

$$T_B = 2{,}5333\ s$$

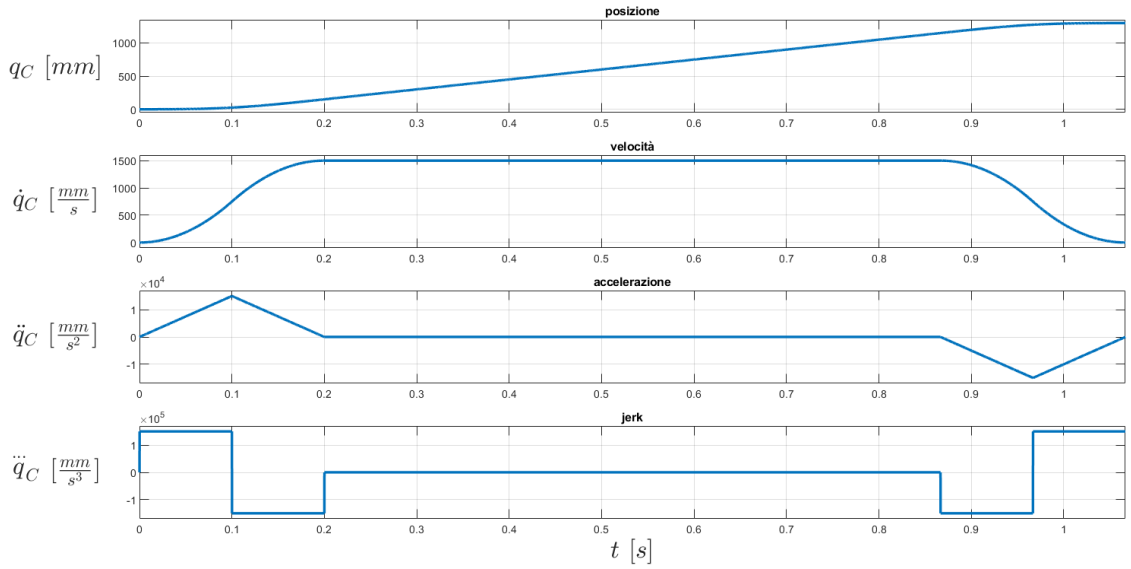


Figura 95 – Configurazione 4x20: alzata di 1300 mm

$$T_C = 1{,}0667\ s$$

- Si esegue il calcolo delle traiettorie per la configurazione 4x20 con il robot di caricamento che preleva a passo di 60 mm (Figura 96).

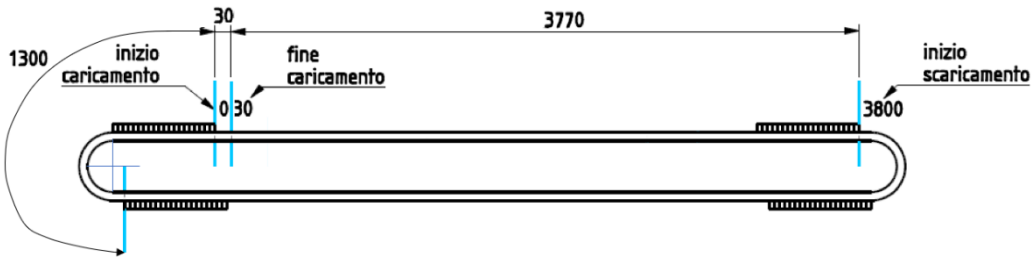


Figura 96 - Distanze da percorrere nella configurazione 4x20 con robot di caricamento a passo 60 mm

```
%initial data
t_step=0.0001;
v0=0;            %mm/s
v1=0;            %mm/s
j_max=150000;    %mm/s^3
a_max=15000;      %mm/s^2
v_max=1500;      %mm/s
q0=0;           %mm
t0=0;           %s

%trattoA
hA=30;          %mm
[qA,qA_dot,qA_2dot,qA_3dot,TA,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hA,t_s
    tep);

%trattoB
hB=3770;          %mm
[qB,qB_dot,qB_2dot,qB_3dot,TB,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hB,t_s
    tep);

%trattoC
hC=1300;          %mm
[qC,qC_dot,qC_2dot,qC_3dot,TC,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hC,t_s
    tep);
```

Si ottengono le leggi di moto rappresentate in Figura 97, Figura 98 e Figura 99.

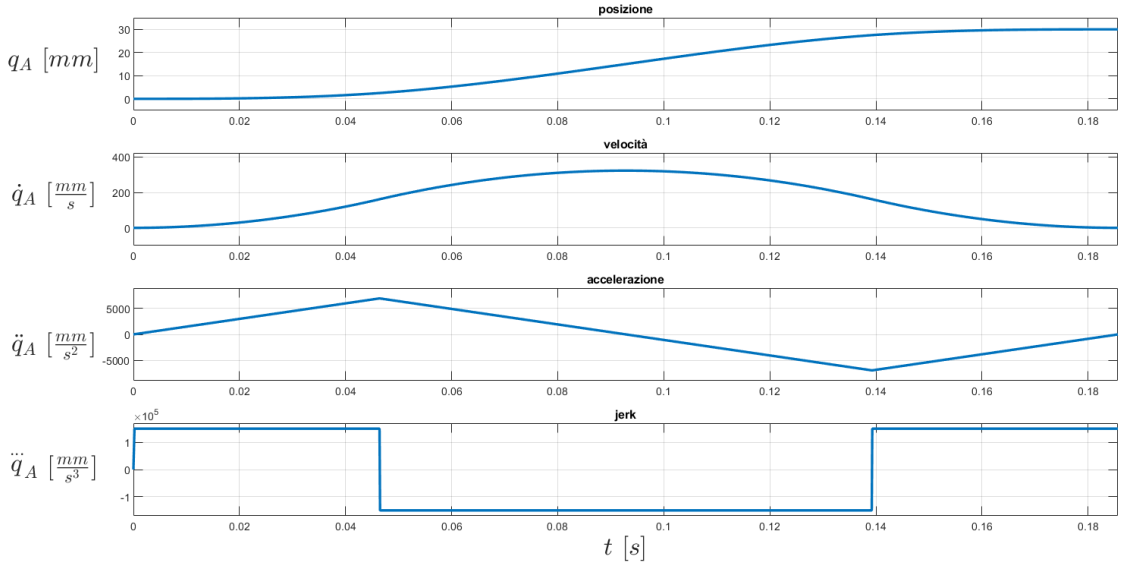


Figura 97 - Configurazione 4x20 con caricamento a passo 60 mm: alzata di 30 mm

$$T_A = 0{,}186\ s$$

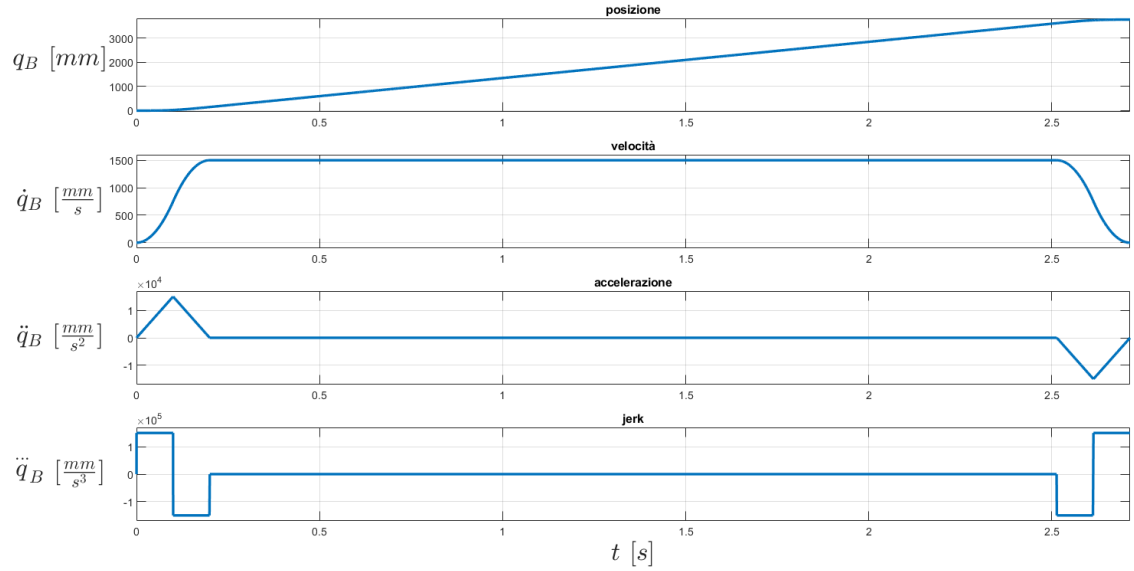


Figura 98 - Configurazione 4x20 con caricamento a passo 60 mm: alzata di 3770 mm

$$T_B = 2{,}713\ s$$

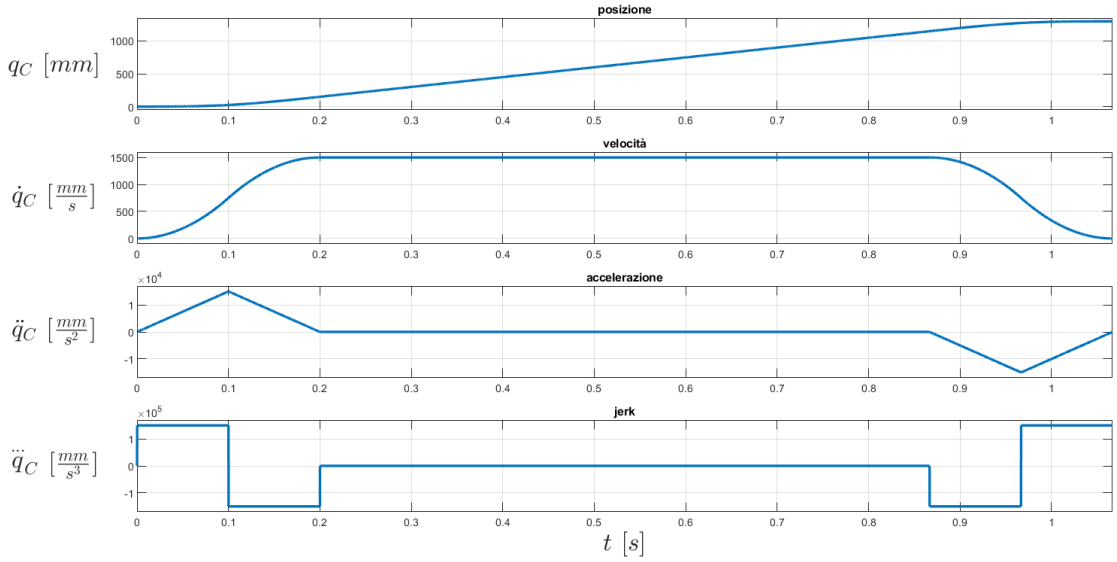


Figura 99 - Configurazione 4x20 con caricamento a passo 60 mm: alzata di 1300 mm

$$T_C = 1{,}067\ s$$

- Si esegue il calcolo delle traiettorie per la configurazione 4x40 (Figura 100).

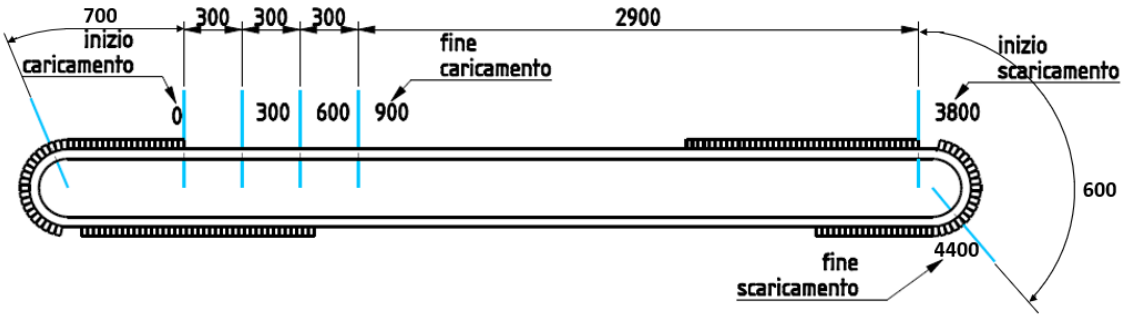


Figura 100 - Distanze da percorrere nella configurazione 4x40

```
85. %initial data
86. t_step=0.0001;
87. v0=0;            %mm/s
88. v1=0;            %mm/s
89. j_max=150000;    %mm/s^3
90. a_max=15000;      %mm/s^2
91. v_max=1500;      %mm/s
92. q0=0;           %mm
93. t0=0;           %s
94.
95. %trattoA
96. hA=300;          %mm
97. [qA,qA_dot,qA_2dot,qA_3dot,TA,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hA,t_s
    tep);
98.
99. %trattoB
100.  hB=600;          %mm
101.  [qB,qB_dot,qB_2dot,qB_3dot,TB,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hB,t
    _step);
102.
103.  %trattoC
104.  hC=700;          %mm
105.  [qC,qC_dot,qC_2dot,qC_3dot,TC,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hC,t
    _step);
106.
107.  %trattoD
108.  hD=2900;          %mm
109.  [qD,qD_dot,qD_2dot,qD_3dot,TD,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hD,t
    _step);
```

Si ottengono le leggi di moto rappresentate in Figura 101, Figura 102, Figura 103 e Figura 104.

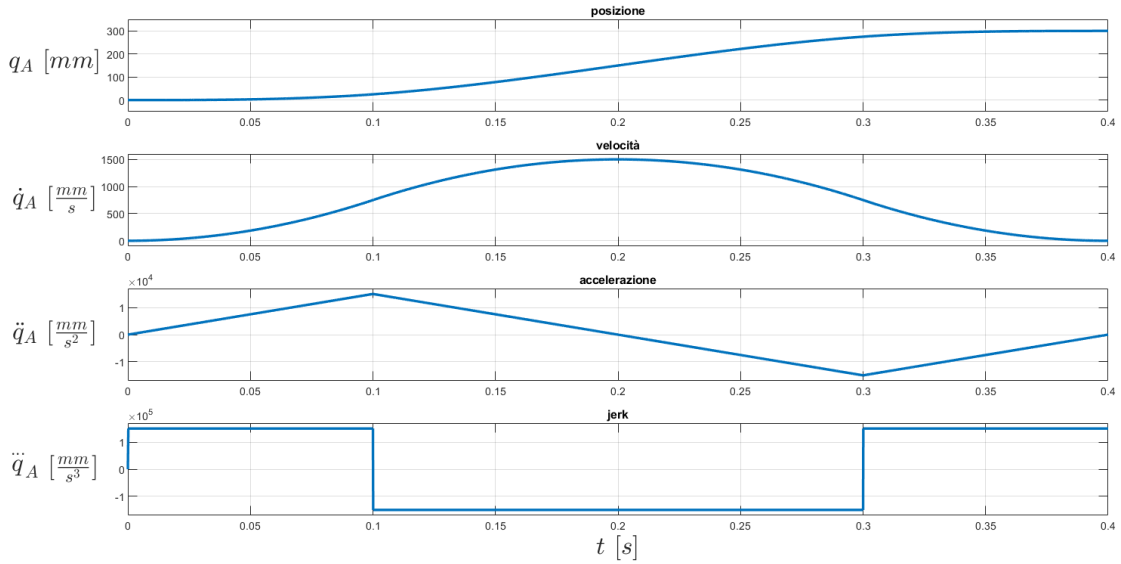


Figura 101 – Configurazione 4x40: alzata di 300 mm

$$T_A = 0{,}4\ s$$

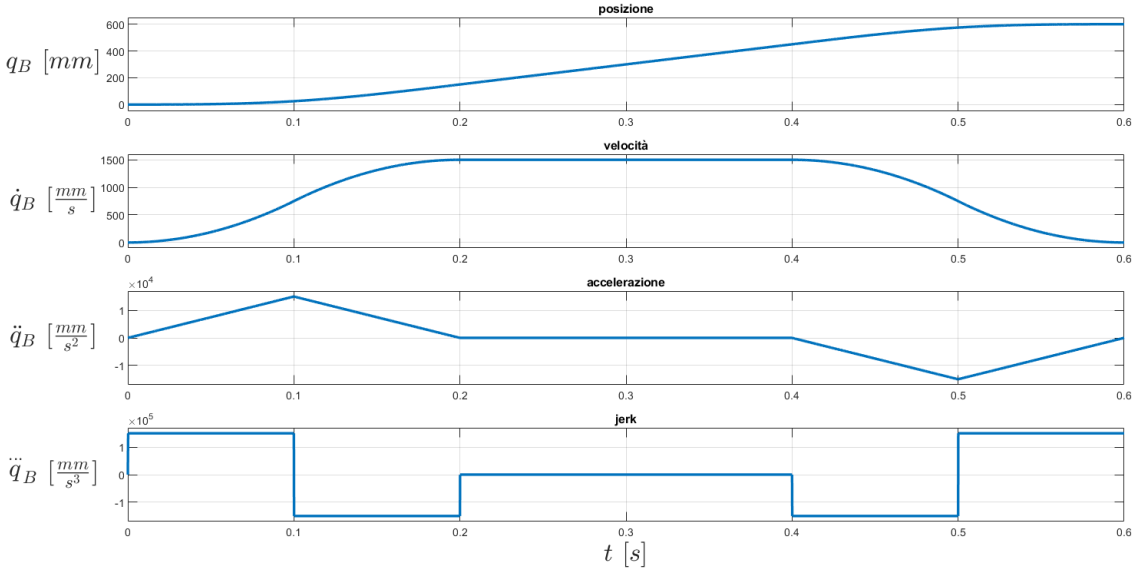


Figura 102 – Configurazione 4x40: alzata di 600 mm

$$T_B = 0{,}6\ s$$

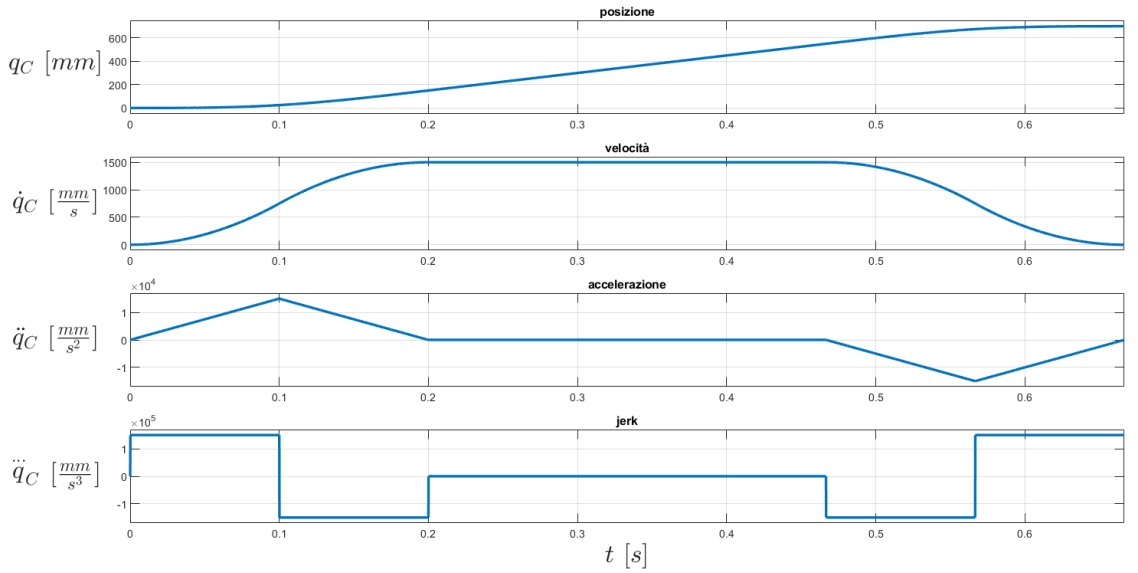


Figura 103 – Configurazione 4x40: alzata di 700 mm

$$T_C = 0{,}6666\ s$$

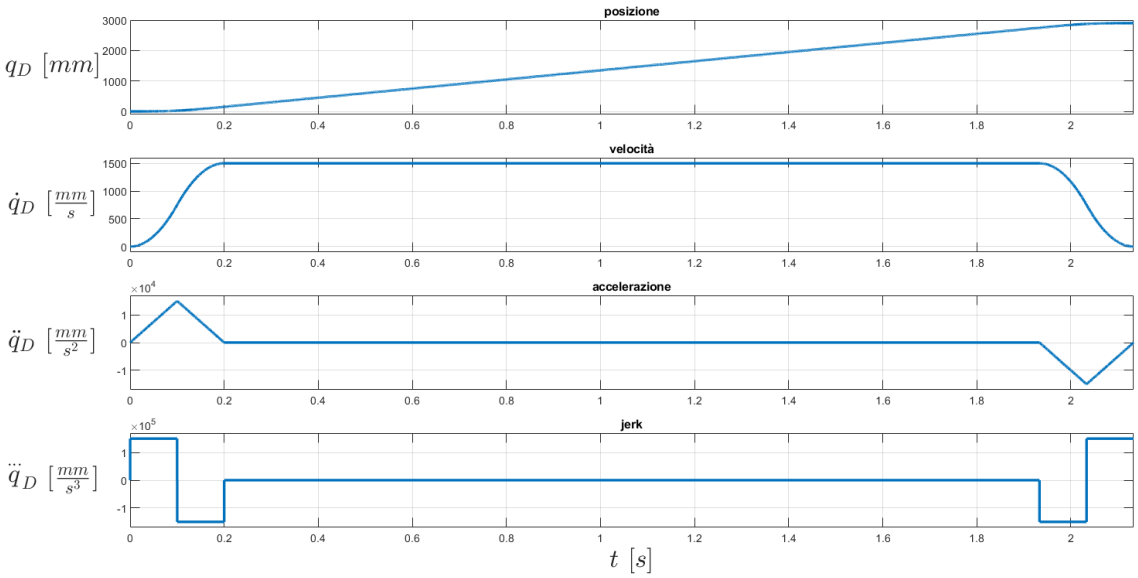


Figura 104 – Configurazione 4x40: alzata di 2900 mm

$$T_D = 2{,}1334\ s$$

- Si esgue il calcolo delle traiettorie per la configurazione 4x40 con il robot di caricamento che preleva a passo di 60 mm (Figura 105).

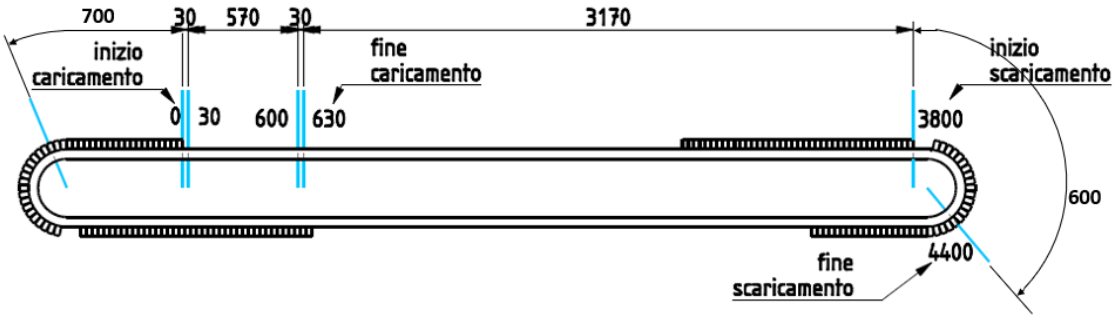


Figura 105 - Distanze da percorrere nella configurazione 4x40 con robot di caricamento con passo 60 mm

```
%initial data
t_step=0.0001;
v0=0;            %mm/s
v1=0;            %mm/s
j_max=150000;    %mm/s^3
a_max=15000;      %mm/s^2
v_max=1500;      %mm/s
q0=0;           %mm
t0=0;           %s

%trattoA
hA=30;          %mm
[qA,qA_dot,qA_2dot,qA_3dot,TA,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hA,t_step);

%trattoB
hB=570;          %mm
[qB,qB_dot,qB_2dot,qB_3dot,TB,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hB,t_step);

%trattoC
hC=600;          %mm
[qC,qC_dot,qC_2dot,qC_3dot,TC,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hC,t_step);

%trattoD
hD=700;          %mm
[qD,qD_dot,qD_2dot,qD_3dot,TD,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hD,t_step);

%trattoE
hD=3170;          %mm
[qE,qE_dot,qE_2dot,qE_3dot,TE,]=fastest_alzata(j_max,a_max,v_max,q0,t0,v0,v1,hE,t_step);
```

Si ottengono le leggi di moto rappresentate in Figura 106, Figura 107, Figura 108, Figura 109 e Figura 110.

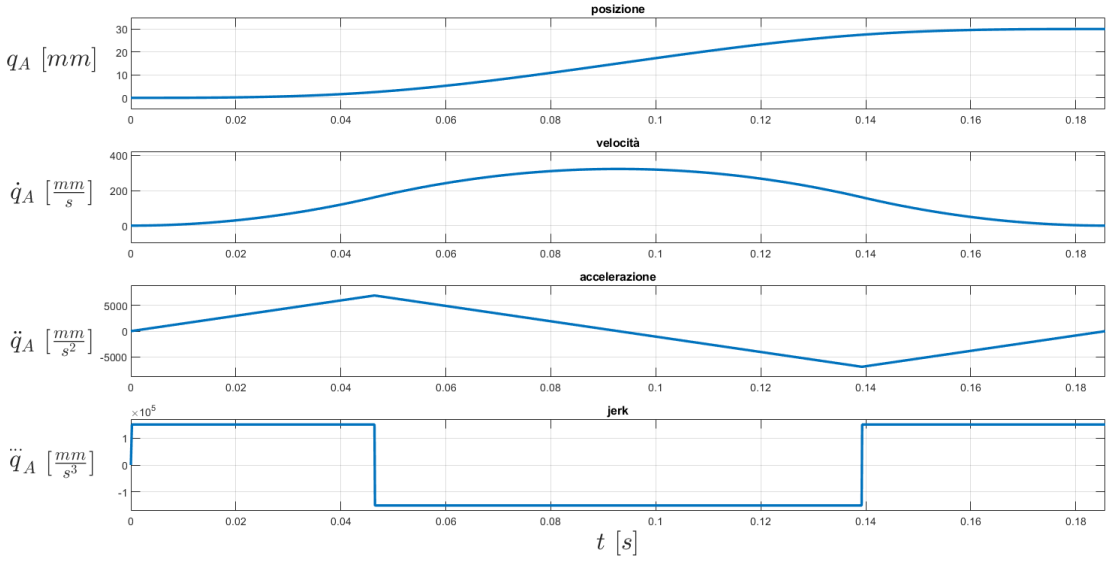


Figura 106 - Configurazione 4x40 con caricamento a passo 60 mm: alzata di 30 mm

$$T_A = 0{,}185\ s$$

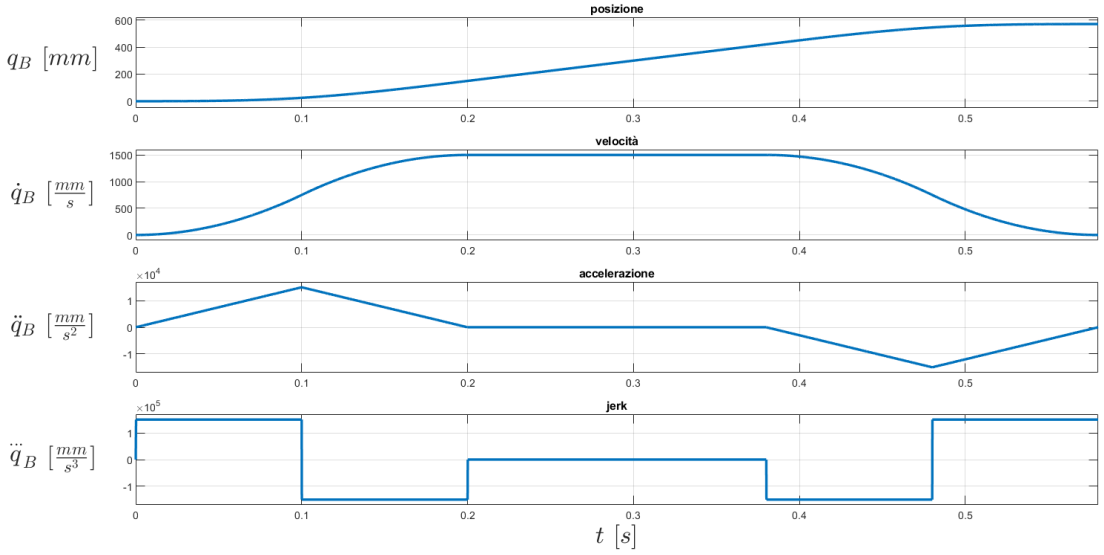


Figura 107 - Configurazione 4x40 con caricamento a passo 60 mm: alzata di 570 mm

$$T_B = 0{,}58\ s$$

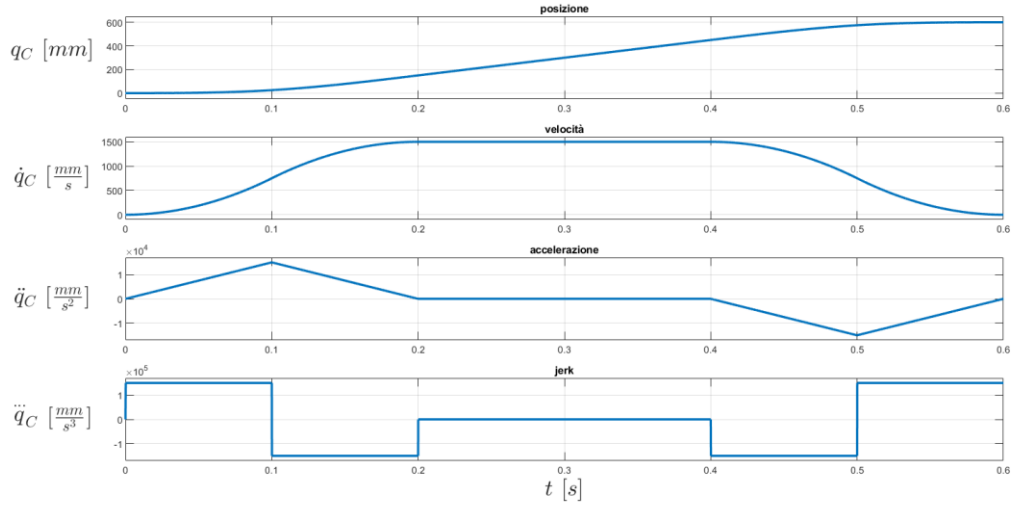


Figura 108 - Configurazione 4x40 con caricamento a passo 60 mm: alzata di 600 mm

$$T_C = 0{,}6\ s$$

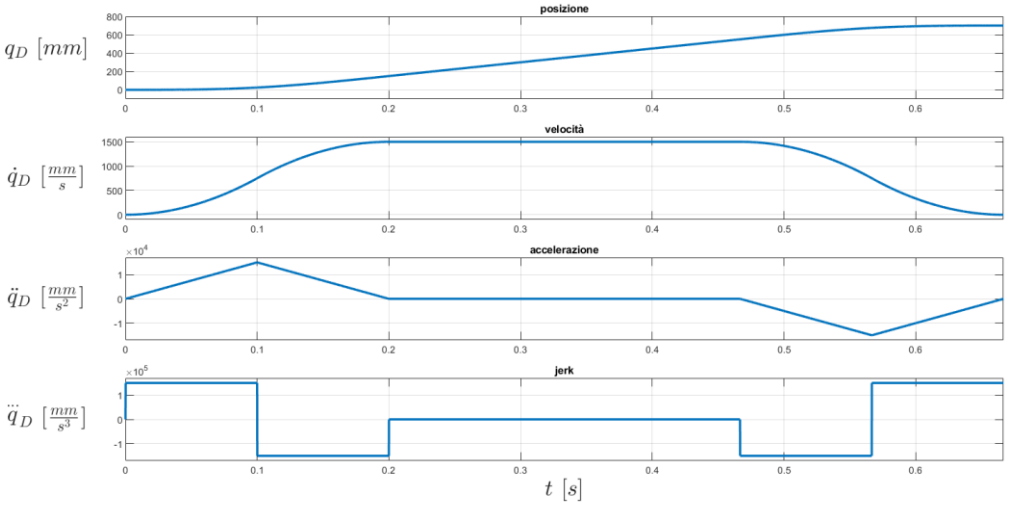


Figura 109 - Configurazione 4x40 con caricamento a passo 60 mm: alzata di 700 mm

$$T_D = 0{,}6666\ s$$

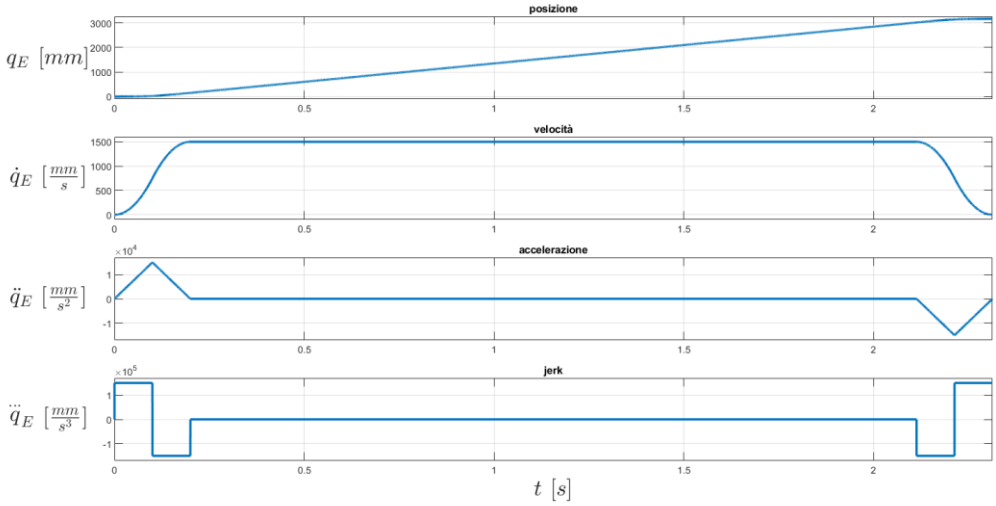


Figura 110 - Configurazione 4x40 con caricamento a passo 60 mm: alzata di 3170 mm

$$T_E = 2{,}3134\ s$$

5. Simulazione virtuale

5.1. Preparazione del modello – motori, sensori e leggi di moto

L'ultima fase del progetto consiste nella simulazione virtuale del sistema sincro-dinamico con l'obiettivo di valutare le prestazione di ognuna delle configurazioni selezionate dall'algoritmo descritto nella paragrafo 3.3.

Si utilizza il software PTC CREO per la modellazione geometrica 3D dei componenti principali costituenti il sistema. Il parco di componenti necessario da modellare è costituito da due telai per la rappresentazione delle cinghie di movimentazione, i cassetti da montare sulle cinghie, il robot di caricamento e gli spingitori che costituiscono i mezzi di interfaccia rispettivamente per la stazione di caricamento e la stazione di scaricamento (Figura 111, Figura 113 e Figura 114). Per le tre configurazioni con caricamento a passo di 60 mm è stata creata una variante del robot con distanza tra gli end-effector pari a 60 mm (Figura 112).

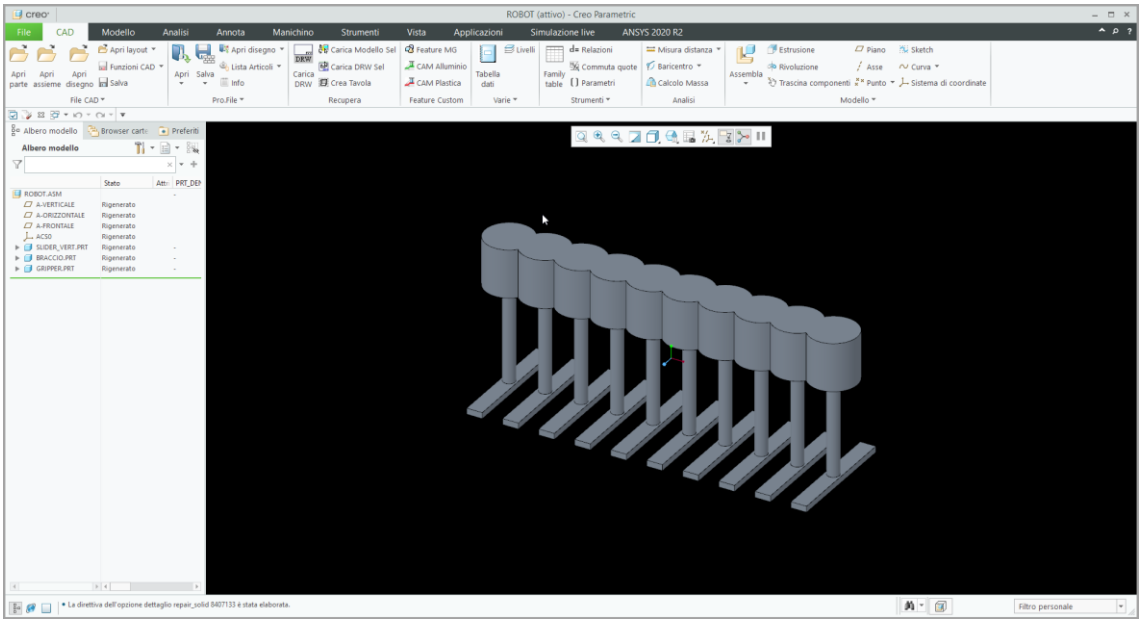


Figura 111 - Immagine CAD del robot di caricamento

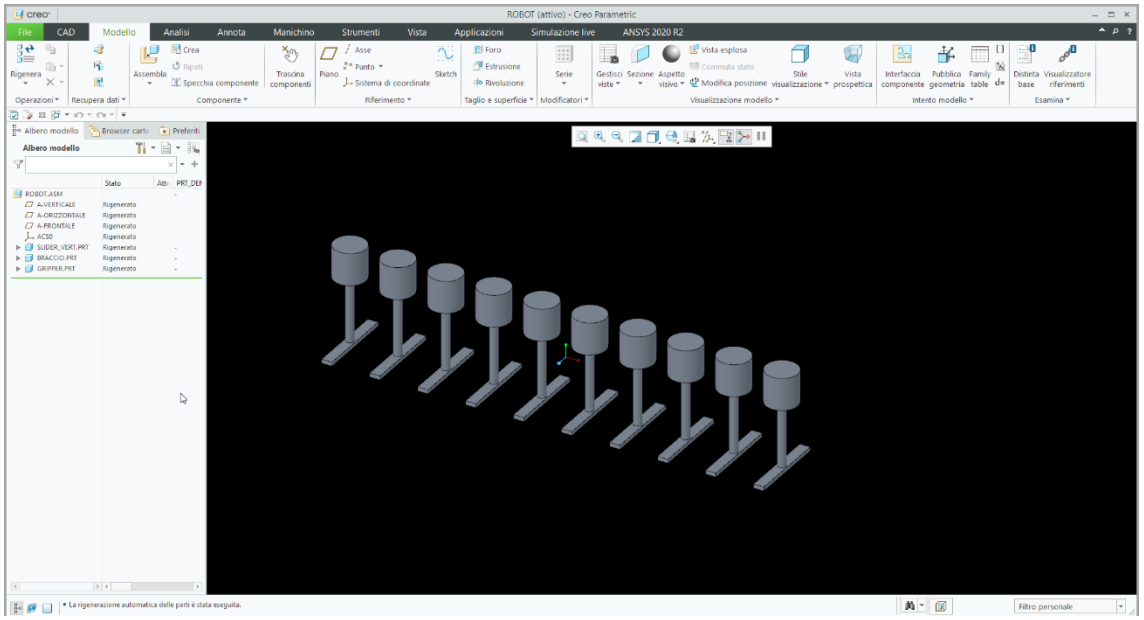


Figura 112 - Immagine CAD del robot di caricamento nella versione con passo di 60 mm

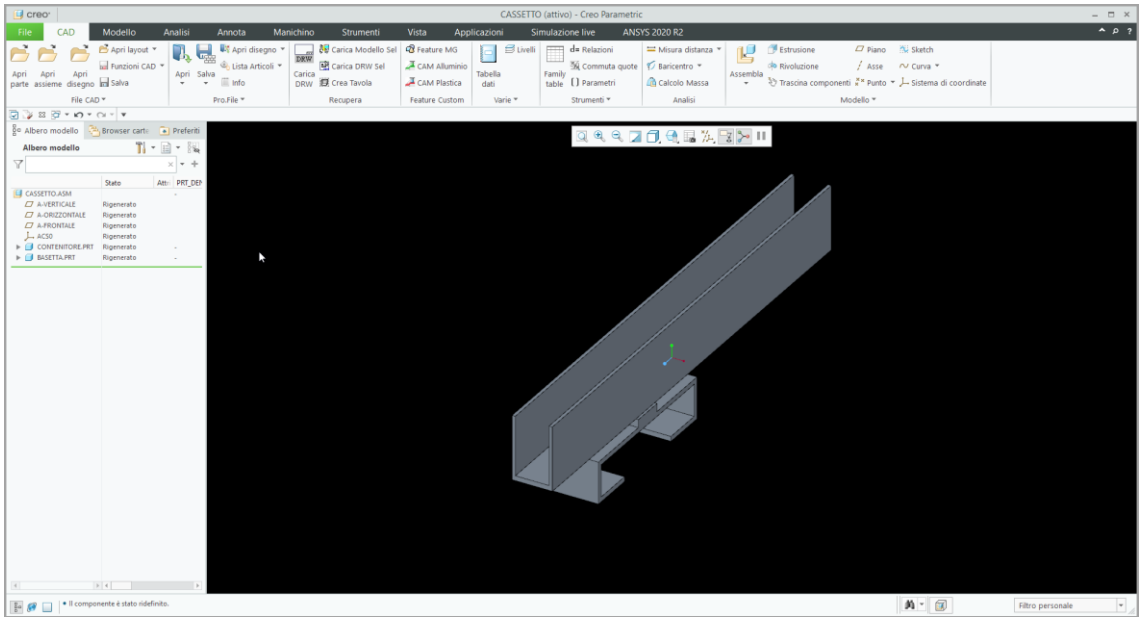


Figura 113 - Immagine CAD di un cassetto

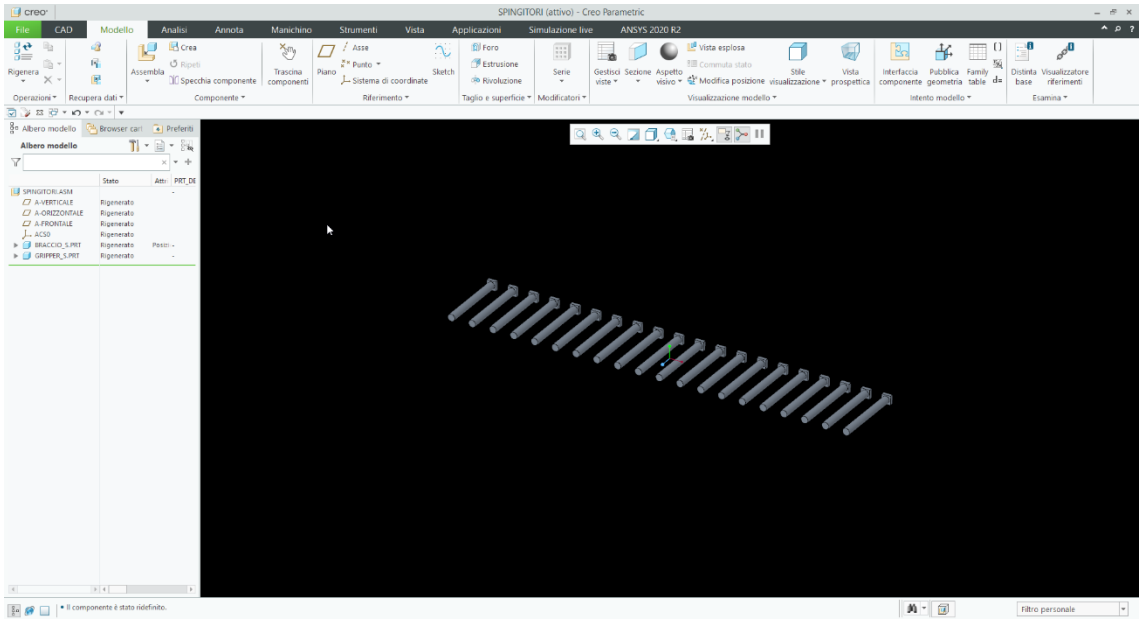
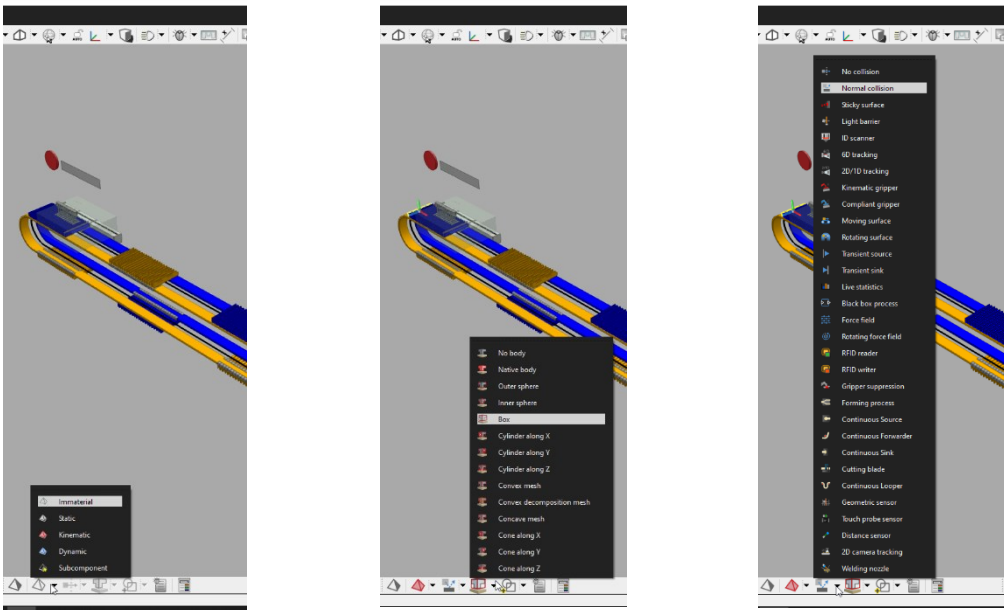


Figura 114 - Immagine CAD degli spingitori

Successivamente si importano i modelli CAD nel software di industrialPhysics e si aggiungono alcune geometrie molto semplici come rappresentazione simbolica di alcune funzionalità dedite allo svolgimento della simulazione virtuale.

In industrialPhysics è possibile attribuire quattro proprietà geometriche ai componenti della simulazione (tipologia di corpo) e diverse proprietà di carattere simulativo (tipologia di funzione e tipologia della mesh di ricoprimento) come si può vedere in Figura 115. Le proprietà geometriche sono molto importanti per il corretto svolgimento della simulazione per due motivi: pesantezza computazionale e corretta interpretazione della simulazione.



a) *b)* *c)*

Figura 115 - a) tipologia di corpo del componente; b) tipologia di mesh del componente; c) tipologia di funzione del componente

Un componente di tipo "*immaterial*" non ha nessuna interazione con il resto dei componenti. Componenti di questo tipo vengono utilizzati per scopi puramente visivi e stilistici.

Un componente di tipo "*static*" è il componente di default ed è un oggetto che non varia la sua posizione rispetto al sistema di riferimento locale e pertanto non è possibile attribuire nessuna legge di moto. Un componente di tipo *static* è rivestito da una mesh che interagisce geometricamente con i componenti di tipo "*dynamic*".

Un componente di tipo "*kinematic*" può muoversi rispetto all'origine del suo sistema di riferimento seguendo la legge di moto impostata, la mesh può interagire con i componenti "*dynamic*".

Un componente di tipo "*dynamic*" è dotato di sei gradi di libertà, può instaurare dei vincoli cinematici con corpi di tipo "*kinematic*" e durante la simulazione viene istantaneamente calcolato l'effetto di forze di inerzia, gravità ed eventuali altri campi di forza.

5.1.1. Componenti della simulazione

IndustrialPhysics ha una interfaccia grafica simile ad un programma di modellazione CAD con un albero modello che elenca tutti i componenti della simulazione e la loro gerarchia strutturale (Figura 116). Nella simulazione si utilizzeranno come elementi gerarchici di alto livello dei sistemi di riferimento locali: in questo modo si può impostare una legge di moto al sistema di riferimento e posizionando un componente in posizione gerarchica inferiore si può vincolare rigidamente il componente al sistema di riferimento. In questo modo il sistema di riferimento farà muovere anche il componente con la stessa traiettoria. Questo approccio faciliterà la modifica del modello per simulare le diverse configurazioni di sistema, poiché ad esempio sarà sufficiente importare una diversa geometria del robot di caricamento mantenendo comunque tutte le relazioni con gli altri componenti che sono invece relazionati esclusivamente con il sistema di riferimento di livello più alto.

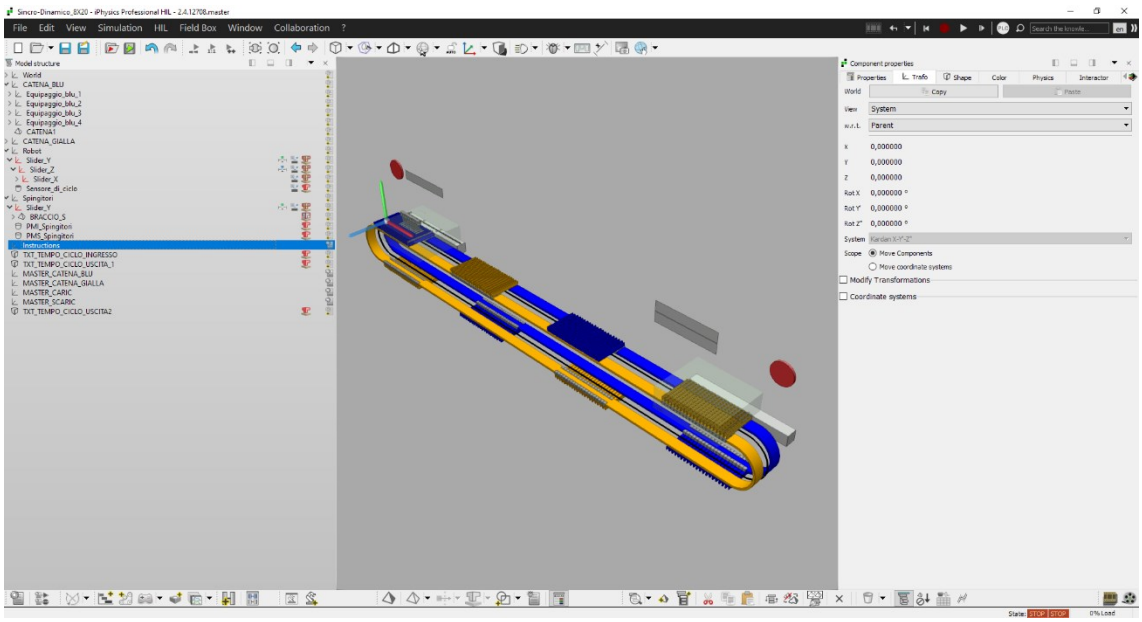


Figura 116 - Interfaccia grafica del software industrialPhysics; a sinistra è presente l'albero modello, nella zona centrale sopra il modello 3D è visualizzato il sistema di riferimento inerziale (direzione X in rosso, direzione Y in verde e direzione Z in azzurro)

Internamente ad industrialPhysics sono state aggiunte due strutture che rappresentano simbolicamente la parte finale della macchina a monte e la parte iniziale della macchina a valle. Queste due strutture servono come strumento di generazione dei prodotti per la stazione di caricamento e come uscita dei prodotti per la stazione di scaricamento.

Una di queste due strutture è la zona di prelievo prodotti (Figura 117). Attraverso il comando "*add a chain*" è stato modellato un sistema di palette che serve per trasportare i prodotti mantenendo il controllo sulla posizione, in questo modo si garantisce che i prodotti rimangano a passo. Questa scelta è fondamentale: se i prodotti non fossero al passo corretto il robot non sarebbe in grado di afferrare un set di dieci prodotti tutti assieme. La funzione "*chain*" permette di inserire un azionamento della tipologia catena/puleggia per permettere il movimento delle palette.

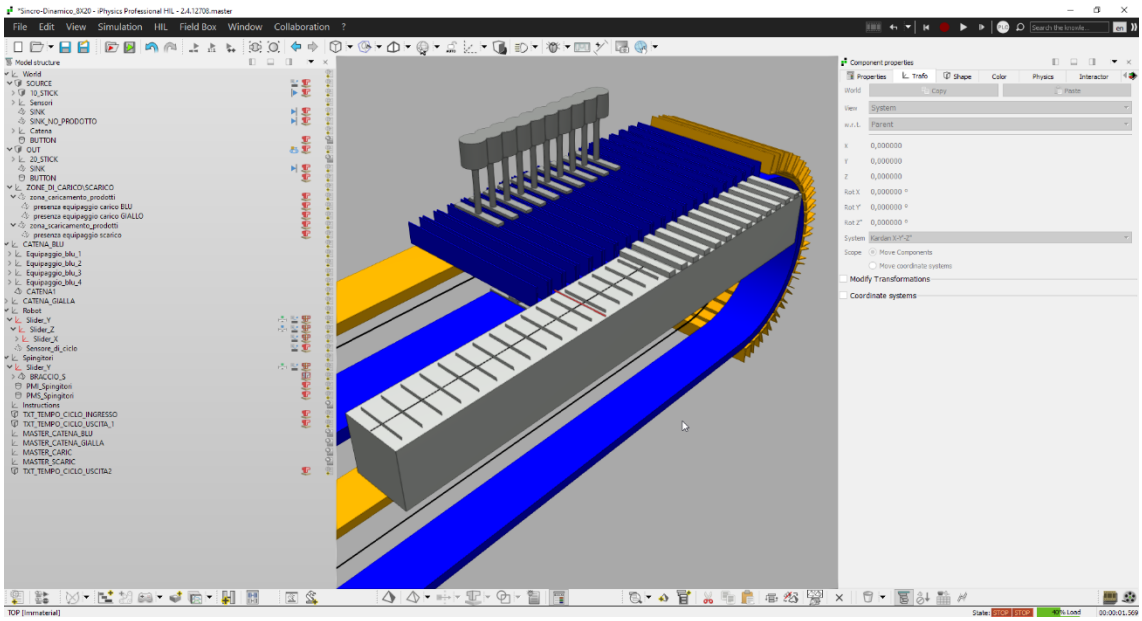


Figura 117 - Zona di prelievo con catena di palette per mettere al passo i prodotti in ingresso

Nella zona di prelievo è stato inserito il componente "*10_STICK*" che rappresenta un singolo prodotto. A questo componente è stata attribuita la proprietà "*transient source*", che permette la generazione di copie del componente con tipologia di corpo *dynamic*. Questa funzionalità è stata impostata per generare automaticamente delle copie con una frequenza preimpostata, ma può anche essere attivata/disattivata tramite segnale esterno.

Un altro tipo di componente inserito è "*SINK*", con la funzionalità "*transient sink*" che permette di eliminare dalla simulazione un componente. Le copie generate, una volta finito il loro compito, sono eliminate in modo da mantenere snella la simulazione.

Infine, sono stati inseriti anche dei componenti con la funzionalità "*light barrier*", che rappresentano un sensore a fascio laser (Figura 118). Questi oggetti sono dotati di uno stato booleano true/false che rileva la presenza o meno di un oggetto quando questo attraversa la geometria del sensore. Lo scopo di questi sensori è di trasmettere l'informazione al robot di caricamento relativamente alla presenza di dieci prodotti consecutivi nella zona di prelievo. Un altro sensore laser è stato utilizzato per rilevare la posizione del robot in modo da calcolare il tempo ciclo delle sue operazioni.

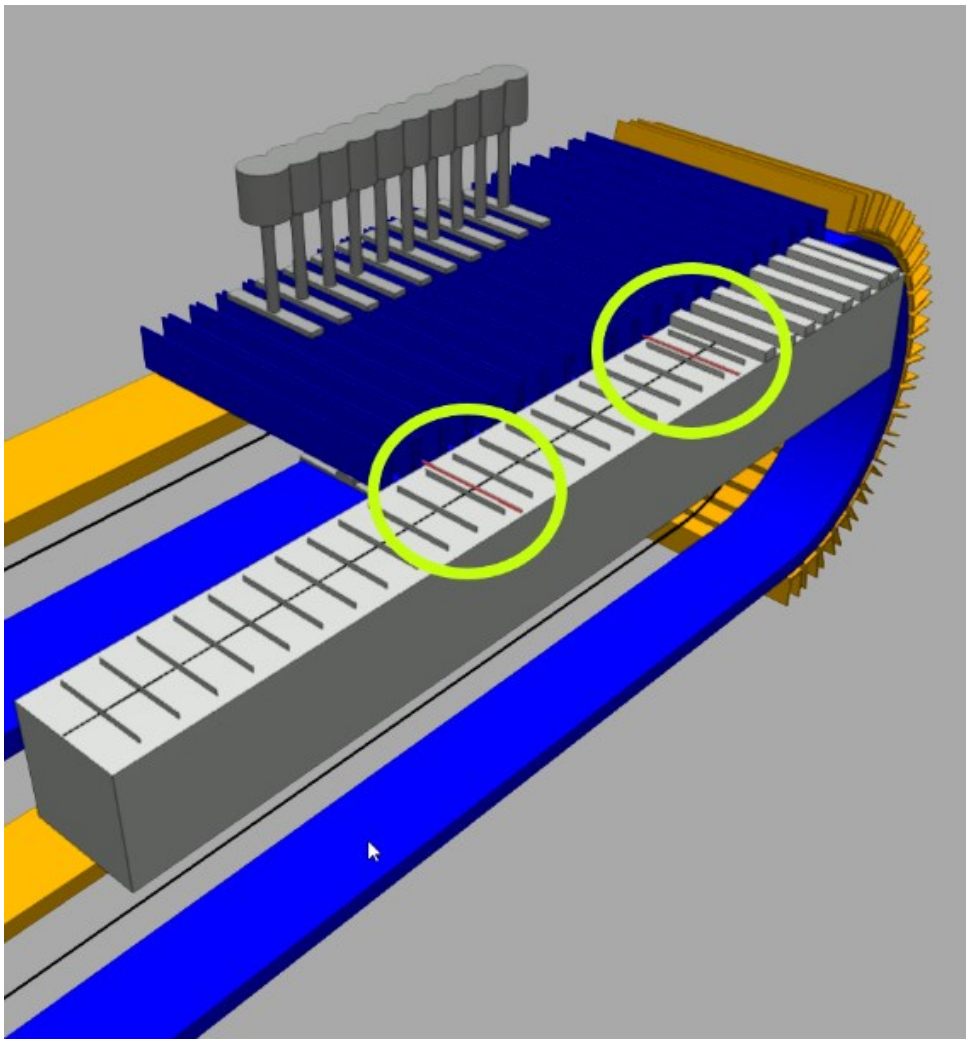


Figura 118 - Rappresentazione dei sensori laser. In figura sono rappresentati dei sensori laser per verificare la presenza prodotto nella zona di prelievo.

Nella zona di consegna prodotti è stato inserito un oggetto di tipo *transient source* per generare 20 prodotti in uscita dal sistema sincro-dinamico. I prodotti vengono generati quando gli spingitori giungono al punto morto inferiore. Si sottintende quindi che i prodotti non vengono fisicamente prelevati dal robot, trasportati dai cassetti e consegnati dagli spingitori, ma compaiono e scompaiono nelle due stazioni seguendo i movimenti degli equipaggi e la frequenza di generazione impostata. Questa scelta simulativa è dettata dal fatto che la modellazione completa di questo processo non avrebbe portato informazioni aggiuntive alla risoluzione del problema, ma avrebbe aggravato il carico computazionale. Anche nella zona di consegna sono presenti dei sensori laser per identificare la posizione degli spingitori.

Le due cinghie di movimentazione del sistema sincro-dinamico sono state modellate utilizzando ancora una volta la funzione *chain*. Questa funzione permette di definire un azionamento di tipo *chain*: si definisce una curva chiusa costituente un vincolo di percorso degli oggetti. All'interno di questa funzione è possibile aggiungere un certo numero di componenti di tipo *kinematic*. I componenti aggiunti erediteranno la posizione iniziale sulla curva a partire dalla loro disposizione spaziale nel modello e verranno movimentati seguendo la curva chiusa preimpostata. Per ciascun azionamento chain, tramite la funzione sono stati vincolati i cassetti al loro azionamento di riferimento (Figura 119).

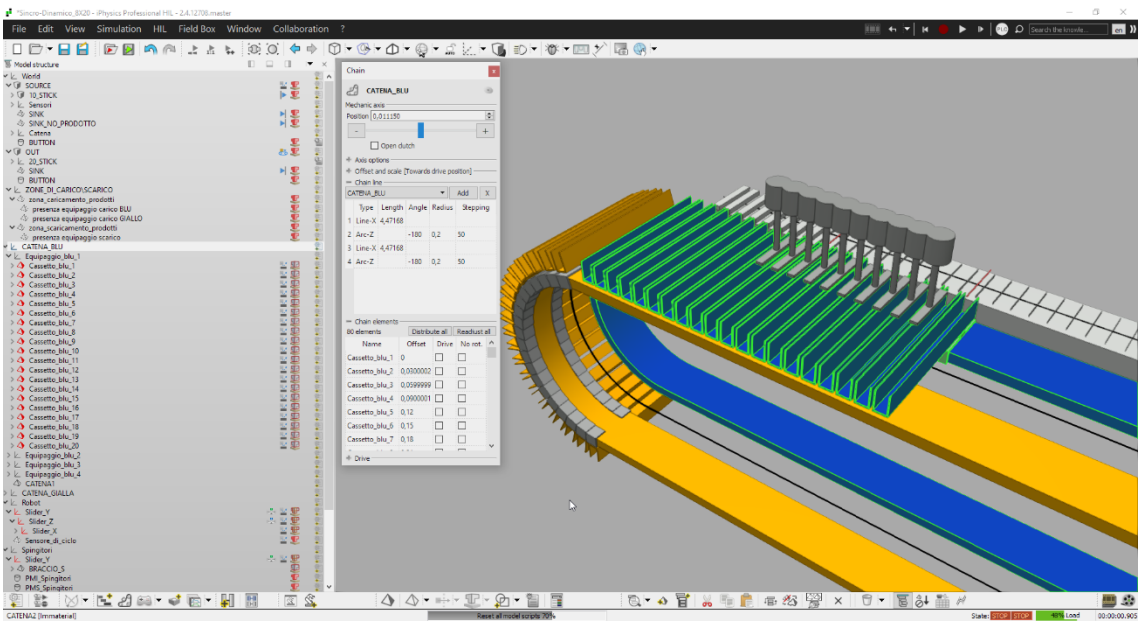


Figura 119 - Funzione chain e inserimento dei componenti "cassetto" al suo interno.

Nell'albero modello sono stati inseriti alcuni sistemi di riferimento che non hanno alcuna interazione con il resto dei componenti della simulazione, il loro unico scopo è esclusivamente quello di generare un "*model script*", ovvero una serie di istruzioni scritte con un linguaggio conforme allo standard utilizzato per la programmazione dei PLC. Questi model script verranno utilizzati per gestire l'intera simulazione attraverso una logica di programmazione a stati.

5.1.2. Azionamenti

Per poter generare il movimento dei componenti, il software permette di implementare una serie di azionamenti che simulano i motori reali di una macchina automatica o robot.

Nella simulazione sono stati modellati sei azionamenti: due azionamenti per il robot di caricamento, uno per gli spingitori e tre azionamenti di tipo puleggia, di cui uno per ciascuna delle due cinghie di movimentazione del sistema sincro-dinamico e l'ultimo per la messa al passo dei prodotti nella zona di prelievo.

Gli azionamenti vengono assegnati tramite la funzione *chain* per quanto riguarda le pulegge e attraverso la funzione "*kinematic*" per assegnare dei movimenti traslatori o rotativi lungo una direzione. Per compiere movimenti nel piano e nello spazio è necessario assegnare più azionamenti in direzioni diverse.

Una volta assegnato l'azionamento è necessario stabilire che tipologia di controllo del motore utilizzare. Tra tutte le tipologie di controllo le due utilizzate nella simulazione sono "*Direct drive*" e "*Bit drive*".

- *Direct drive*: si controlla il motore in posizione. Vengono associati dei parametri al componente e si controllano questi parametri tramite model script o PLC esterno;
- *Bit drive*: si controlla il motore in velocità costante impostando un modulo della velocità e un verso.

L'azionamento di tipo *chain* che mantiene al passo e movimenta i prodotti nella zona di prelievo è comandato impostando un motore di tipo *bit drive*, sempre attivo e con valore costante di velocità: l'unico scopo di questo azionamento è quello di fornire un costante input di prodotti al robot di caricamento e non c'è necessità di utilizzare logiche di controllo più complesse (Figura 120).

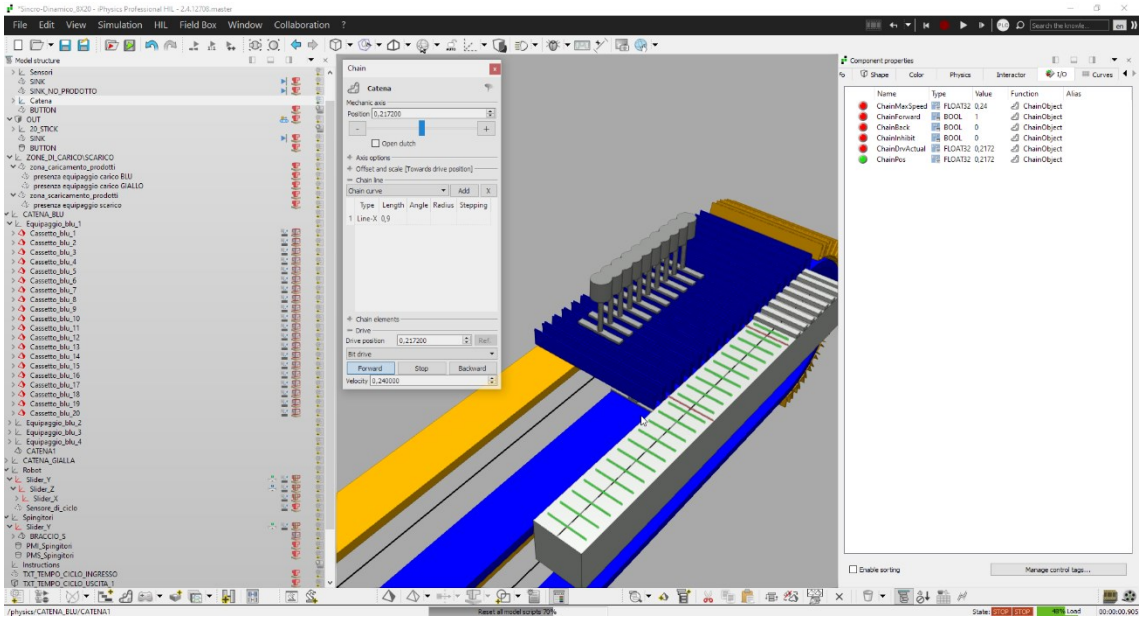


Figura 120 - Motore utilizzato per l'azionamento delle palette di messa al passo prodotti

Per i restanti azionamenti è necessario impostare delle leggi di moto. Vengono quindi controllati in posizione assegnando dei controlli di tipo *direct drive*.

Nella simulazione è impostato un sistema di riferimento inerziale XYZ e diversi sistemi di riferimento locali xyz per ciascun componente. Per impostare un azionamento traslatorio o rotatorio bisogna scegliere la direzione del movimento rispetto al sistema di riferimento locale del componente. In questa simulazione i sistemi di riferimento locali dei componenti sono stati orientati in maniera concorde a quello inerziale.

Il robot ha un azionamento traslatorio nella direzione Y e un azionamento traslatorio nella direzione Z (Figura 121).

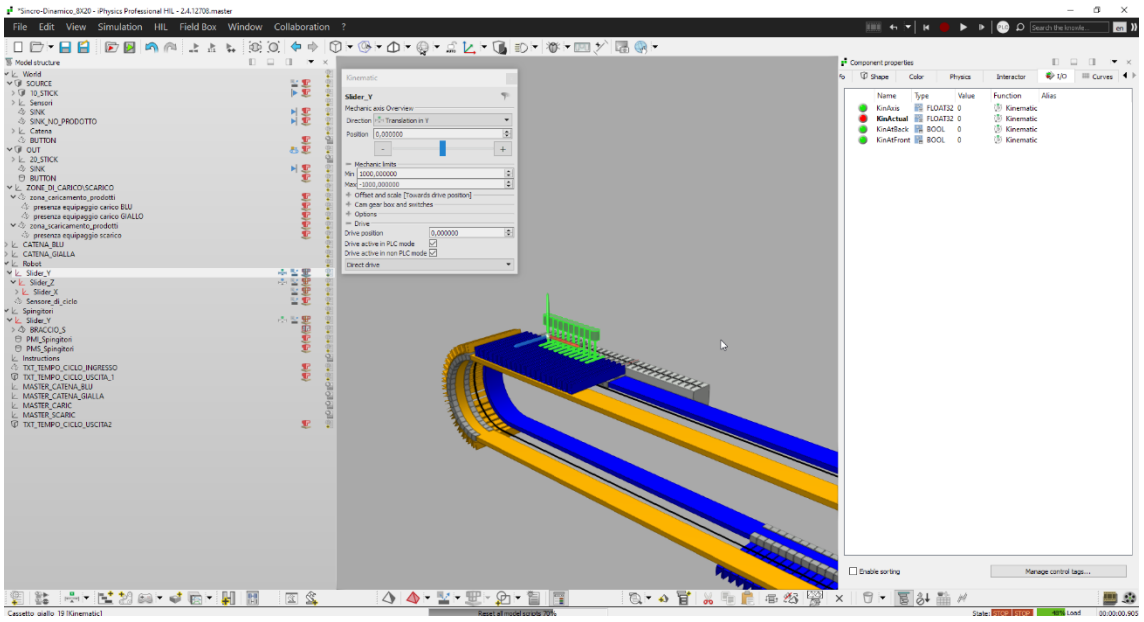


Figura 121 – Motore utilizzato per il robot di caricamento. In figura è visualizzato il motore che genera il movimento lungo l'asse Y.

L'azionamento lungo Y serve per effettuare il movimento verticale del robot di prelievo prodotti e deposito prodotti. L'azionamento lungo Z serve per far spostare il robot dalla zona di prelievo verso l'equipaggio e viceversa.

Agli spingitori è stato assegnato un azionamento traslatorio lungo la direzione Z.

Per le cinghie di movimentazione degli equipaggi sono stati impostati due azionamenti di tipo *chain* (Figura 122). Quando si assegna un azionamento *chain* si deve definire una curva chiusa come percorso per i componenti della catena. La curva viene definita a partire dall'origine del sistema di riferimento del componente. È possibile impostare manualmente la posizione degli elementi sulla curva rispetto all'origine della stessa oppure si può impostare un offset iniziale dall'origine.

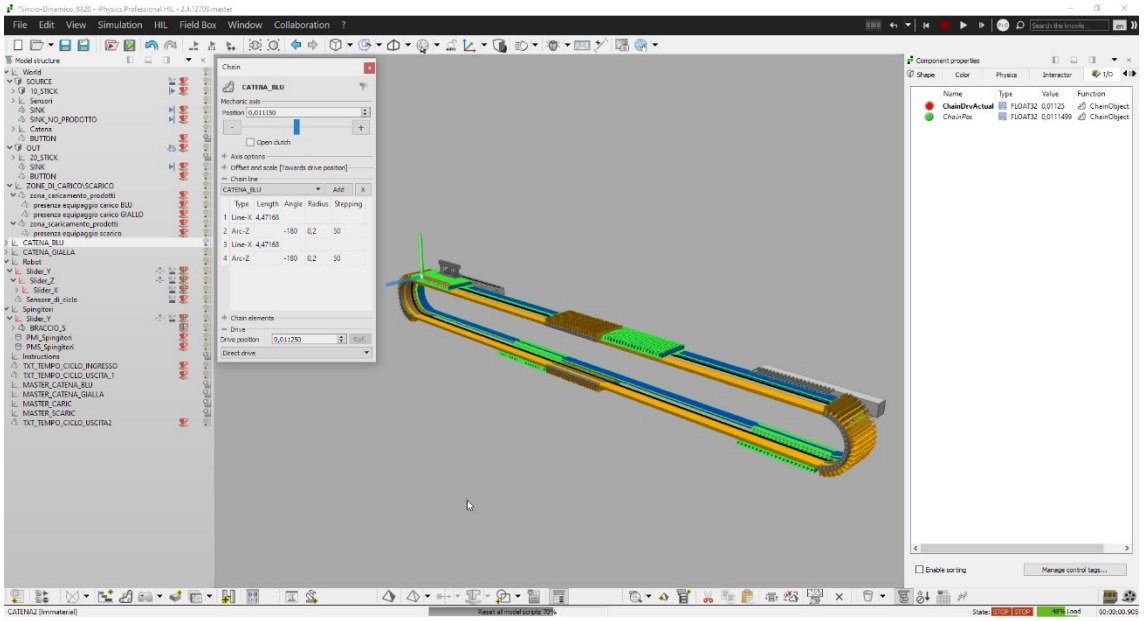


Figura 122 – Motore utilizzato per la movimentazione della cinghia blu.

5.1.3. Implementazione delle leggi di moto

Quando si assegna un azionamento ad un componente, nella sua scheda proprietà si popola automaticamente la sezione degli input/output definita come "*I/O*" (Figura 123). Questa sezione può ospitare due tipi di parametri:

- I parametri di colore verde sono dei valori in uscita dal componente che forniscono informazioni su un determinato parametro del componente, come per esempio potrebbe essere la posizione rispetto al sistema di riferimento locale o un parametro booleano. Questi parametri non sono modificabili dall'utente.
- I parametri di colore rosso sono dei valori in ingresso al componente che assegnano o sovrascrivono determinati parametri del componente, come ad esempio potrebbero essere i parametri booleani che attivano o disattivano un motore o il modulo della velocità. Questi parametri sono trattati come delle variabili e possono essere modificati dall'utente manualmente o collegando la variabile ad uno script.

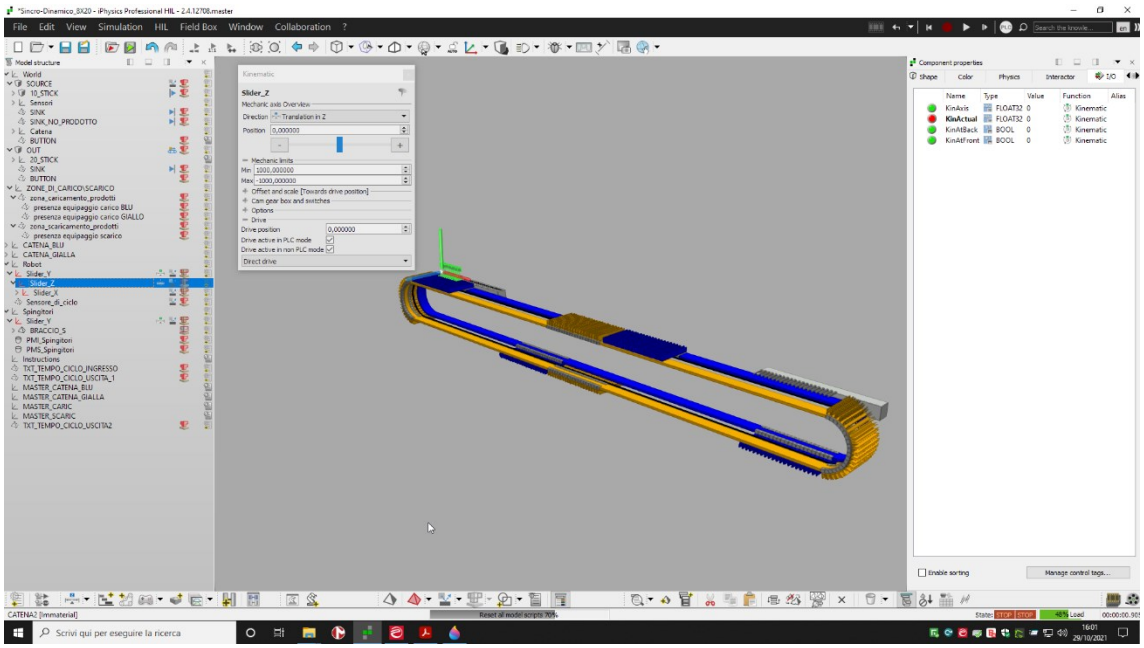


Figura 123 - Parametri I/O di un motore, visibili nella parte destra dell'immagine nelle proprietà componente

In base al tipo di controllo del motore assegnato, la scheda I/O del componente può popolarsi di parametri differenti, poiché la tipologia di controllo lavora diversamente.

In una prima fase di prototipazione del modello è utile utilizzare lo strumento "*sequence control*".

Tramite l'apposito comando si apre una finestra rappresentante un asse master per l'assegnazione di una o più leggi di moto. In questa finestra è possibile aggiungere uno o più assi slave a cui collegare gli azionamenti (Figura 124). Per ogni asse slave è possibile creare un profilo di posizione definendo una serie di punti iniziali e finali in vari intervalli temporali. All'avanzare dell'asse master tutti gli slave seguiranno il profilo di moto stabilito mantenendo la stessa fase.

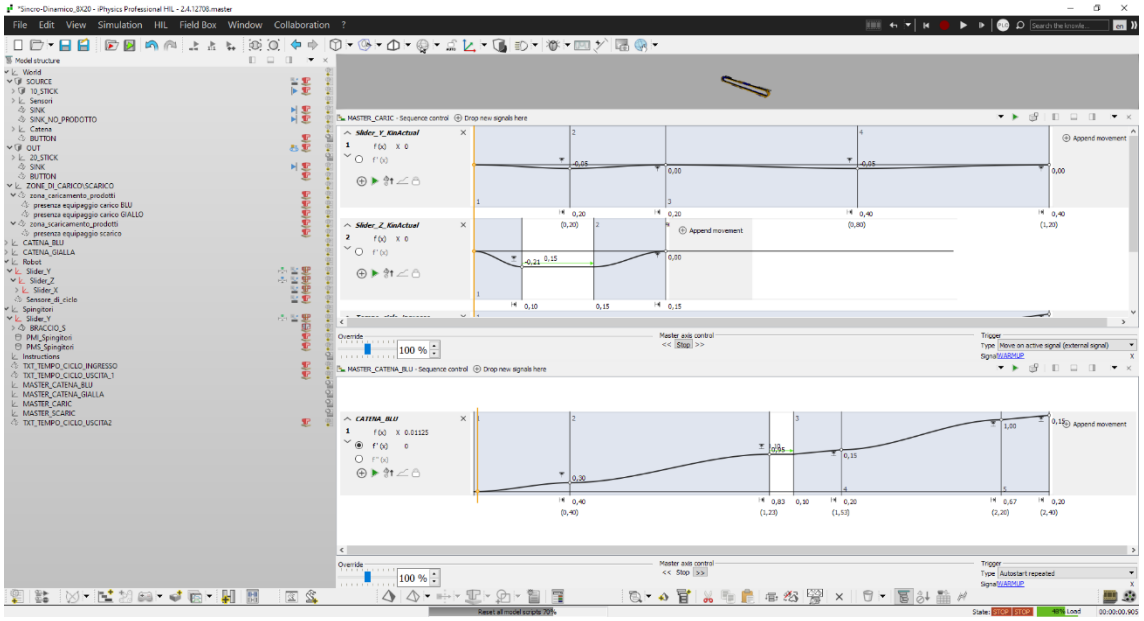


Figura 124 - Sequence control; in figura ci sono due assi master, il master "MASTER_CARIC" contiene due assi slave.

I profili di moto che è possibile creare vengono definiti soltanto tramite l'assegnazione di punti iniziali e finali e possono essere soltanto profili polinomiali del 3°, 5° e 9° ordine, profili lineari e profilo armonico.

Una volta definita la legge di moto si può assegnare il movimento di un componente semplicemente trascinando la variabile "*KinActual*" dalla scheda I/O del componente verso l'asse su cui è stata creata la traiettoria (Figura 125). Il parametro *KinActual* è il parametro che comanda in posizione l'azionamento del componente.

Attraverso questo collegamento il motore comanderà la posizione del componente seguendo il profilo descritto dal *sequence control*.

Il *sequence control* può essere eseguito solo una volta oppure eseguito ogni volta che un segnale ad esso collegato diventi *TRUE*. Alternativamente è possibile anche che venga eseguito in maniera ripetitiva utilizzando un segnale a dente di sega: il modulo della funzione contenuta nel *sequence control* stabilisce il valore massimo di un segnale a dente di sega che resetta automaticamente la funzione al punto iniziale, ma incrementando il suo argomento (la posizione della traiettoria) a partire dal suo precedente valore.

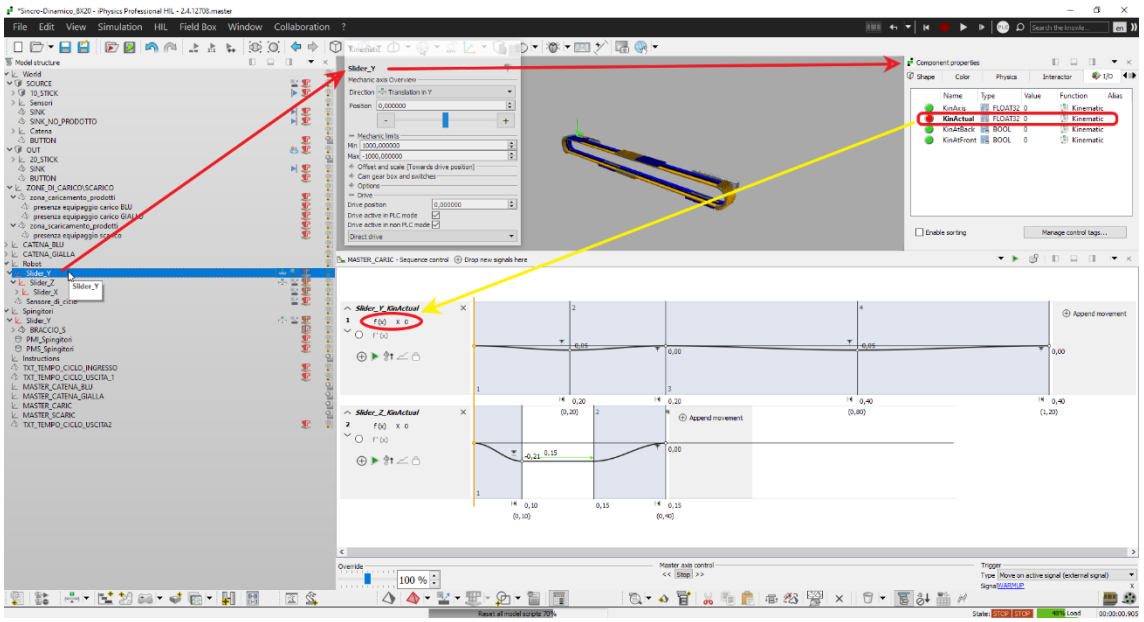


Figura 125 - Collegamento del parametro KinActual con il sequence control

Per poter implementare una logica a stati è necessario utilizzare degli elementi di supporto che possano azionare o fermare un asse in base a ciò che accade durante la simulazione. Si aggiungono quattro diversi componenti fittizi alla simulazione che fungeranno da supporto per la logica a stati. Questi componenti fittizi hanno l'unico scopo di contenere degli script che gestiranno le operazioni da svolgere in base allo stato in cui si trova la macchina istante per istante.

Il *sequence control* viene eseguito in maniera ripetitiva utilizzando il segnale a dente di sega e può essere fermato e azionato da uno di questi script.

Per poter azionare o fermare un asse è necessario impostare tramite script il parametro "*MasterInhibit*" presente nelle proprietà del *sequence control*. Quindi, all'interno del componente fittizio ci sarà un codice che imposterà come *TRUE* il parametro *MasterInhibit* per fermare l'asse master, oppure come *FALSE* per azionare l'asse master (Figura 126).

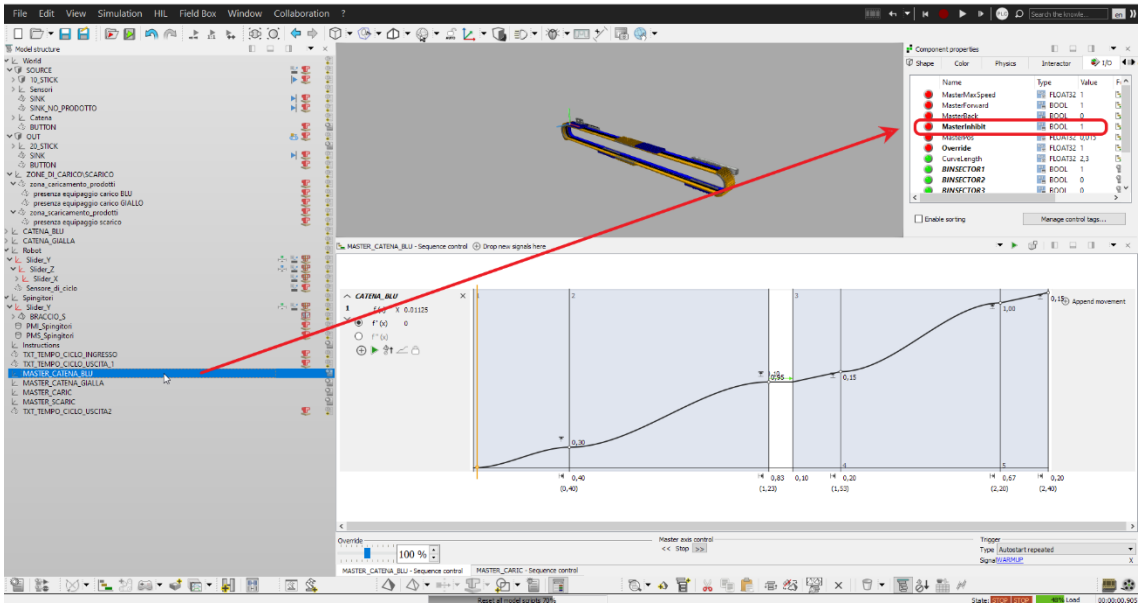


Figura 126 – Collegamento parametro MasterInhibit

Due script sono adibiti per il controllo delle due cinghie mentre gli altri due servono per il controllo del robot e degli spingitori.

Il master che controlla gli spingitori ha un solo asse slave che controlla la posizione dell'azionamento traslatorio lungo la sola direzione Z (Figura 127).

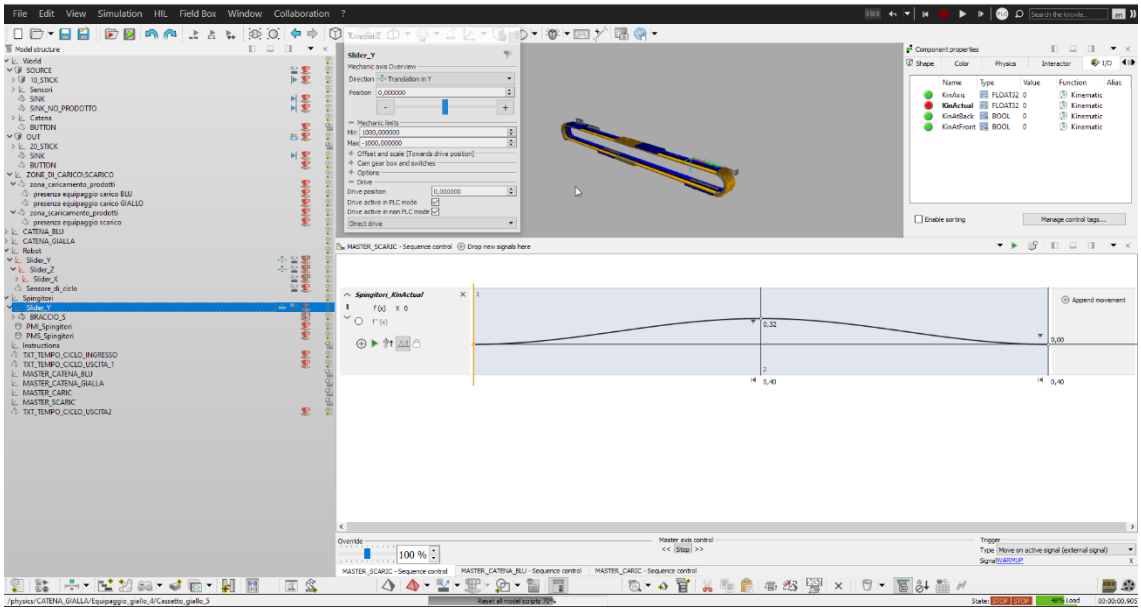


Figura 127 – Asse master spingitori. Al suo interno c'è solo un asse slave

La durata totale della corsa corrisponde al tempo comune di operazione di scaricamento tra equipaggio e spingitori, pari a 0,8 secondi.

Nell'asse master del robot di caricamento sono stati creati due slave a cui sono assegnati i due azionamenti lungo Y e lungo Z. Le leggi di moto create sono state studiate in modo che lo spostamento lungo Z avvenga solo quando il robot deve prelevare i prodotti dalla zona di prelievo, mentre quando deve depositarli nell'equipaggio si può muovere solo lungo la direzione Y (Figura 128).

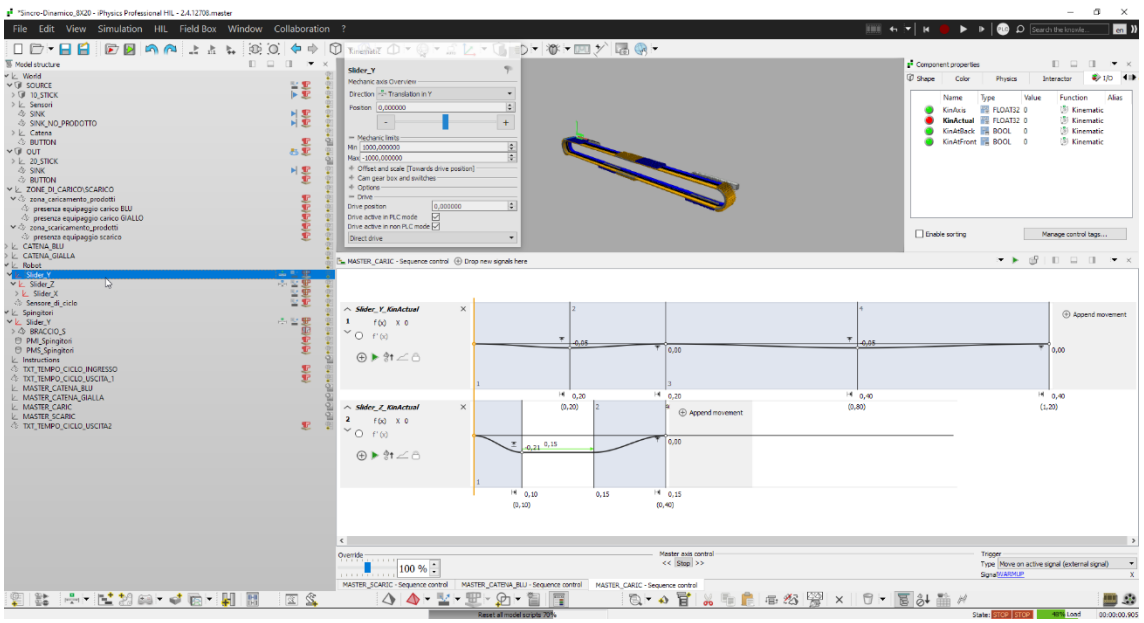


Figura 128 – Asse master robot. Al suo interno ci sono due assi slave

La corsa totale del robot corrisponde al tempo di operazione di 1,2 secondi di cui 0,8 secondi sono la corsa che riguarda il solo deposito prodotti nell'equipaggio.

Le due cinghie di movimentazione hanno un asse master ciascuno, ma il funzionamento è analogo per entrambe.

In questi assi viene creato un solo asse slave con un profilo di moto che gestirà tutti i movimenti degli equipaggi appartenenti ad uno dei due motori.

In una prima fase di prototipazione si può utilizzare lo strumento *sequence control* per la sua rapidità di utilizzo e chiarezza grafica. La traiettoria si costruisce impostando i punti iniziali e finali di diverse alzate e accostando in sequenza tutte le alzate nello stesso asse

slave. Ogni alzata approssima una delle traiettorie calcolate nel paragrafo 4.3 (Figura 129).

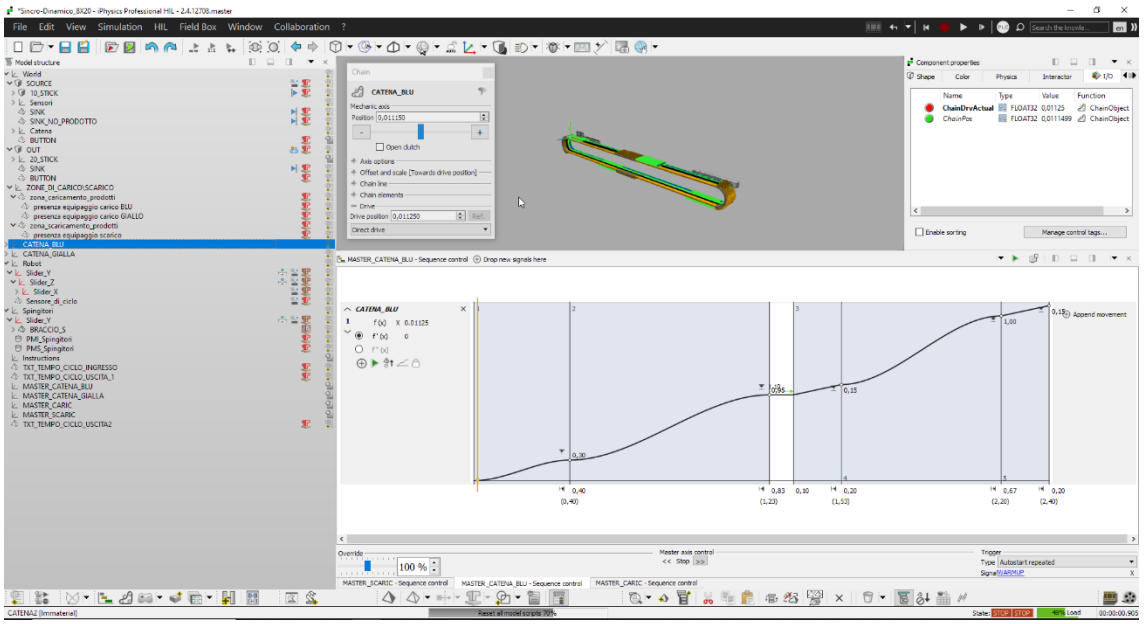


Figura 129 - Asse master cinghia blu. Al suo interno c'è un solo asse slave

La potenza di questo strumento consiste nel poter impostare una sequenza di alzate e testare immediatamente il movimento degli equipaggi eseguendo il *run* della simulazione. Se il movimento è scorretto si può rapidamente cambiare la sequenza delle alzate e far ripartire la simulazione. Si può facilmente cambiare la distanza percorsa dalle alzate o il tempo impiegato modificando i valori iniziali e finali degli intervalli, anche in real time durante la simulazione stessa. Si possono anche aggiungere dei tempi di attesa tra una alzata ed un'altra per poter quantificare più accuratamente la presenza di eventuali tempi di ozio durante il ciclo totale delle operazioni.

Una volta finita la fase di prototipazione si può eliminare la funzione *sequence control* delle cinghie di movimentazione e utilizzare un "*virtual axis*" attraverso la funzione *Kinematic* (Figura 130). Questo asse virtuale rappresenta anch'esso un asse master e presenta le stesse proprietà di un azionamento, potendone impostare una logica di controllo.

Attraverso un *model script* si collega il parametro *KinActual* dell'azionamento *chain* con il parametro *KinActual* del *virtual axis*. In questo modo si controlla automaticamente

la posizione della cinghia attraverso il *virtual axis* esattamente come nel caso del *sequence control*. Utilizzando il *virtual axis* è possibile importare, attraverso un file .txt tabulato o un file di tipo .csv, una sequenza di punti rappresentativa di una legge di moto personalizzata (Figura 131).

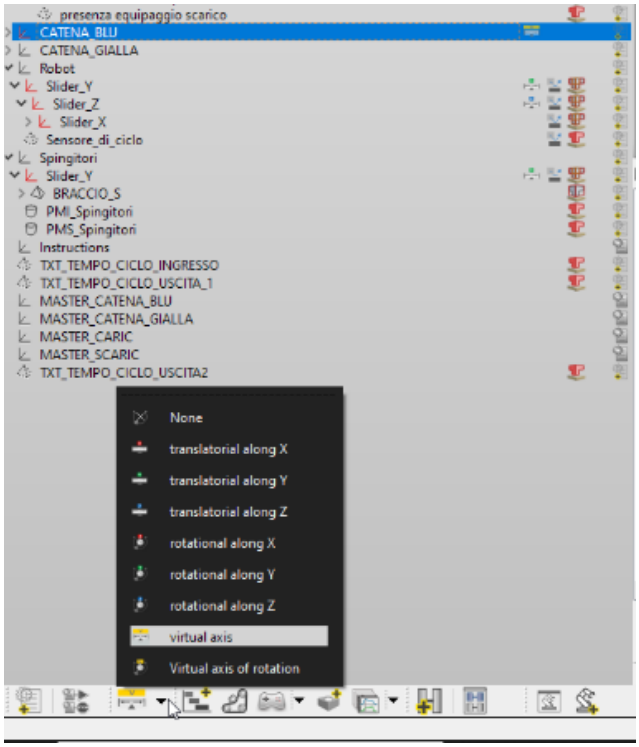


Figura 130 - Funzione virtual axis

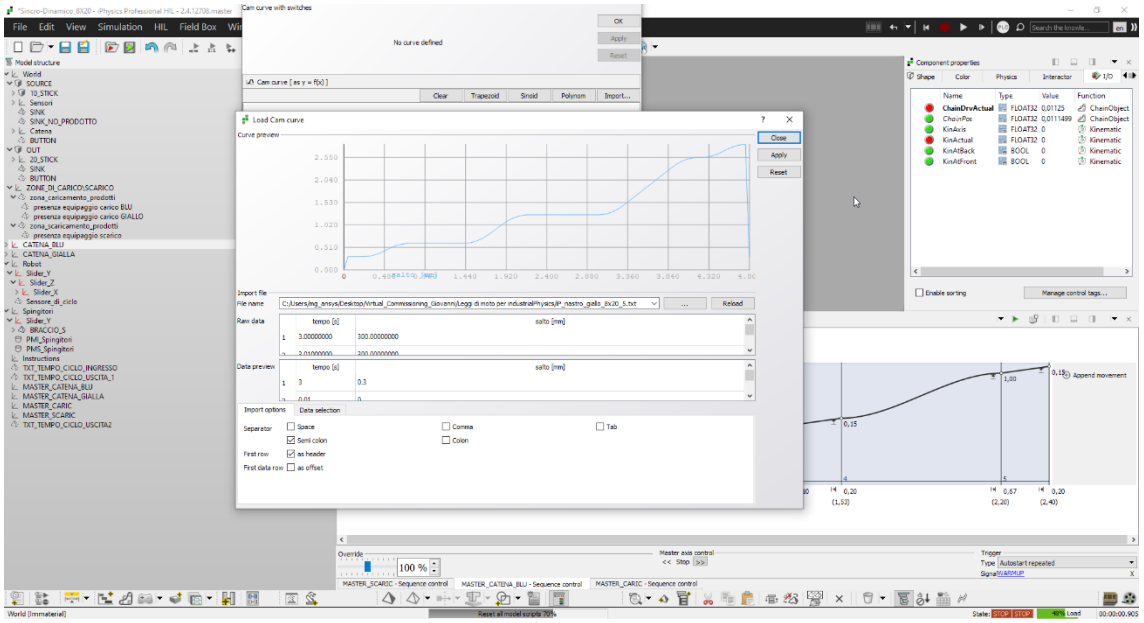


Figura 131 - Importazione di una legge di moto personalizzata tramite virtual axis

Attraverso MATLAB si esportano in un file excel i punti costituenti le varie traiettorie e si accostano questi intervalli di punti nella giusta sequenza come studiato in precedenza nel *sequence control*. Dopodiché si importa tale file nel *virtual axis* e si ottiene una legge di moto personalizzata per il componente cinghia. Così facendo si può implementare la legge di moto reale ed eseguire la simulazione del sistema.

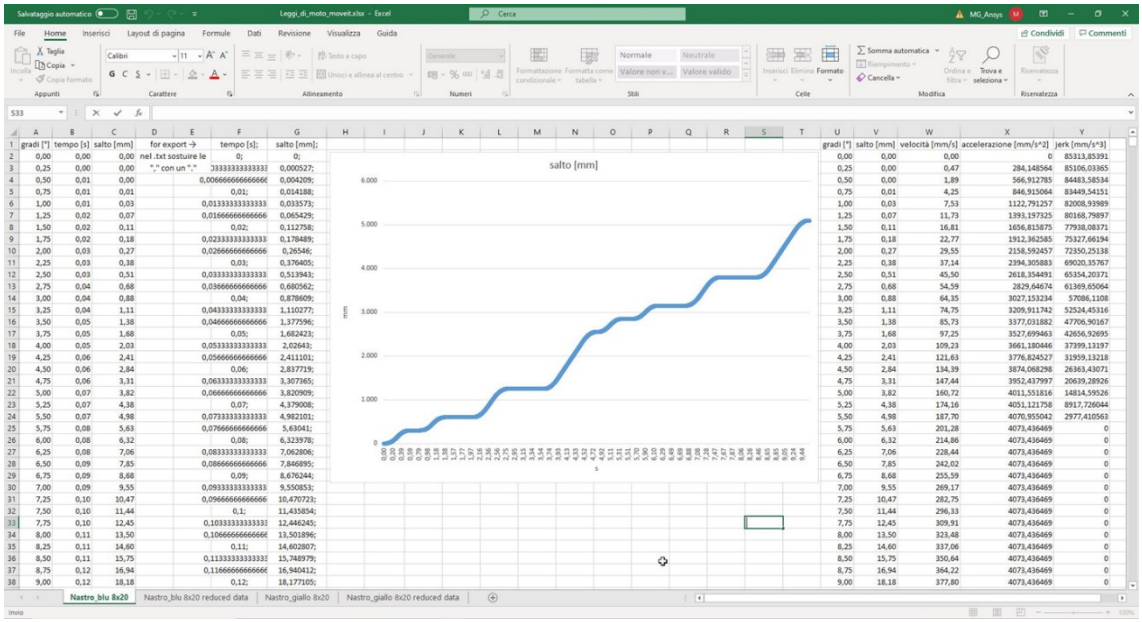


Figura 132 - File excel per esportare la legge di moto da MATLAB verso industrialPhysics

Per poter svolgere correttamente la simulazione del modello è necessario impostare una logica generale di controllo della macchina utilizzando i *model script*. Per garantire una simulazione robusta ed efficiente si è utilizzata la logica di programmazione di macchina a stati.

Utilizzando solo ed esclusivamente una composizione di alzate per generare la traiettoria completa degli equipaggi si può incorrere in diverse problematiche:

- Suscettibilità ad errori. La simulazione non è robusta, la sequenza delle operazioni è troppo deterministica e non è possibile simulare eventi casuali come ritardi o mancato arrivo di un prodotto. Gli equipaggi continueranno il loro movimento ignorando qualunque condizione al contorno della simulazione. Prima di raggiungere la sequenza di operazioni di regime è necessario affrontare una sequenza di operazioni warm-up diversa dalla sequenza di regime, complicando di molto la costruzione del modello.
- Perdita di automazione del processo. La sequenza di operazioni deve essere tarata su una specifica configurazione di macchina impostando manualmente ogni singolo periodo di attesa degli equipaggi che potrebbe cambiare in seguito a modiche sulla configurazione o modifiche al modello sulla stessa

configurazione. Per ogni modifica si è costretti a cambiare e controllare tutti i valori di attesa impostati nella sequenza di alzate. I periodi di attesa potrebbero essere dovuti a tempi di ozio oppure causati dall'esecuzione delle operazioni di caricamento e scaricamento prodotti.

- Mancata modellazione di processi industriali come la comunicazione tra sensori e azionamenti.

Attraverso una programmazione di macchina a stati invece si possono affrontare e risolvere tutti i problemi precedentemente elencati, creando una simulazione realistica del funzionamento macchina. Il principale vantaggio è che i periodi di attesa di ogni equipaggio possono essere implementati automaticamente dalla simulazione tramite comunicazione di sensori e motori, nella generazione della traiettoria completa si possono ignorare gli intervalli di attesa; pertanto, sarà sufficiente mettere in sequenza soltanto le alzate corrispondenti alle movimentazioni calcolate su MATLAB.

5.2. Preparazione del modello – programmazione macchina a stati

IndustrialPhysics permette di gestire la simulazione eseguendo azioni automatiche sul modello attraverso l'uso di script. Questi script sono una lista di istruzioni scritte secondo lo standard IEC 61131 Structured Text, utilizzato per la programmazione di dispositivi industriali come i PLC.

Attraverso l'uso di un *model script* è possibile creare delle variabili o degli input, collegare i parametri I/O dei componenti, eseguire operazioni aritmetiche sulle variabili della simulazione ed altre operazioni.

Per creare uno script si deve selezionare un componente dell'albero modello e si deve utilizzare il comando "*add model script"*. Si apre una finestra (Figura 133) in cui è possibile scrivere la lista di istruzioni. La finestra si divide in due sezioni:

- *Definitions*, qui è possibile creare delle variabili o connettere allo script i parametri I/O dei componenti di un modello;
- *Statements*, qui è possibile scrivere la lista di istruzioni per manipolare le variabili presenti dentro lo script.

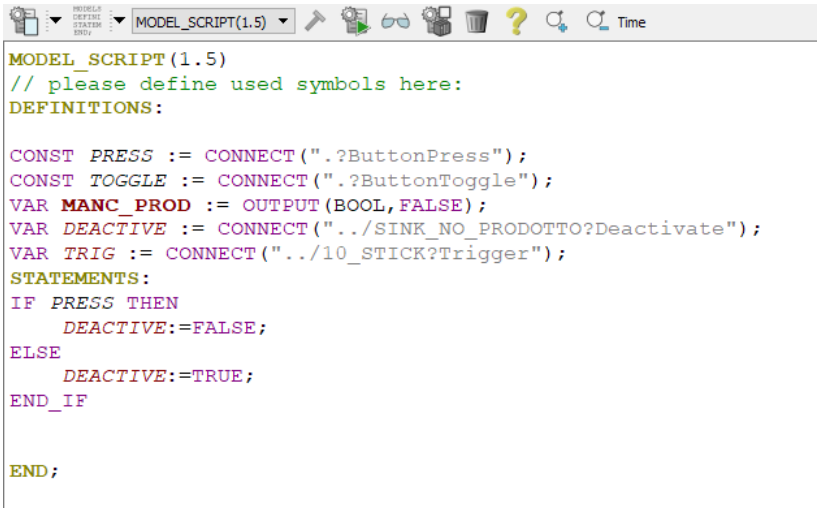


Figura 133 - Struttura di uno script

Connettendo i parametri I/O di un componente ad uno script è possibile leggere in tempo reale il valore dei parametri del componente, oppure si può assegnare un valore alle variabili dei componenti, come ad esempio il segnale di riferimento per il controllo dei motori.

I parametri I/O di un componente sono di due tipologie:

- *Output*, rappresentati da un pallino verde. Gli output sono dei parametri che non è possibile sovrascrivere, ma soltanto leggere. Gli output possono essere collegati ad uno script permettendo la lettura del valore del parametro da parte dello script. Quando un output viene connesso ad uno script, questo viene interpretato come *const* (Figura 134).
- *Input*, rappresentati da un pallino rosso. Gli input sono sovrascrivibili ed è possibile assegnare un valore attraverso l'uso di uno script. Per assegnare un valore è necessario connettere l'output allo script e assegnare il valore negli *statements* (Figura 135).

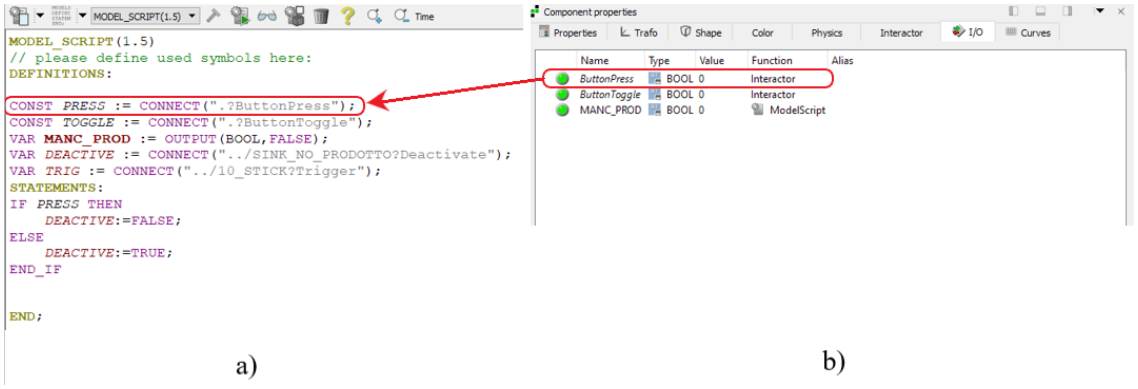


Figura 134 – Connessione dell'output di un componente ad uno script. a) finestra dello script; b) finestra dei parametri I/O di un componente

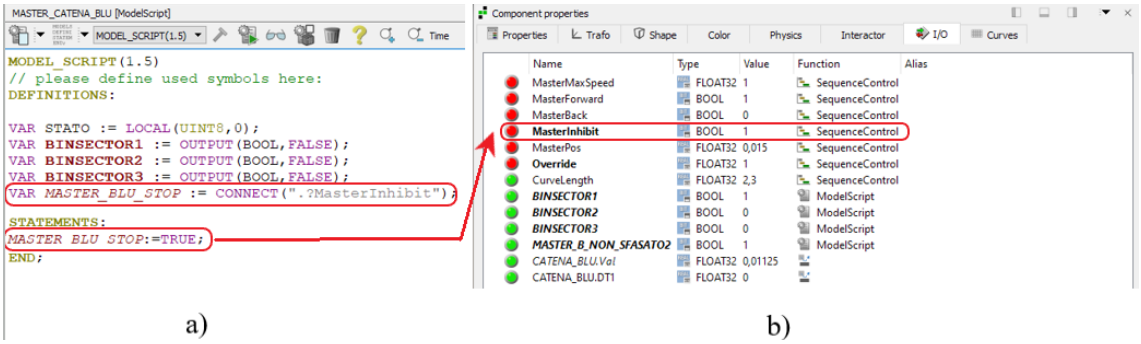


Figura 135 - Connessione dell'input di un componente ad uno script. a) finestra dello script; b) finestra dei parametri I/O di un componente

In uno script, è possibile creare delle variabili. Queste vengono utilizzate per eseguire operazioni aritmetiche, per assegnare un valore all'input di un componente, per scambiare eventi di notifica con altri script ed altro ancora.

Quando si creano delle variabili è necessario assegnare loro la tipologia di variabile, la tipologia di dati (*int*, *float*, *bool*, ecc.) e il loro valore iniziale. La tipologia di variabile indica lo scopo di quest'ultima. Le tipologie usate in questa simulazione sono:

- Variabili *local.* Queste variabili sono utilizzabili solo all'interno dello script in cui sono state create (Figura 136).
- Variabili *output*. Queste variabili sono create all'interno dello script ma possono essere lette dall'esterno, ad esempio da altri script. Quando una

variabile viene definita come *output*, nella scheda delle proprietà componente dello script si popola un parametro I/O di tipo *output* (Figura 137).

MASTER_CATENA_BLU [ModelScript]

MODEL_SCRIPT(1.5) Time

```
MODEL_SCRIPT(1.5)
// please define used symbols here:
DEFINITIONS:

VAR STATO := LOCAL(UINT8,0);
VAR BINSECTOR1 := OUTPUT(BOOL,FALSE);
VAR BINSECTOR2 := OUTPUT(BOOL,FALSE);
VAR BINSECTOR3 := OUTPUT(BOOL,FALSE);
VAR MASTER_BLU_STOP := CONNECT(".?MasterInhibit");
CONST PROD_DEPOSIT := CONNECT("../MASTER_CARIC?PROD_DEPOSIT");
CONST PROD_CONS := CONNECT("../MASTER_SCARIC?PROD_CONS");
CONST COLLISION_BLU := CONNECT("../Instructions?COLLISION_BLU");
VAR COUNT := LOCAL(UINT8,0);
CONST TWAIT := CONNECT("../Instructions?TWAIT");
STATEMENTS:
IF COUNT=TWAIT THEN
    COUNT:=0;
    STATO:=0;
ELSE
    COUNT:=COUNT+1;
END_IF
```

Figura 136 – Variabile script di tipologia local

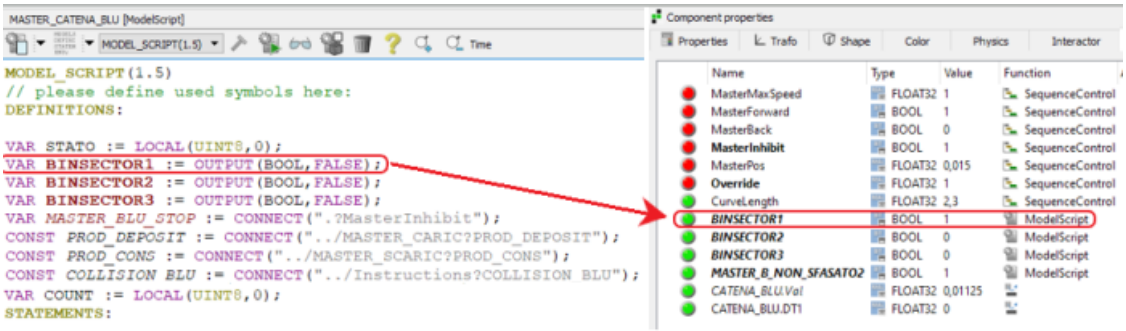


Figura 137 – Variabile script di tipologia output

La simulazione del modello segue una programmazione secondo la teoria degli *automi* e dei *diagrammi parte-intero*. Per generare gli stati macchina si utilizza una funziona nativa dei model script, la funzione *<CASE "nome variabile" OF>*.

Si crea una variabile nello script, chiamata variabile *case*, che può assumere solo valori interi (tipologia di dati: *uint8*). Come esempio, si definisce una variabile "*stato"* che può assumere i valori 0, 1, ecc. La funzione è definita come segue:

```
1.  CASE stato OF
2.      0:
3.        <istruzioni>
4.
5.      1:
6.        <istruzioni>
7.
8.      …
9.
10. END_CASE
```

Al variare del valore di *stato*, la funzione esegue solo la lista di istruzioni che corrispondono al suo valore. In altre parole, se *stato* assume il valore 1, la funzione esegue soltanto le istruzioni corrispondenti allo "stato" 1.

Quando si crea una variabile *case*, oltre alla tipologia di dati si assegna anche il suo valore iniziale, determinando così lo stato iniziale (Figura 138).

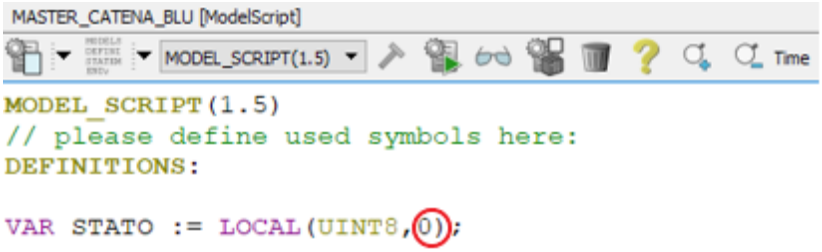


Figura 138 - Assegnamento dello stato inziale

La transizioni da uno stato all'altro vengono determinate dalle istruzioni contenute nello script, che sovrascrivono il valore della variabile *case*. Le transizioni possono essere causate direttamente all'interno dello script, come ad esempio allo scadere di un timer o al raggiungimento di un valore attraverso incrementi di una variabile, e in tal caso vengono definite transizioni *autonome* (Figura 139).

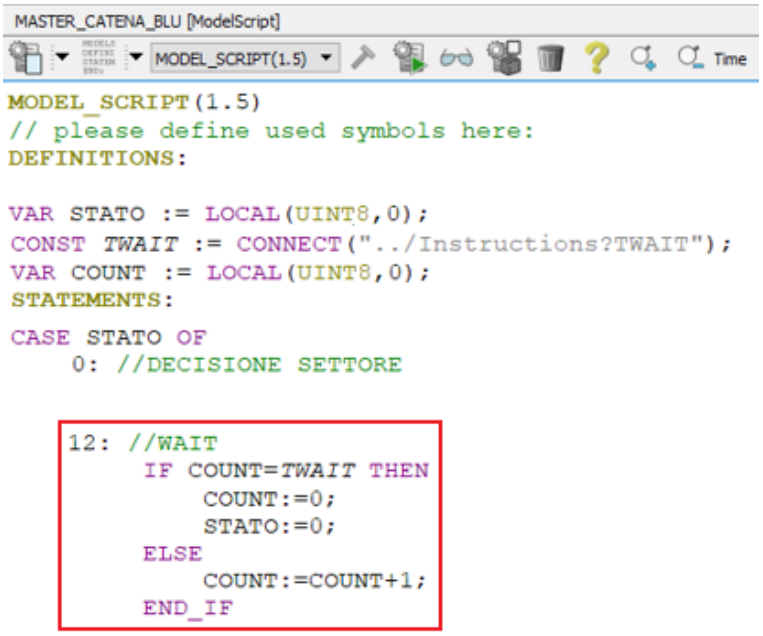


Figura 139 - Esempio di transizione autonoma

Le altre transizioni invece, vengono causate da informazioni scambiate dall'esterno, come ad esempio il cambio di valore di un parametro I/O di un componente, il superamento di un limite di una variabile *output*, ecc. Queste transizioni vengono quindi attivate da fattori esterni e sono dette transizioni *attivabili*. In Figura 140 viene collegato allo script l'output <*Prod_Deposit*> , quando questo parametro diventa *TRUE*, si esegue la lista di istruzioni della condizione *IF*, tra cui è presente il cambio di stato verso lo stato 12.

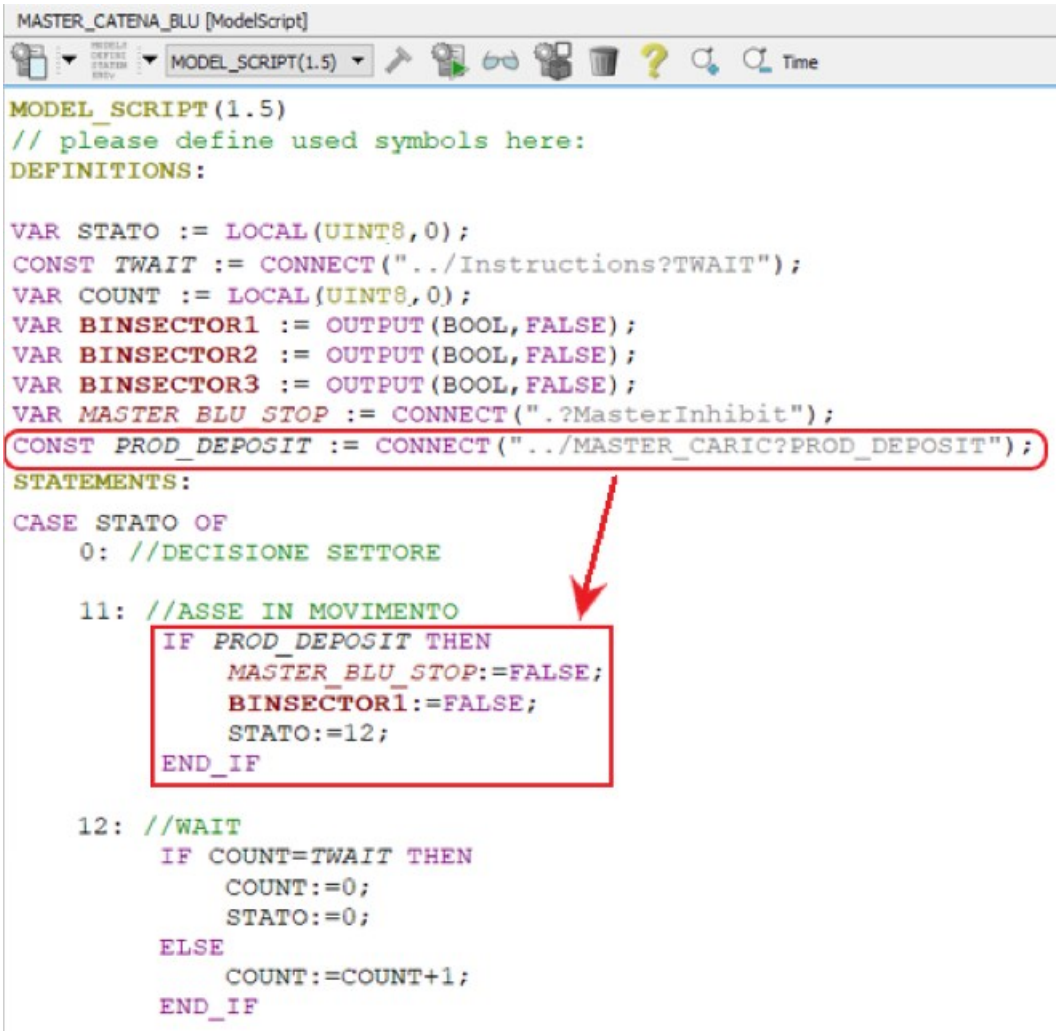


Figura 140 - Esempio di transizione attivabile

La programmazione dell'intero modello è stata realizzata utilizzando cinque *model script* ed ognuno di loro viene rappresentato graficamente come interfaccia di un olone.

Tutti gli oloni sono organizzati all'interno di una oliarchia e ciascun olone scambia informazioni con gli altri permettendo le transizioni di stato.

Nell'oliarchia sono stati esclusi i sensori e gli azionamenti, pertanto essi vengono considerati come elementi esterni. Gli oloni interagiscono con i sensori per leggere la condizione del sistema e agiscono sugli azionamenti per influenzare il sistema.

I sensori utilizzati nel modello forniscono i seguenti *input*:

- *IR1,* è un parametro booleano che proviene da un sensore di presenza prodotto presente nella zona di caricamento. Se il sensore rileva il prodotto allora fornisce in output il valore *TRUE*, altrimenti restituisce *FALSE*;

- *Prelievo*, è un parametro booleano che in stato *TRUE* indica che il robot di caricamento ha prelevato i prodotti dalla zona di caricamento, altrimenti fornisce il valore *FALSE*;
- *Tpos*, è un parametro *FLOAT64* che fornisce la fase dell'asse master del robot di caricamento;
- *Uscita_Prod*, è un parametro booleano che restituisce il valore *TRUE* quando gli spingitori hanno raggiunto il punto morto superiore del loro ciclo;
- *PMI*, è un parametro booleano che restituisce il valore *TRUE* quando gli spingitori hanno raggiunto il punto morto inferiore del loro ciclo;
- *Catena_Blu_Val*, *Catena_Aranc_Val*, sono dei parametri *FLOAT64* che forniscono la posizione degli equipaggi lungo la cinghia, rispettivamente della cinghia blu e della cinghia arancione;
- *EinC*, *EinS*, sono dei parametri booleani che notificano la presenza degli equipaggi rispettivamente nella stazione di caricamento e nella stazione di scaricamento.

Gli azionamenti vengono controllati in posizione dalle leggi di moto. Nel modello vengono caricate soltanto le alzate necessarie per spostarsi da una stazione ad un'altra senza considerare i periodi di attesa. I periodi di attesa vengono gestiti dagli oloni attraverso eventi di notifica che attivano o disattivano il parametro *Master_Inhibit* dei vari motori, cioè gli oloni decidono il periodo di fermo o il periodo di attivazione del motore. Gli eventi di notifica che comandano gli azionamenti sono:

- *MASTER_CARIC_STOP*, abbreviato *MC_Stop*. Se in stato *TRUE* blocca l'asse master del robot di caricamento, se in stato *FALSE* aziona l'asse master del robot di caricamento;
- *MASTER_SCARIC_STOP*, abbreviato *MS_Stop*. Se in stato *TRUE* blocca l'asse master degli spingitori, se in stato *FALSE* aziona l'asse master degli spingitori;
- *MASTER_BLU_STOP*, abbreviato *MB_Stop*. Se in stato *TRUE* blocca l'asse master della cinghia blu, se in stato *FALSE* aziona l'asse master della cinghia blu;

- *MASTER_ARANC_STOP*, abbreviato *MA_Stop*. Se in stato *TRUE* blocca l'asse master della cinghia arancione, se in stato *FALSE* aziona l'asse master della cinghia arancione;

I cinque *model script* che governano il modello sono:

- *MASTER_CARIC*, abbreviato *MC*, che gestisce le operazioni svolte dal robot di caricamento;
- *MASTER_SCARIC*, abbreviato *MS*, che gestisce le operazioni svolte dagli spingitori;
- *MASTER_BLU*, abbreviato *MB*, che gestisce la movimentazione della cinghia blu;
- *MASTER_ARANC*, abbreviato *MA*, che gestisce la movimentazione della cinghia arancione;
- *ISTRUZIONI*, che esegue una serie di istruzioni a supporto per gli altri oloni.

I *model script* utilizzati, e gli oloni che ne derivano, sono simili per ogni configurazione del modello.

Si analizzeranno nel dettaglio gli stati del modello prendendo come riferimento i *model script* della configurazione 8x20. In seguito, verranno elencate le differenze presenti nelle altre configurazioni.

Come primo passo si analizzano solo le interfacce degli oloni associati ai cinque *model script*, quindi considerando in ogni analisi tutti gli altri oloni come elemento esterni.

Successivamente si analizzeranno le implementazioni delle interfacce fino alla costruzione completa dell'oliarchia del sistema.

L'interfaccia dell'olone *MC* è schematizzata in Figura 141.

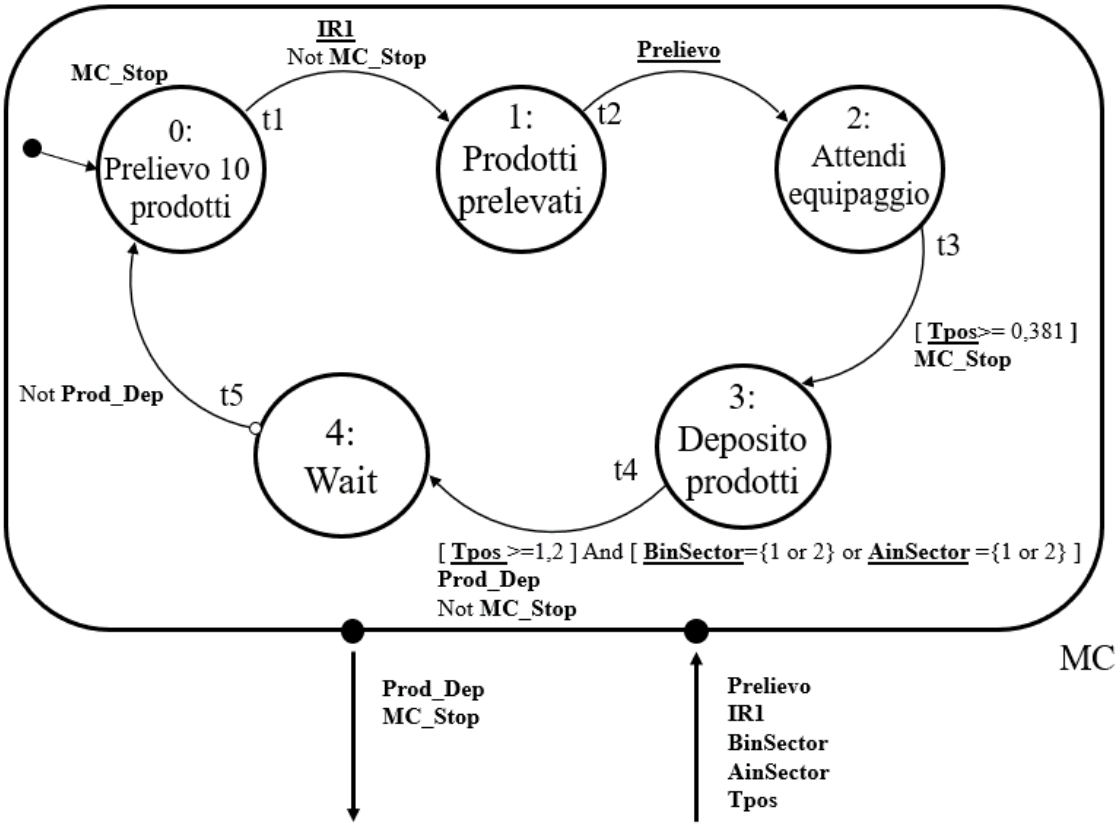


Figura 141 - Interfaccia dell'olone MC

In Figura 142 si può vedere il *model script MASTER_CARIC*.

Lo stato iniziale è lo stato `0: Prelievo 10 prodotti`. Quando il componente *MC* si trova in questo stato genera sempre un output <*MC_Stop=TRUE*>.

- Nello stato 0 il robot di caricamento è bloccato.

La transizione `t1` avviene quando il valore di *IR1* è *TRUE*. Durante la transizione viene generato l'output <*MC_Stop=FALSE*>. Si passa allo stato 1.

- La transizione *t1* avviene quando viene segnalato che nella zona di caricamento sono presenti i prodotti. Il robot di caricamento viene azionato.

Nello stato `1: Prodotti prelevati` non viene eseguita alcuna istruzione.

- Nello stato 1 si attende qualche evento di notifica.

La transizione `t2` avviene quando il valore di *Prelievo* è *TRUE*. Si passa allo stato 2.

- La transizione *t2* avviene quando sono stati prelevati i prodotti.

Nello stato `2: Attendi equipaggio` non viene eseguita alcuna istruzione.

- Nello stato 2 si attende qualche evento di notifica.

La transizione `t3` avviene quando il parametro *Tpos* supera il valore 0,381. Durante la transizione si genera l'output <*MC_Stop=TRUE*>. Si passa allo stato 3.

- Se la fase dell'asse master del robot di caricamento giunge al punto della traiettoria in cui il robot si è posizionato sopra la cinghia, si blocca il robot di caricamento.

Nello stato `3: Deposito prodotti` se si riceve il segnale che *BinSector* o *AinSector* hanno valore 1 o 2, si genera l'output <*MC_Stop=FALSE*>.

- Nello stato 3 si riattiva il robot di caricamento, per far depositare i prodotti, quando un equipaggio Blu o un equipaggio Arancione sono arrivati nella stazione di caricamento.

La transizione `t4` avviene quando il parametro *Tpos* supera il valore 1,2 e si genera l'output <*Prod_Dep*>. Si passa allo stato 4.

- Se il robot di caricamento ha finito il ciclo delle sue operazioni si segnala all'esterno che i prodotti sono stati depositati nell'equipaggio.

Nello stato `4: Wait` si esegue per ogni time_step l'incremento della variabile locale *COUNT*.

- Nello stato 4 si sta aspettando un certo numero di time_step prima di eseguire la transizione.

La transizione `t5` avviene quando la variabile *COUNT* raggiunge il valore 5, dopodiché si resetta la variabile a 0, si cambia a *FALSE* il valore del parametro *Prod_Dep* e si ritorna allo stato 0.

- Dopo un periodo di attesa si ritorna allo stato iniziale, notificando all'esterno che i prodotti non sono più depositati nell'equipaggio.

MASTER_CARIC [ModelScript]

MODEL_SCRIPT(1.5) Time

```
MODEL_SCRIPT(1.5)
// please define used symbols here:
DEFINITIONS:
//SIGNAL: PRODOTTI PRESENTI
CONST IR1 := CONNECT("../World/SOURCE/Sensori/laser_1?IR");
//START/STOP ASSE STAZIONE CARICAMENTO
VAR MASTER_CARIC_STOP := CONNECT(".?MasterInhibit");
//STATO ASSE
VAR STATO := LOCAL(UINT8,0);
CONST TPOS := CONNECT(".?Tempo_ciclo_ingresso.Val");
VAR PROD_DEPOSIT := OUTPUT(BOOL,FALSE);
CONST BINSECTOR1 := CONNECT("../MASTER_CATENA_BLU?BINSECTOR1");
CONST BINSECTOR2 := CONNECT("../MASTER_CATENA_BLU?BINSECTOR2");
CONST GINSECTOR1 := CONNECT("../MASTER_CATENA_GIALLA?GINSECTOR1");
CONST GINSECTOR2 := CONNECT("../MASTER_CATENA_GIALLA?GINSECTOR2");
CONST MASTER_B_NON_SFASATO2 := CONNECT("../MASTER_CATENA_BLU?MASTER_B_NON_SFASATO2");
CONST MASTER_G_NON_SFASATO2 := CONNECT("../MASTER_CATENA_GIALLA?MASTER_G_NON_SFASATO2");
CONST PRELIEVO := CONNECT("../Robot/Sensore_di_ciclo?Collision");
VAR PROD_PRELEVATI := LOCAL(BOOL,FALSE);
VAR COUNT:=LOCAL(UINT8,0);
STATEMENTS:

CASE STATO OF
    0: //PRELIEVO 10 PRODOTTI
        IF IR1 THEN
            MASTER_CARIC_STOP:=FALSE;
            STATO:=1;
        ELSE
            MASTER_CARIC_STOP:=TRUE;
        END_IF

    1: //PRODOTTI PRELEVATI
        IF PRELIEVO THEN
            STATO:=2;
        END_IF


    2: //ATTENDI L'EQUIPAGGIO IN STAZIONE
        IF TPOS>=0.381 THEN
            MASTER_CARIC_STOP:=TRUE;
            STATO:=3;
        END_IF

    3: //DEPOSITO PRODOTTI

        IF BINSECTOR1 OR BINSECTOR2 OR GINSECTOR1 OR GINSECTOR2 THEN
            MASTER_CARIC_STOP:=FALSE;
            IF TPOS<=0 OR TPOS>=1.19 THEN
                PROD_DEPOSIT:=TRUE;
                STATO:=4;
            ELSE
                PROD_DEPOSIT:=FALSE;
            END_IF
        END_IF

    4: //WAIT
        IF COUNT=5 THEN
           COUNT:=0;
           PROD_DEPOSIT:=FALSE;
           STATO:=0;
        ELSE
           COUNT:=COUNT+1;
        END_IF
```

Line 26, Col. 38

Local console

Figura 142 – Model script di MASTER_CARIC

L'interfaccia dell'olone *MS* è schematizzata in Figura 143.

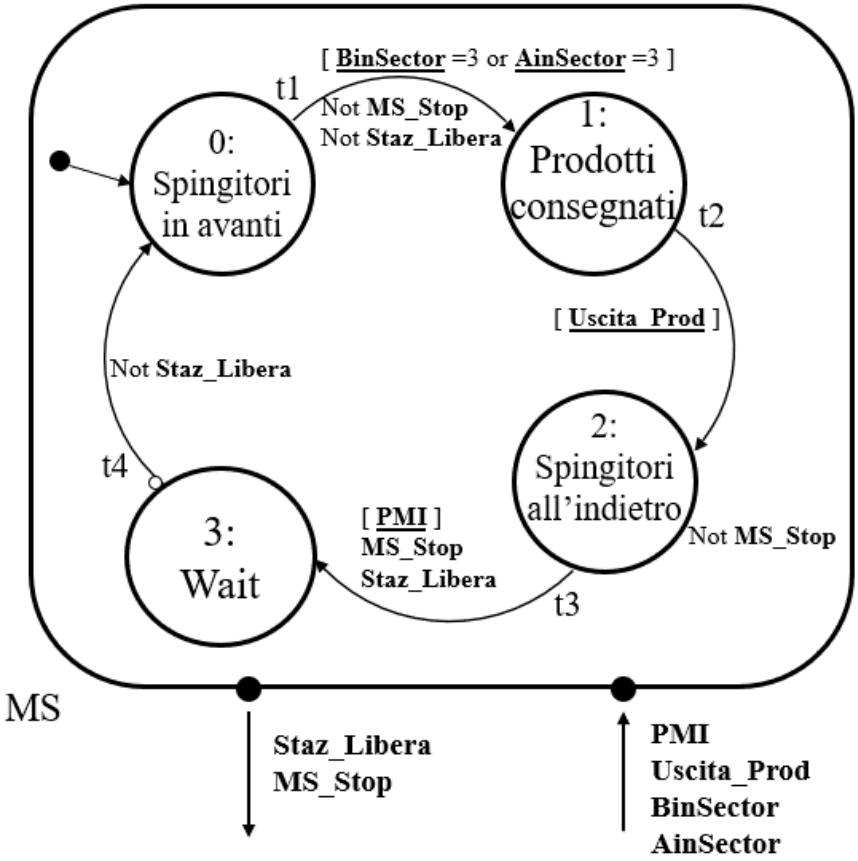


Figura 143 - Interfaccia dell'olone MS

In Figura 144 si può vedere il *model script MASTER_SCARIC*.

Lo stato iniziale è lo stato `0: Spingitori in avanti`. Quando il componente *MS* si trova in questo stato non esegue alcuna istruzione.

- Nello stato 0 gli spingitori attendono un evento di notifica.

La transizione `t1` avviene quando il segnale *BinSector* o *AinSector* assumono valore pari a 3. Durante la transizione viene generato l'output <*MS_Stop=FALSE*> e si emette un evento di notifica <*Staz_Libera=FALSE*>. Si passa allo stato 1.

- La transizione *t1* avviene quando viene segnalato che nella zona di scaricamento è presente un equipaggio blu o arancione. Gli spingitori vengono azionati e si notifica all'esterno che la stazione è occupata.

Nello stato `1: Prodotti consegnati` non viene eseguita alcuna istruzione.

- Nello stato 1 si attende qualche evento di notifica.

La transizione `t2` avviene quando il segnale di *Uscita_Prod* è *TRUE*. Si passa allo stato 2.

- La transizione *t2* avviene quando i prodotti sono stati consegnati.

Nello stato `2: Spingitori all'indietro` viene generato l'output <*MS_Stop=FALSE*>.

- Nello stato 2 si mantiene attivo l'asse master degli spingitori.

La transizione `t3` avviene quando si riceve il segnale *PMI* con valore *TRUE*. Durante la transizione si genera l'output <*MS_Stop=TRUE*> e contemporaneamente si notifica all'esterno l'evento <*Staz_Libera=TRUE*>. Si passa allo stato 3.

- Se gli spingitori hanno raggiunto il punto morto inferiore del loro ciclo allora la stazione risulta libera e viene bloccato l'asse master degli spingitori.

Nello stato `3: Wait` si esegue per ogni time_step l'incremento della variabile locale *COUNT*.

- Nello stato 3 si sta aspettando un certo numero di time_step prima di eseguire la transizione.

La transizione `t4` avviene quando la variabile *COUNT* raggiunge il valore 4, dopodiché si resetta la variabile a 0, si cambia a *FALSE* il valore del segnale *Staz_Libera* e si ritorna allo stato 0.

- Dopo un periodo di attesa si ritorna allo stato iniziale, notificando all'esterno che la stazione non è più libera.

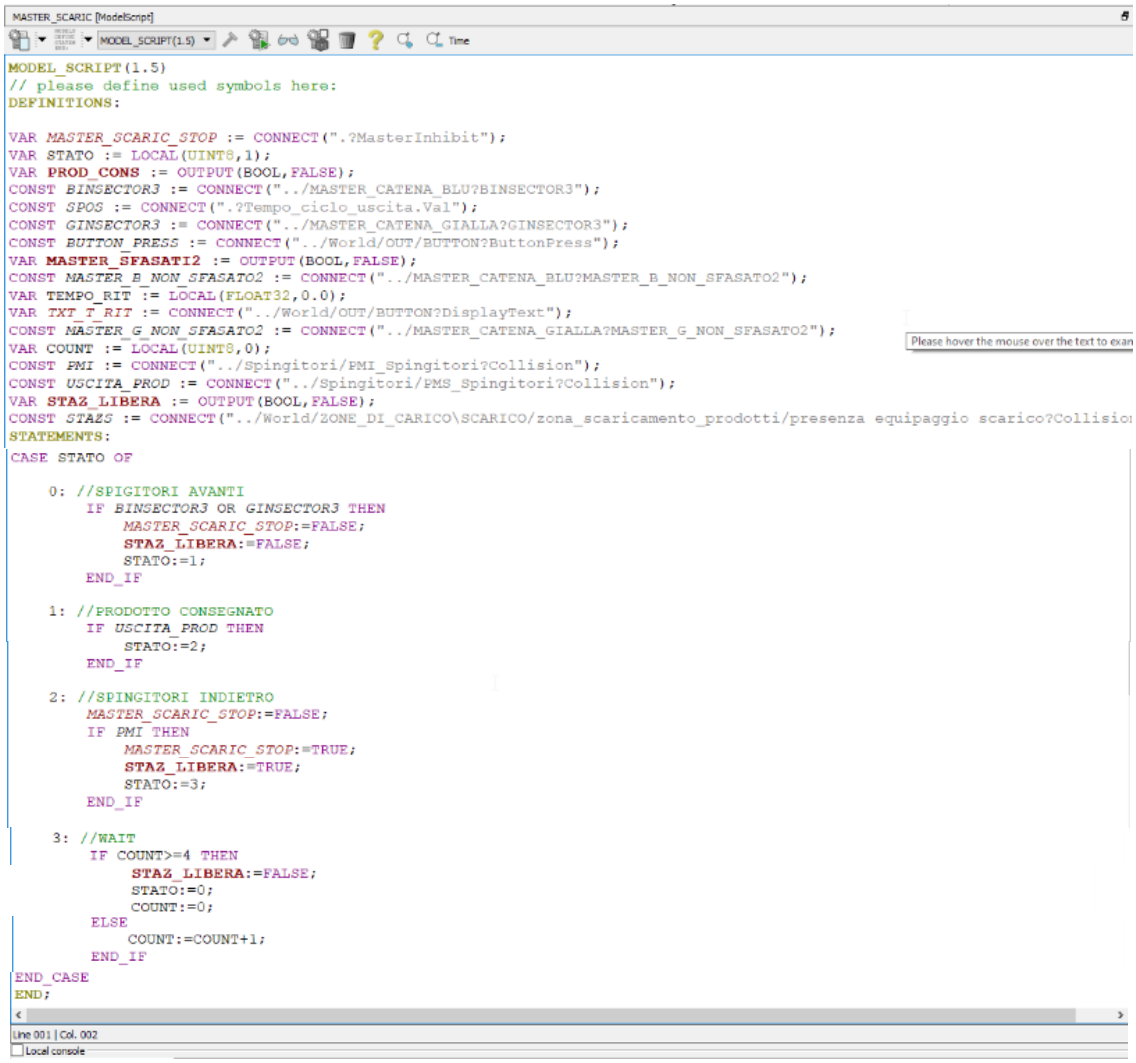


Figura 144 - Model script di MASTER_SCARIC

Il *model script ISTRUZIONI* viene utilizzato solo come supporto per gli altri *model* script, non viene quindi strutturato con una programmazione a stati ma esegue ad ogni *time step* una serie di istruzioni che coordinano e gestiscono la simulazione del modello (Figura 145 e Figura 146).

Instructions [ModelScript]

MODEL_SCRIPT(1.5) Time

```
MODEL_SCRIPT(1.5)
// please define used symbols here:
DEFINITIONS:
//SENSORI STAZIONE DI CARICAMENTO
CONST IR10 := CONNECT("../World/SOURCE/Sensori/laser_10?IR");

//VARIABILI PER CONTARE IL TEMPO CICLO
VAR COUNT := LOCAL(UINT8,0);
VAR COUNT2 := LOCAL(UINT8,0);
CONST CICLO_ROBOT := CONNECT("../Robot/Sensore_di_ciclo?CollisionRisingEdge");
VAR WARMUP := OUTPUT(BOOL,FALSE);
VAR TEMPO_IN := LOCAL(FLOAT64,0);
VAR TEMPO_CICLO_IN := OUTPUT(FLOAT64,0);
VAR TEMPO_OUT := LOCAL(FLOAT64,0.0);
VAR TEMPO_CICLO_OUT := OUTPUT(FLOAT64,0);
VAR TEMPO_CICLO_ASSI := LOCAL(FLOAT64,4.8);
CONST CICLO_SPINGITORI := CONNECT("../Spingitori/PMS_Spingitori?CollisionRisingEdge");

//PER SCRITTURA TESTO
VAR TXT_T_OUT_1 := CONNECT("../TXT_TEMPO_CICLO_USCITA_1?DisplayText");
VAR TXT_T_OUT_OFFSET_1 := CONNECT("../TXT_TEMPO_CICLO_USCITA_1?TextOffset");
VAR TXT_T_IN := CONNECT("../TXT_TEMPO_CICLO_INGRESSO?DisplayText");
VAR TXT_T_IN_OFFSET := CONNECT("../TXT_TEMPO_CICLO_INGRESSO?TextOffset");
VAR TXT_T_OUT_2 := CONNECT("../TXT_TEMPO_CICLO_USCITA2?DisplayText");
VAR TXT_T_OUT_OFFSET_2 := CONNECT("../TXT_TEMPO_CICLO_USCITA2?TextOffset");
VAR PRODUTTIVITA := OUTPUT(FLOAT32,0.0);

//COLLISION DETECTION
VAR COLLISION_BLU:=OUTPUT(BOOL,FALSE);
VAR COLLISION_ ARANCIONE :=OUTPUT(BOOL,FALSE);
CONST CATENA_ ARANCIONE_VAL := CONNECT("../MASTER_CATENA_GIALLA?CATENA_GIALLA.ChainDrvActual.Val");
CONST CATENA_BLU_VAL := CONNECT("../MASTER_CATENA_BLU?CATENA_BLU.Val");
//DEFINIZIONE OPERAZIONI CATENA
VAR OPZ1B:=OUTPUT(BOOL,FALSE);
VAR OPZ2B:=OUTPUT(BOOL,FALSE);
VAR OPZ3B:=OUTPUT(BOOL,FALSE);
VAR OPZ1G:=OUTPUT(BOOL,FALSE);
VAR OPZ2G:=OUTPUT(BOOL,FALSE);
VAR OPZ3G:=OUTPUT(BOOL,FALSE);
CONST CHAINPOSB := CONNECT("../CATENA_BLU?ChainPos");
CONST CHAINPOSG := CONNECT("../CATENA_GIALLA?ChainPos");
CONST C20E4B := CONNECT("../CATENA_BLU/Equipaggio_blu_4/Cassetto_blu_20?Collision");
CONST C10E4B := CONNECT("../CATENA_BLU/Equipaggio_blu_4/Cassetto_blu_10?Collision");
CONST C20E3B := CONNECT("../CATENA_BLU/Equipaggio_blu_3/Cassetto_blu_20?Collision");
CONST C10E3B := CONNECT("../CATENA_BLU/Equipaggio_blu_3/Cassetto_blu_10?Collision");
CONST C20E2B := CONNECT("../CATENA_BLU/Equipaggio_blu_2/Cassetto_blu_20?Collision");
CONST C10E2B := CONNECT("../CATENA_BLU/Equipaggio_blu_2/Cassetto_blu_10?Collision");
CONST C20E1B := CONNECT("../CATENA_BLU/Equipaggio_blu_1/Cassetto_blu_20?Collision");
CONST C10E1B := CONNECT("../CATENA_BLU/Equipaggio_blu_1/Cassetto_blu_10?Collision");
CONST C20E4G := CONNECT("../CATENA_GIALLA/Equipaggio_giallo_4/Cassetto_giallo_20?Collision");
CONST C10E4G := CONNECT("../CATENA_GIALLA/Equipaggio_giallo_4/Cassetto_giallo_10?Collision");
CONST C20E3G := CONNECT("../CATENA_GIALLA/Equipaggio_giallo_3/Cassetto_giallo_20?Collision");
CONST C10E3G := CONNECT("../CATENA_GIALLA/Equipaggio_giallo_3/Cassetto_giallo_10?Collision");
CONST C20E2G := CONNECT("../CATENA_GIALLA/Equipaggio_giallo_2/Cassetto_giallo_20?Collision");
CONST C10E2G := CONNECT("../CATENA_GIALLA/Equipaggio_giallo_2/Cassetto_giallo_10?Collision");
CONST C20E1G := CONNECT("../CATENA_GIALLA/Equipaggio_giallo_1/Cassetto_giallo_20?Collision");
CONST C10E1G := CONNECT("../CATENA_GIALLA/Equipaggio_giallo_1/Cassetto_giallo_10?Collision");
CONST STAZS := CONNECT("../World/ZONE_DI_CARICO\SCARICO/zona_scaricamento_prodotti/presenza equipaggio scarico?Co
CONST STAZCB := CONNECT("../World/ZONE_DI_CARICO\SCARICO/zona_caricamento_prodotti/presenza equipaggio carico BLU
CONST STAZCG := CONNECT("../World/ZONE_DI_CARICO\SCARICO/zona_caricamento_prodotti/presenza equipaggio carico GIA

//DEFINIZIONE TEMPO DI ATTESA OPERAZIONI CATENA
VAR TWAIT := OUTPUT(UINT8,35);
```

Figura 145 – Settore -Definitions- del model script ISTRUZIONI

```
STATEMENTS:
// OPERAZIONI CATENA BLU
IF STAZCB AND C20E1B THEN
    OPZ1B:=TRUE;
ELSIF STAZCB AND C10E1B THEN
    OPZ2B:=TRUE;
ELSIF STAZCB AND C20E2B THEN
    OPZ1B:=TRUE;
ELSIF STAZCB AND C10E2B THEN
    OPZ2B:=TRUE;
ELSIF STAZCB AND C20E3B THEN
    OPZ1B:=TRUE;
ELSIF STAZCB AND C10E3B THEN
    OPZ2B:=TRUE;
ELSIF STAZCB AND C20E4B THEN
    OPZ1B:=TRUE;
ELSIF STAZCB AND C10E4B THEN
    OPZ2B:=TRUE;
ELSE
    OPZ1B:=FALSE;
    OPZ2B:=FALSE;
END_IF

IF STAZS AND C20E1B  THEN
    OPZ3B:=TRUE;
ELSIF STAZS AND C20E2B THEN
    OPZ3B:=TRUE;
ELSIF STAZS AND C20E3B THEN
    OPZ3B:=TRUE;
ELSIF STAZS AND C20E4B THEN
    OPZ3B:=TRUE;
ELSE
    OPZ3B:=FALSE;
END_IF

// OPERAZIONI CATENA ARANCIONE

IF STAZCG AND C20E1G THEN
    OPZ1G:=TRUE;
ELSIF STAZCG AND C10E1G THEN
    OPZ2G:=TRUE;
ELSIF STAZCG AND C20E2G THEN
    OPZ1G:=TRUE;
ELSIF STAZCG AND C10E2G THEN
    OPZ2G:=TRUE;
ELSIF STAZCG AND C20E3G THEN
    OPZ1G:=TRUE;
ELSIF STAZCG AND C10E3G THEN
    OPZ2G:=TRUE;
ELSIF STAZCG AND C20E4G THEN
    OPZ1G:=TRUE;
ELSIF STAZCG AND C10E4G THEN
    OPZ2G:=TRUE;
ELSE
    OPZ1G:=FALSE;
    OPZ2G:=FALSE;
END_IF

IF STAZS AND C20E1G THEN
    OPZ3G:=TRUE;
ELSIF STAZS AND C20E2G THEN
    OPZ3G:=TRUE;
ELSIF STAZS AND C20E3G THEN
    OPZ3G:=TRUE;
ELSIF STAZS AND C20E4G THEN
    OPZ3G:=TRUE;
ELSE
    OPZ3G:=FALSE;
END_IF


//COLLISION DETECTION
IF 0.62+CATENA_ARANCIONE_VAL-CATENA_BLU_VAL<=0.03 THEN
    COLLISION_BLU:=TRUE;
ELSE
    COLLISION_BLU:=FALSE;
END_IF
IF 0.685+CATENA_BLU_VAL-CATENA_ARANCIONE_VAL <=0.03 THEN
    COLLISION_ARANCIONE:=TRUE;
ELSE
    COLLISION_ARANCIONE:=FALSE;
END_IF

END;
```

Line 001 | Col. 002

Local console

Figura 146 - Settore -Statements- del model script ISTRUZIONI

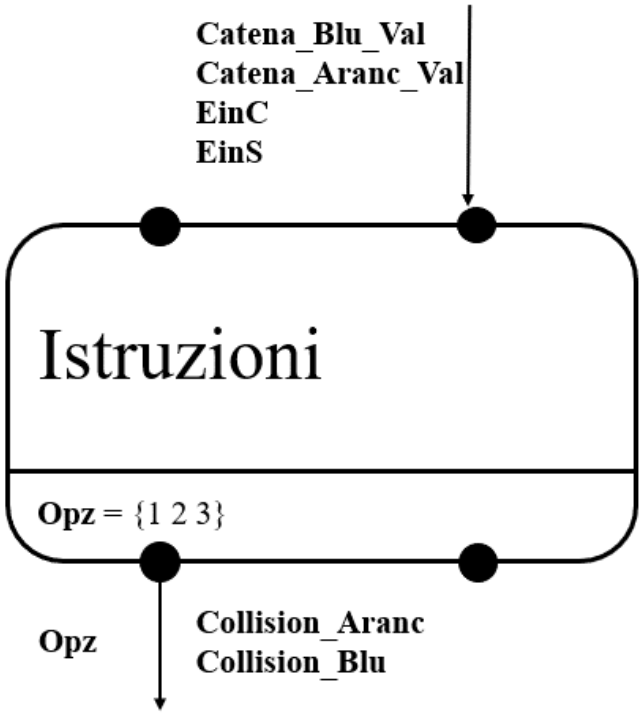


Figura 147 - Interfaccia dell'olone ISTRUZIONI

Dalla Figura 147 si possono vedere gli input e gli output del *model script*.

Guardando alla Figura 146, la parte di codice <//*OPERAZIONI CATENA BLU*> e la parte <//*OPERAZIONI CATENA ARANCIONE*> generano l'output <*Opz = {1 2 3}*> elaborando il segnali di input *EinC* e il segnale di input *EinS* che indicano la presenza di un equipaggio rispettivamente nella stazione di caricamento e la stazione di scaricamento.

La parte di codice <//*COLLISION DETECTION*> prende in input la posizione degli equipaggi lungo la cinghia (*Catena_Blu_Val* e *Catena_Aranc_Val*) e restituisce in output le variabili <*Collision_Blu*> e <*Collision_Aranc*>. Questi output segnalano agli elementi esterni che un equipaggio (appartenente alla cinghia blu o arancione) sta per collidere con un altro equipaggio.

L'interfaccia dell'olone *MB* è schematizzata in Figura 148.

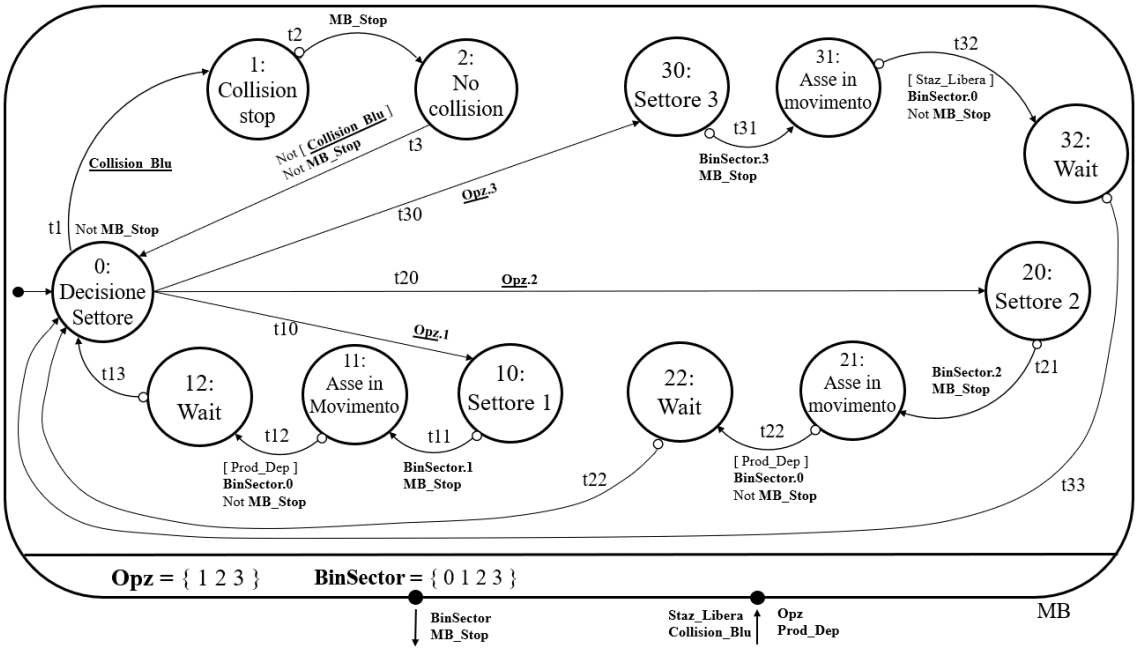


Figura 148 - Interfaccia dell'olone MB

In Figura 149 si può vedere il *model script MASTER_BLU*.

Lo stato iniziale è lo stato `0: Decisione Settore`. Quando il componente *MB* si trova in questo stato genera l'output <*MB_Stop=FALSE*>.

- Nello stato 0 l'asse master delle cinghia blu è attivato.

La transizione `t1` avviene quando il segnale *Collision_Blu* diventa *TRUE*. Questa transizione avviene tramite un'istruzione non subordinata alla funzione <*CASE … OF*>, in questo modo, la transizione di stato può essere attivata indipendentemente da quale stato si trova il sistema. Si passa allo stato 1.

- La transizione *t1* avviene quando viene segnalato che l'equipaggio blu sta per collidere con un altro equipaggio. Il segnale proveniente dal *collision detection* viene controllato ad ogni *time_step* della simulazione.

Nello stato `1: Collision stop` viene generato l'output <*MB_Stop=TRUE*>.

- Nello stato 1 si blocca l'asse master della cinghia blu.

La transizione `t2` avviene immediatamente dopo l'entrata nello stato 1. Si passa allo stato 2.

- La transizione è immediata subito dopo aver bloccato la cinghia di movimentazione.

Nello stato `2: No Collision` non viene eseguita alcuna istruzione.

- Nello stato 2 si attende qualche evento di notifica.

La transizione `t3` avviene quando il segnale di *Collision_Blu* è *FALSE*. Si ritorna allo stato iniziale 0.

- La transizione *t3* avviene quando il *collision detection* rileva che non c'è più pericolo di collisione tra gli equipaggi.

La transizione `t10` avviene quando si riceve il segnale *Opz* con valore uguale ad 1.

- La transizione *t10* avviene quando viene notificato dall'esterno che l'equipaggio si trova nelle condizioni di eseguire la prima operazione di caricamento.

Nello stato `10: Settore 1` si notifica all'esterno l'evento <*BinSector=1*> e si genera l'output <*MB_Stop*>.

- Nello stato 10 si blocca l'asse master della cinghia blu e si comunica all'esterno che sull'equipaggio blu può essere effettuata la prima operazione di caricamento.

La transizione `t11` avviene immediatamente dopo l'entrata nello stato 10. Si passa allo stato 11.

- La transizione è immediata subito dopo aver bloccato la cinghia di movimentazione.

Nello stato `11: Asse in movimento` non viene eseguita alcuna istruzione.

- Nello stato 11 si attende qualche evento di notifica.

La transizione `t12` avviene quando il segnale di *Prod_Dep* è *TRUE*. Si genera l'output <*MB_Stop=FALSE*> e si notifica all'esterno <*BinSector=0*>. Si passa allo stato 12.

- La transizione *t12* avviene quando è notificato che sono stati depositati i prodotti sull'equipaggio. Si riattiva l'asse master della cinghia blu e si notifica che l'equipaggio non è più disponibile per operazioni di carico/scarico.

Nello stato `12: Wait` si esegue per ogni time_step l'incremento della variabile locale *COUNT*.

- Nello stato 12 si sta aspettando un certo numero di time_step prima di eseguire la transizione.

La transizione `t13` avviene quando la variabile *COUNT* raggiunge il valore 5, dopodiché si resetta la variabile a 0 e si ritorna allo stato iniziale 0.

- Dopo un periodo di attesa si ritorna allo stato iniziale.

La transizione `t20` avviene quando si riceve il segnale *Opz* con valore uguale ad 2.

- La transizione *t20* avviene quando viene notificato dall'esterno che l'equipaggio si trova nelle condizioni di eseguire la seconda operazione di caricamento.

Nello stato `20: Settore 2` si notifica all'esterno l'evento <*BinSector=2*> e si genera l'output <*MB_Stop*>.

- Nello stato 20 si blocca l'asse master della cinghia blu e si comunica all'esterno che sull'equipaggio blu può essere effettuata la seconda operazione di caricamento.

La transizione `t21` avviene immediatamente dopo l'entrata nello stato 20. Si passa allo stato 21.

- La transizione è immediata subito dopo aver bloccato la cinghia di movimentazione.

Nello stato `21: Asse in movimento` non viene eseguita alcuna istruzione.

- Nello stato 21 si attende qualche evento di notifica.

La transizione `t22` avviene quando il segnale di *Prod_Dep* è *TRUE*. Si genera l'output <*MB_Stop=FALSE*> e si notifica all'esterno <*BinSector=0*>. Si passa allo stato 22.

- La transizione *t22* avviene quando è notificato che sono stati depositati i prodotti sull'equipaggio. Si riattiva l'asse master della cinghia blu e si notifica che l'equipaggio non è più disponibile per operazioni di carico/scarico.

Nello stato `22: Wait` si esegue per ogni time_step l'incremento della variabile locale *COUNT*.

- Nello stato 22 si sta aspettando un certo numero di time_step prima di eseguire la transizione.

La transizione `t23` avviene quando la variabile *COUNT* raggiunge il valore 5, dopodiché si resetta la variabile a 0 e si ritorna allo stato iniziale 0.

- Dopo un periodo di attesa si ritorna allo stato iniziale.

La transizione `t30` avviene quando si riceve il segnale *Opz* con valore uguale ad 3.

- La transizione *t30* avviene quando viene notificato dall'esterno che l'equipaggio si trova nelle condizioni di eseguire la prima operazione di scaricamento.

Nello stato `30: Settore 3` si notifica all'esterno l'evento <*BinSector=3*> e si genera l'output <*MB_Stop*>.

- Nello stato 30 si blocca l'asse master della cinghia blu e si comunica all'esterno che sull'equipaggio blu può essere effettuata la prima operazione di scaricamento.

La transizione `t31` avviene immediatamente dopo l'entrata nello stato 30. Si passa allo stato 31.

- La transizione è immediata subito dopo aver bloccato la cinghia di movimentazione.

Nello stato `31: Asse in movimento` non viene eseguita alcuna istruzione.

- Nello stato 31 si attende qualche evento di notifica.

La transizione `t32` avviene quando il segnale di *Staz_Libera* è *TRUE*. Si genera l'output <*MB_Stop=FALSE*> e si notifica all'esterno <*BinSector=0*>. Si passa allo stato 32.

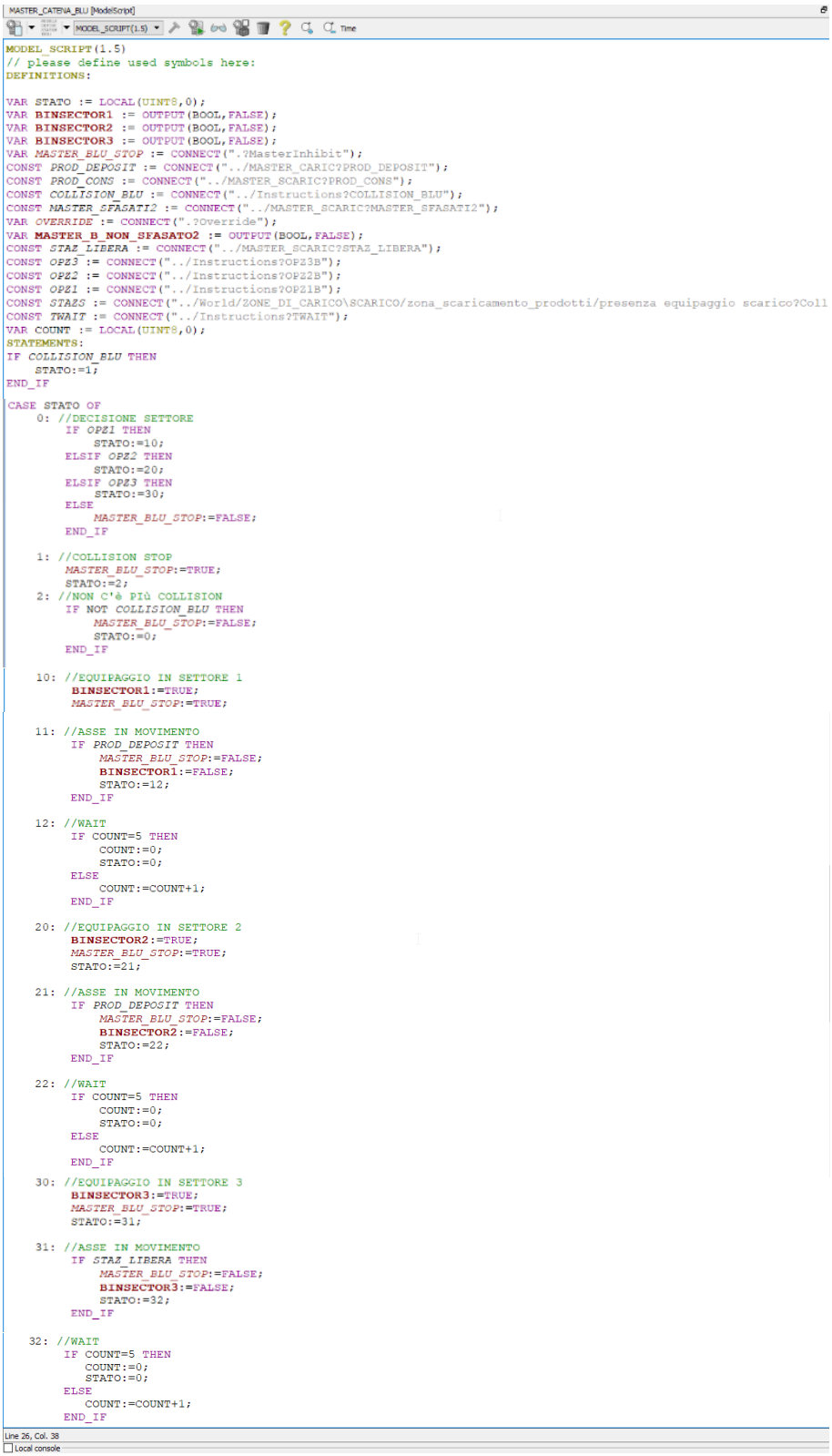
- La transizione *t32* avviene quando è notificato che sono stati consegnati i prodotti dall'equipaggio. Si riattiva l'asse master della cinghia blu e si notifica che l'equipaggio non è più disponibile per operazioni di carico/scarico.

Nello stato `32: Wait` si esegue per ogni time_step l'incremento della variabile locale *COUNT*.

- Nello stato 32 si sta aspettando un certo numero di time_step prima di eseguire la transizione.

La transizione `t33` avviene quando la variabile *COUNT* raggiunge il valore 5, dopodiché si resetta la variabile a 0 e si ritorna allo stato iniziale 0.

- Dopo un periodo di attesa si ritorna allo stato iniziale.



```
MASTER_CATENA_BLU [ModelScript]
MODEL_SCRIPT(1.5)
// please define used symbols here:
DEFINITIONS:

VAR STATO := LOCAL(UINT8,0);
VAR BINSECTOR1 := OUTPUT(BOOL,FALSE);
VAR BINSECTOR2 := OUTPUT(BOOL,FALSE);
VAR BINSECTOR3 := OUTPUT(BOOL,FALSE);
VAR MASTER_BLU_STOP := CONNECT(".?MasterInhibit");
CONST PROD_DEPOSIT := CONNECT("../MASTER_CARIC?PROD_DEPOSIT");
CONST PROD_CONS := CONNECT("../MASTER_SCARIC?PROD_CONS");
CONST COLLISION_BLU := CONNECT("../Instructions?COLLISION_BLU");
CONST MASTER_SFASATI2 := CONNECT("../MASTER_SCARIC?MASTER_SFASATI2");
VAR OVERRIDE := CONNECT(".?Override");
VAR MASTER_B_NON_SFASATO2 := OUTPUT(BOOL,FALSE);
CONST STAZ_LIBERA := CONNECT("../MASTER_SCARIC?STAZ_LIBERA");
CONST OPZ3 := CONNECT("../Instructions?OPZ3B");
CONST OPZ2 := CONNECT("../Instructions?OPZ2B");
CONST OPZ1 := CONNECT("../Instructions?OPZ1B");
CONST STAZS := CONNECT("../World/ZONE_DI_CARICO\SCARICO/zona_scaricamento_prodotti/presenza equipaggio scarico?Coll
CONST TWAIT := CONNECT("../Instructions?TWAIT");
VAR COUNT := LOCAL(UINT8,0);
STATEMENTS:
IF COLLISION_BLU THEN
    STATO:=1;
END_IF

CASE STATO OF
    0: //DECISIONE SETTORE
        IF OPZ1 THEN
            STATO:=10;
        ELSIF OPZ2 THEN
            STATO:=20;
        ELSIF OPZ3 THEN
            STATO:=30;
        ELSE
            MASTER_BLU_STOP:=FALSE;
        END_IF

    1: //COLLISION STOP
        MASTER_BLU_STOP:=TRUE;
        STATO:=2;
    2: //NON C'è PIù COLLISION
        IF NOT COLLISION_BLU THEN
            MASTER_BLU_STOP:=FALSE;
            STATO:=0;
        END_IF

    10: //EQUIPAGGIO IN SETTORE 1
         BINSECTOR1:=TRUE;
         MASTER_BLU_STOP:=TRUE;

    11: //ASSE IN MOVIMENTO
         IF PROD_DEPOSIT THEN
             MASTER_BLU_STOP:=FALSE;
             BINSECTOR1:=FALSE;
             STATO:=12;
         END_IF

    12: //WAIT
         IF COUNT=5 THEN
             COUNT:=0;
             STATO:=0;
         ELSE
             COUNT:=COUNT+1;
         END_IF

    20: //EQUIPAGGIO IN SETTORE 2
         BINSECTOR2:=TRUE;
         MASTER_BLU_STOP:=TRUE;
         STATO:=21;

    21: //ASSE IN MOVIMENTO
         IF PROD_DEPOSIT THEN
             MASTER_BLU_STOP:=FALSE;
             BINSECTOR2:=FALSE;
             STATO:=22;
         END_IF

    22: //WAIT
         IF COUNT=5 THEN
             COUNT:=0;
             STATO:=0;
         ELSE
             COUNT:=COUNT+1;
         END_IF
    30: //EQUIPAGGIO IN SETTORE 3
         BINSECTOR3:=TRUE;
         MASTER_BLU_STOP:=TRUE;
         STATO:=31;

    31: //ASSE IN MOVIMENTO
         IF STAZ_LIBERA THEN
             MASTER_BLU_STOP:=FALSE;
             BINSECTOR3:=FALSE;
             STATO:=32;
         END_IF

   32: //WAIT
        IF COUNT=5 THEN
           COUNT:=0;
           STATO:=0;
        ELSE
           COUNT:=COUNT+1;
        END_IF
Line 26, Col. 38
Local console
```

Figura 149 - Model Script di MASTER_BLU

Il *model script MASTER_ARANC*, che gestisce la cinghia arancione, è analogo al *model script MASTER_BLU*. Le uniche differenze sono il nome delle variabili che fanno riferimento alla cinghia arancione piuttosto che a quella blu:

- L'output di azionamento dell'asse master della cinghia arancione <*MA_Stop*> al posto di <*MB_Stop*>;
- La notifica <*AinSector = {0 1 2 3}*> al posto di <*BinSector = {0 1 2 3}*> .

L'interfaccia dell'olone *MA* è riportata in Figura 150.

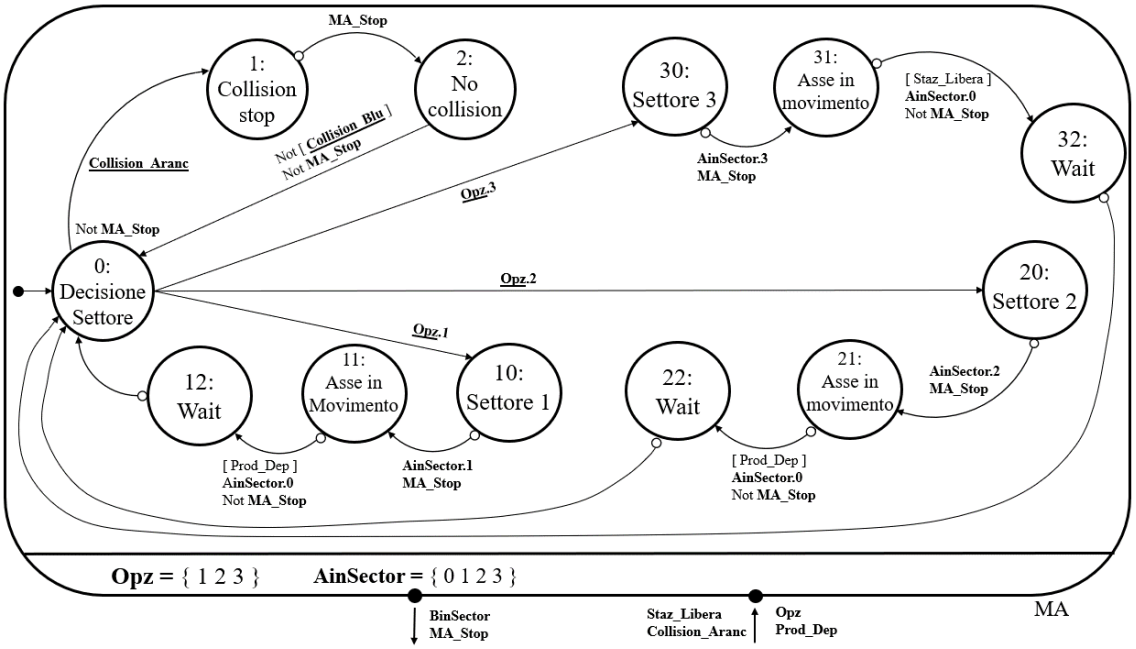


Figura 150 - Interfaccia dell'olone MA

Una volta analizzate le interfacce dei singoli oloni, passiamo all'analisi dell'implementazione di queste interfacce (Figura 151, Figura 152 e Figura 153).

In questa analisi, gli eventi di notifica interni che gli oloni si scambiano tra di loro vengono racchiusi nelle parantesi angolari per differenziarli dagli output scambiati verso l'esterno.

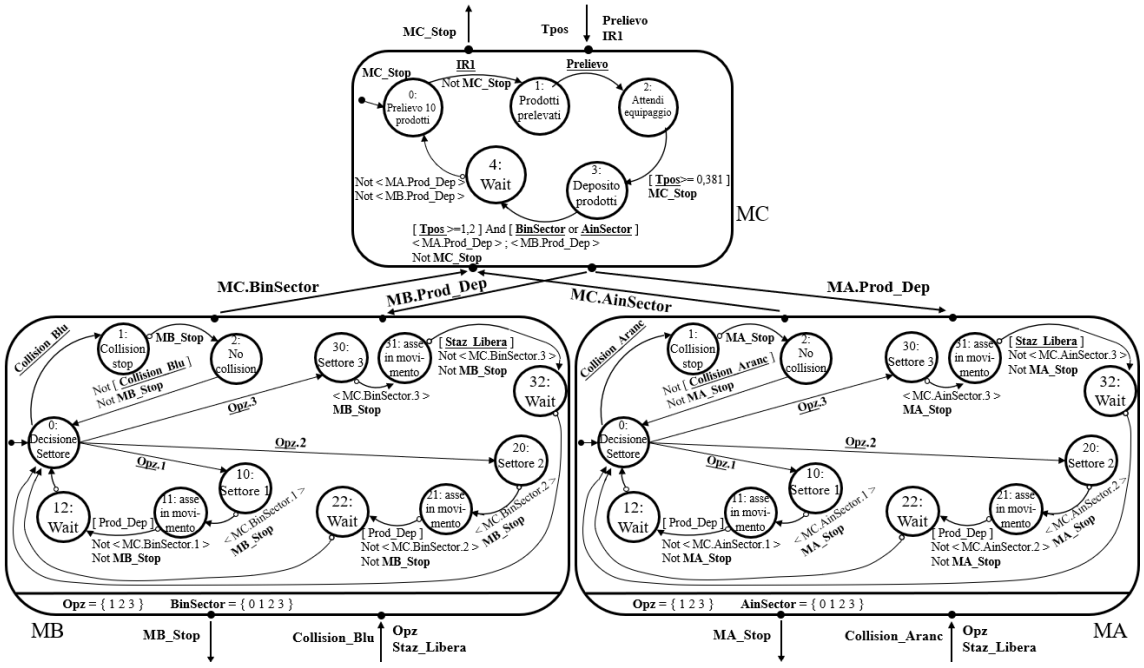


Figura 151 - Implementazione dell'interfaccia di MC, MA e MB

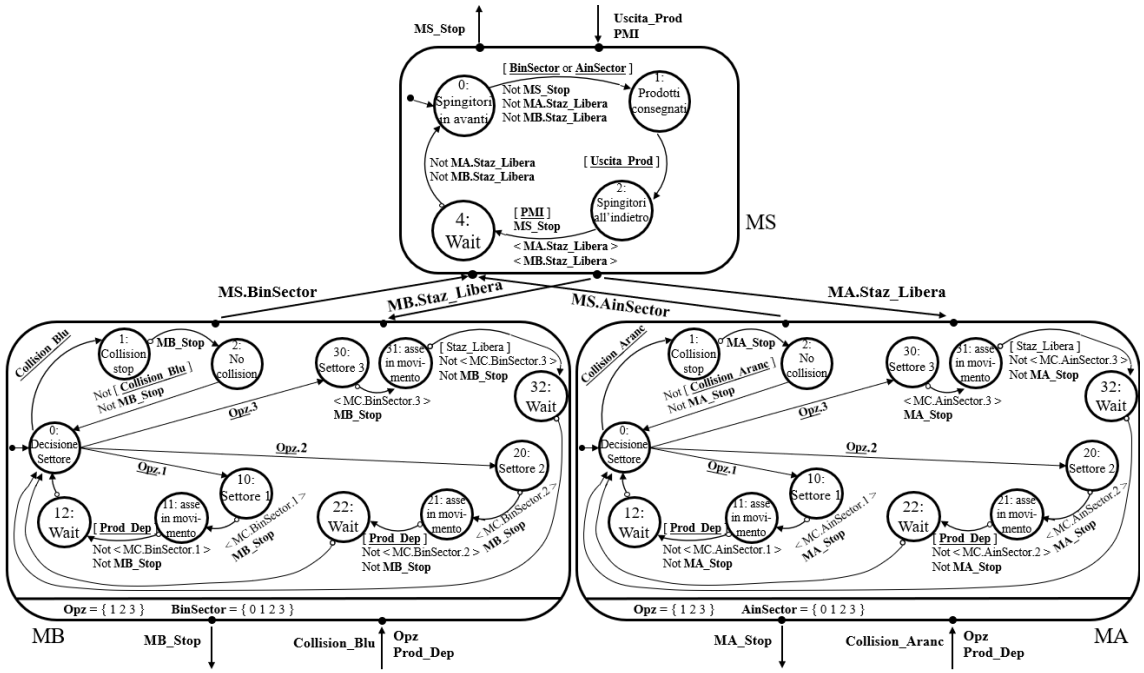


Figura 152 - Implementazione dell'interfaccia di MS, MA e MB

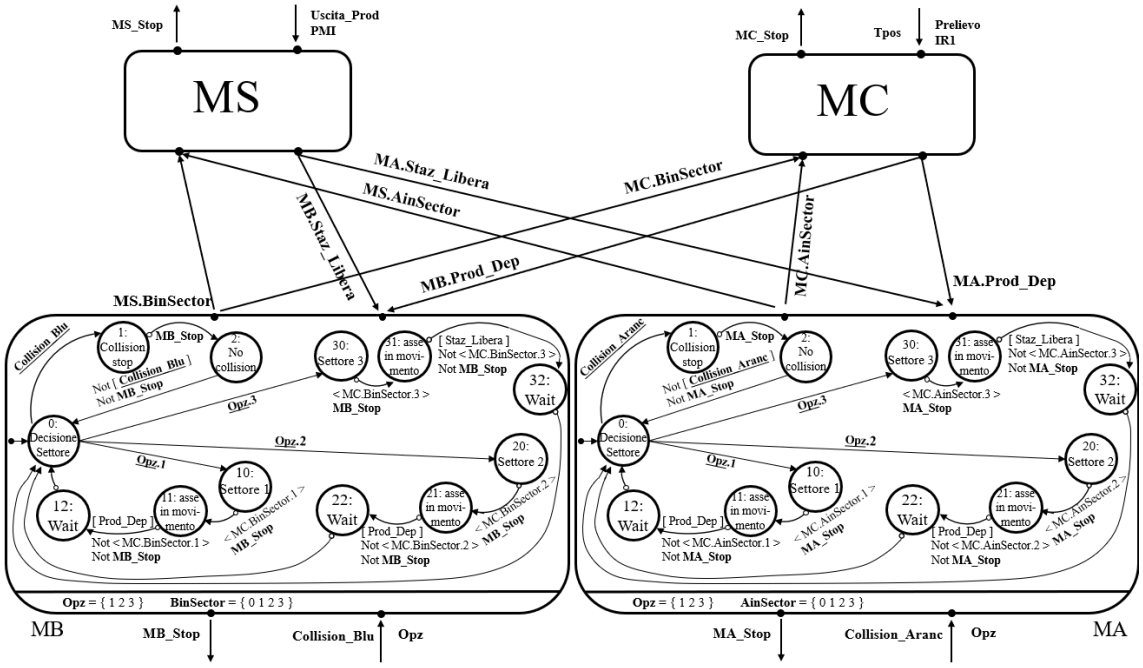


Figura 153 - Implementazione dell'interfaccia di MC, MS, MA e MB

In Figura 154 si può vedere il comportamento completo del sistema sincro-dinamico.

Gli input sono tutti dati provenienti dai sensori, considerati elementi esterni al sistema:

- *Uscita_Prod;*
- *PMI;*
- Tpos;
- Prelievo;
- IR1;
- Catena_Blu_Val;
- Catena_Aranc_Val;
- EinC;
- EinS.

Gli output rappresentano i segnali di controllo che azionano o fermano gli assi master dei vari azionamenti:

- *MA_Stop;*
- *MB_Stop;*
- *MC_Stop;*
- *MS_Stop.*

Attraverso gli output generati da questi oloni è possibile coordinare e sincronizzare il movimento degli equipaggi, che saranno bloccati quando devono eseguire operazioni di caricamento o scaricamento, oppure saranno azionati quando devono spostarsi da una stazione all'altra. La funzione di *collision detection* permette inoltre di evitare la compenetrazione degli equipaggi, garantendo il corretto funzionamento del sistema e svolgendo una importante funzione di supporto durante la fase di prototipazione del sistema.

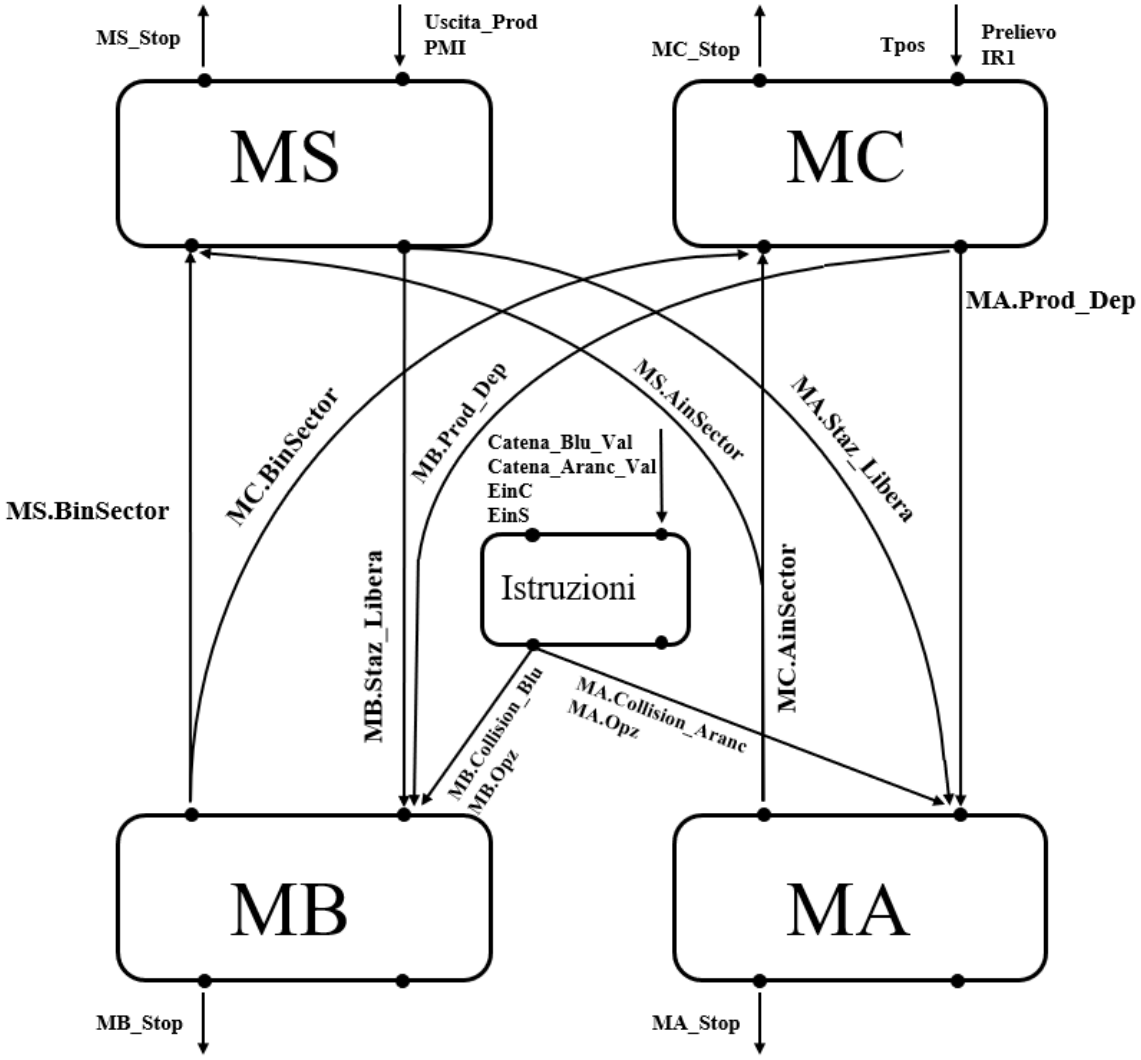


Figura 154 - Oliarchia completa del sistema sincro-dinamico

Al fine di simulare eventi casuali come un fermo macchina o la mancanza di un prodotto, nel modello sono stati aggiunti due pulsanti che permettono all'utente di interagire nella simulazione attraverso lo schermo del computer (Figura 155).

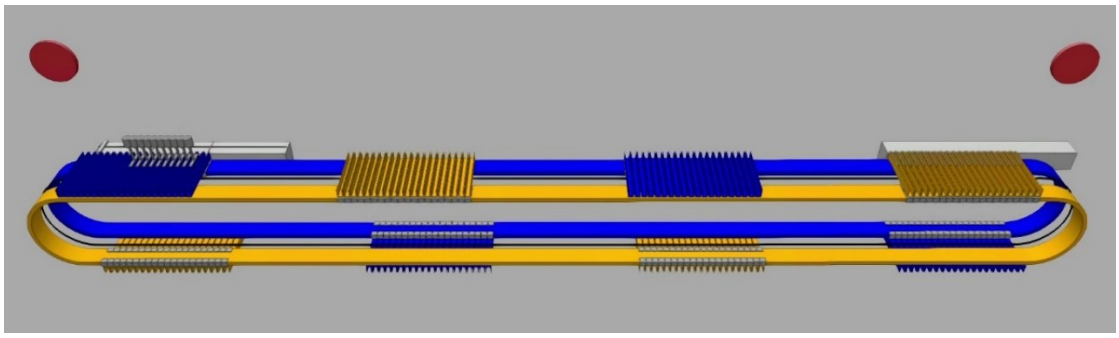


Figura 155 - Pulsanti per l'interazione dell'utente con la simulazione

I prodotti vengono generati da un elemento *transient source* all'inizio della zona di caricamento. Adiacente all'elemento *transient source* è stato inserito, nascosto alla vista, un elemento di tipologia *transient sink* che elimina i prodotti generati dal *transient source*. L'elemento *transient sink* è sempre disattivato e pertanto in una situazione standard non elimina i prodotti, quando invece viene attivato, inizia ad eliminare i prodotti dalla simulazione.

In particolare, il pulsante posizionato a sinistra nella Figura 156, quando premuto, attiva l'elemento *transient sink* e genera dei prodotti mancanti per tutta la durata in cui il pulsante viene premuto (Figura 157).

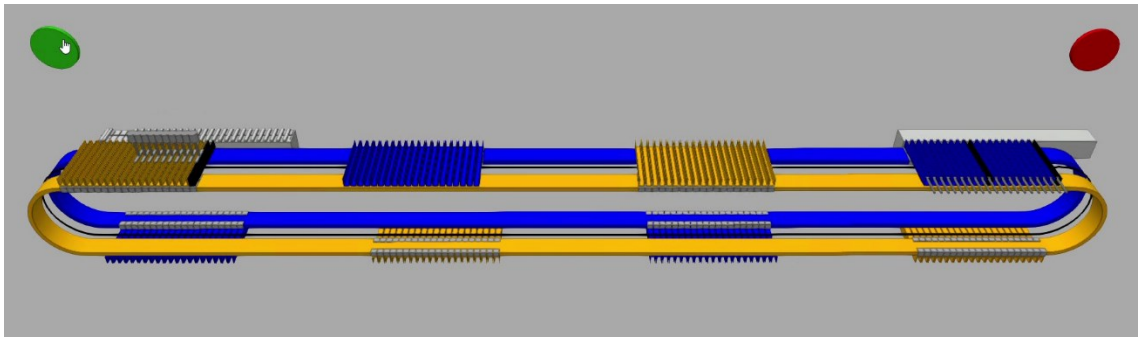


Figura 156 - Pulsante per generare prodotti mancanti

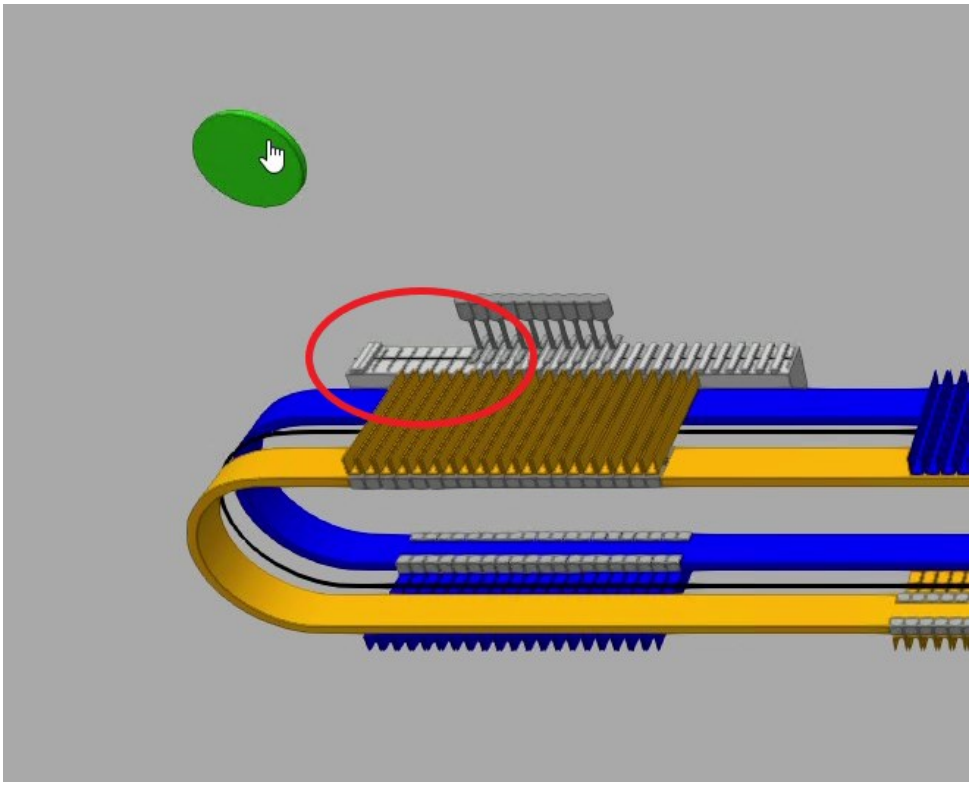


Figura 157 - Prodotti mancanti nella simulazione

L'attivazione dell'elemento *transient sink* tramite pulsante è gestito con il *model script* mostrato in Figura 158.

Questa operazione viene gestita connettendo il parametro *DEACTIVE* (evidenziato in rosso nella Figura 158) che attiva o disattiva l'elemento *transient sink* quando viene impostato come *TRUE* o *FALSE*.

Nello script viene connesso l'input del pulsante *PRESS* (evidenziato in blu nella Figura 158) che assume valore *TRUE* se il pulsante viene cliccato dall'utente, altrimenti restituisce *FALSE*.

Una volta che il pulsante viene rilasciato i prodotti non vengono più eliminati e ricominciano ad essere traspostati correttamente nella zona di caricamento.

BUTTON [ModelScript]

MODEL_SCRIPT(1.5) Time

```
MODEL_SCRIPT(1.5)
// please define used symbols here:
DEFINITIONS:

CONST PRESS := CONNECT(".?ButtonPress");
CONST TOGGLE := CONNECT(".?ButtonToggle");
VAR MANC_PROD := OUTPUT(BOOL,FALSE);
VAR DEACTIVE := CONNECT("../SINK_NO_PRODOTTO?Deactivate");
VAR TRIG := CONNECT("../10_STICK?Trigger");
STATEMENTS:
IF PRESS THEN
    DEACTIVE:=FALSE;
ELSE
    DEACTIVE:=TRUE;
END_IF


END;
```

Figura 158 - Model script per generare prodotti mancanti

Per simulare invece il fermo macchina della macchina a valle del sistema sincro-dinamico si utilizza il pulsante posizionato a destra in Figura 159.

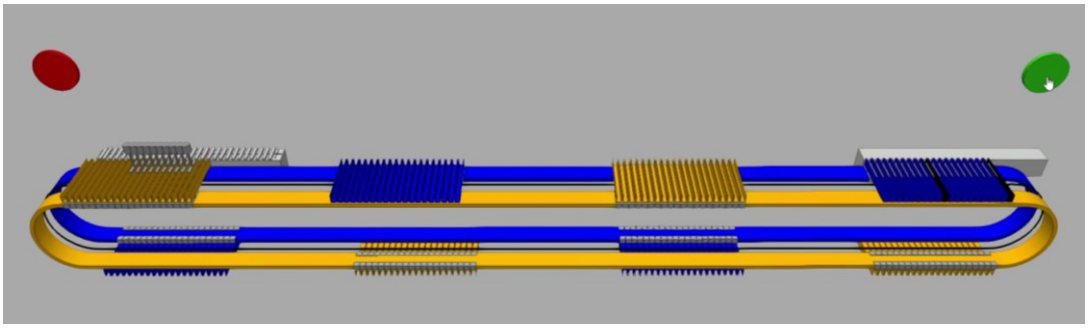


Figura 159 - Pulsante per generare il fermo macchina della macchina a valle

Questo ritardo viene simulato aggiungendo al *model script MASTER_SCARIC* due altri *stati*. Si crea nel *model script* una variabile *local T_RIT* (evidenziata in rosso in Figura 160) che viene utilizzata per impostare un timer. Nel *model script* si collega l'input del pulsante *BUTTON_PRESS* che assume e mantiene il valore *TRUE* quando il pulsante viene premuto dall'utente (evidenziato in blu in Figura 160).

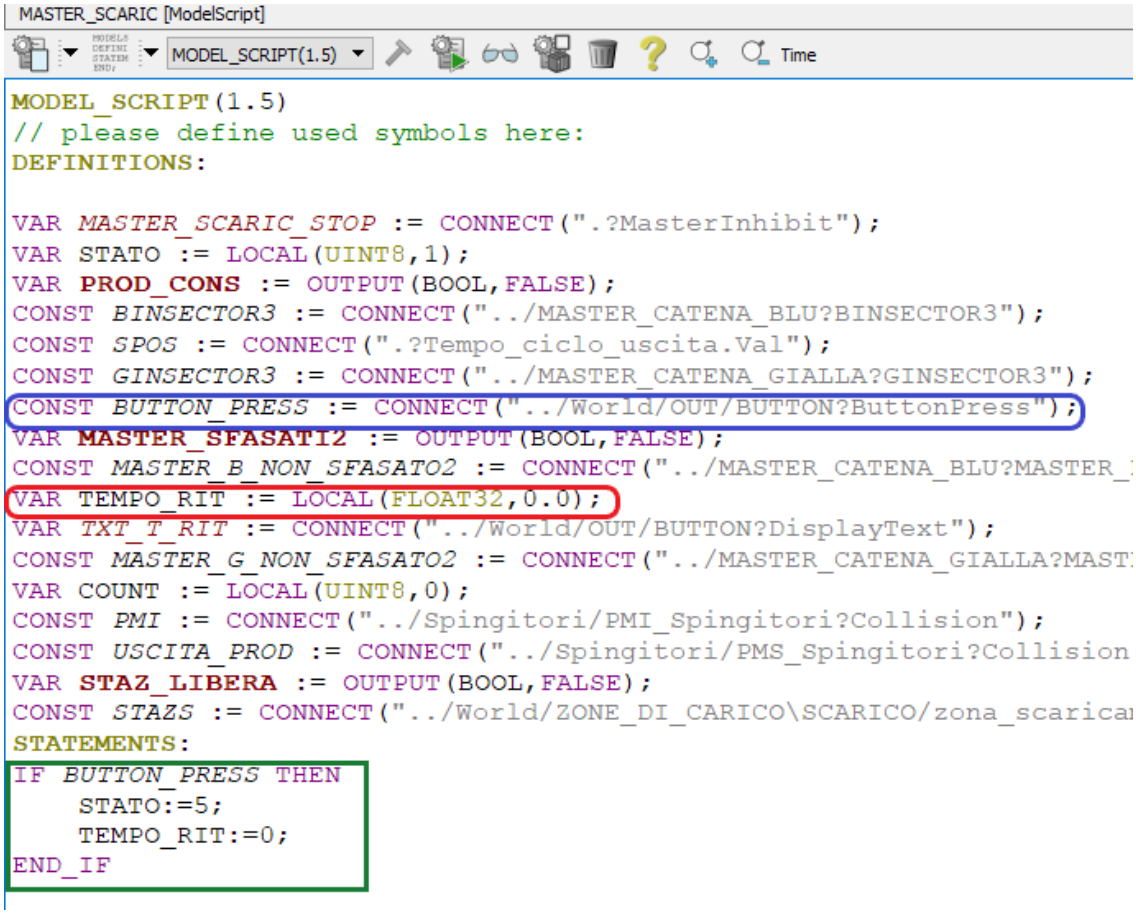


Figura 160 – Implementazione del fermo macchina nel model script di MASTER_SCARIC

All'esterno della funzione <*CASE … OF*> viene scritta la lista di istruzioni evidenziate in verde in Figura 160. Queste istruzioni vengono controllate ad ogni *time_step* di calcolo della simulazione, pertanto se il pulsante viene premuto, indipendentemente da quale stato si trovi il componente *MASTER_SCARIC*, la condizione *IF* diventa soddisfatta e avviene la transizione `t5` allo stato `5: Ritardo uscita` (Figura 161).

Nello stato 5 viene generato l'output <*MS_Stop*>, inoltre si incrementa il valore della variabile *T_RIT* per ogni *time_step della simulazione*.

- Quando viene premuto il pulsante, si blocca l'asse master degli spingitori, di fatto provocando un fermo macchina.

La transizione `t6` avviene quando l'incremento della variabile *T_RIT* raggiunge un valore limite impostato dall'utente (evidenziato in rosso in Figura 161). Si passa allo stato 6.

- La transizione avviene dopo lo scadere di un timer. Nella simulazione il valore limite è stato impostato a 0,39, pertanto il fermo macchina che si è simulato è di 0,39 secondi.

Nello stato `6: Fine ritardo uscita` si resetta a 0 la variabile *T_RIT*.

Dallo stato 6 possono avvenire tre transizioni:

- Se il segnale *Uscita_Prod* è *TRUE*, si esegue una transizione `t7` verso lo stato 1;
- Se il segnale *PMI* è *TRUE*, si esegue una transizione `t8` verso lo stato 3;
- Altrimenti si esegue una transizione `t9` verso lo stato inziale 0.

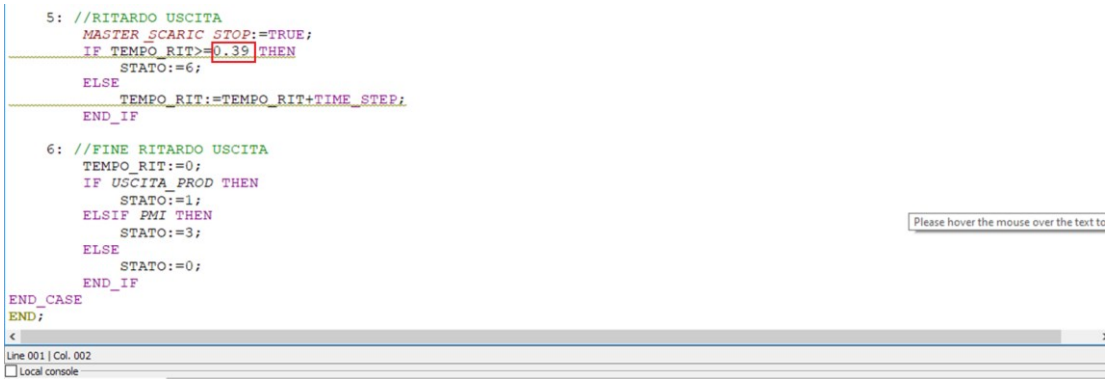


Figura 161 – Stati aggiuntivi implementati nel model script MASTER_SCARIC per generare il fermo macchina.

La nuova interfaccia dell'olone *MS* è rappresentata in Figura 162.

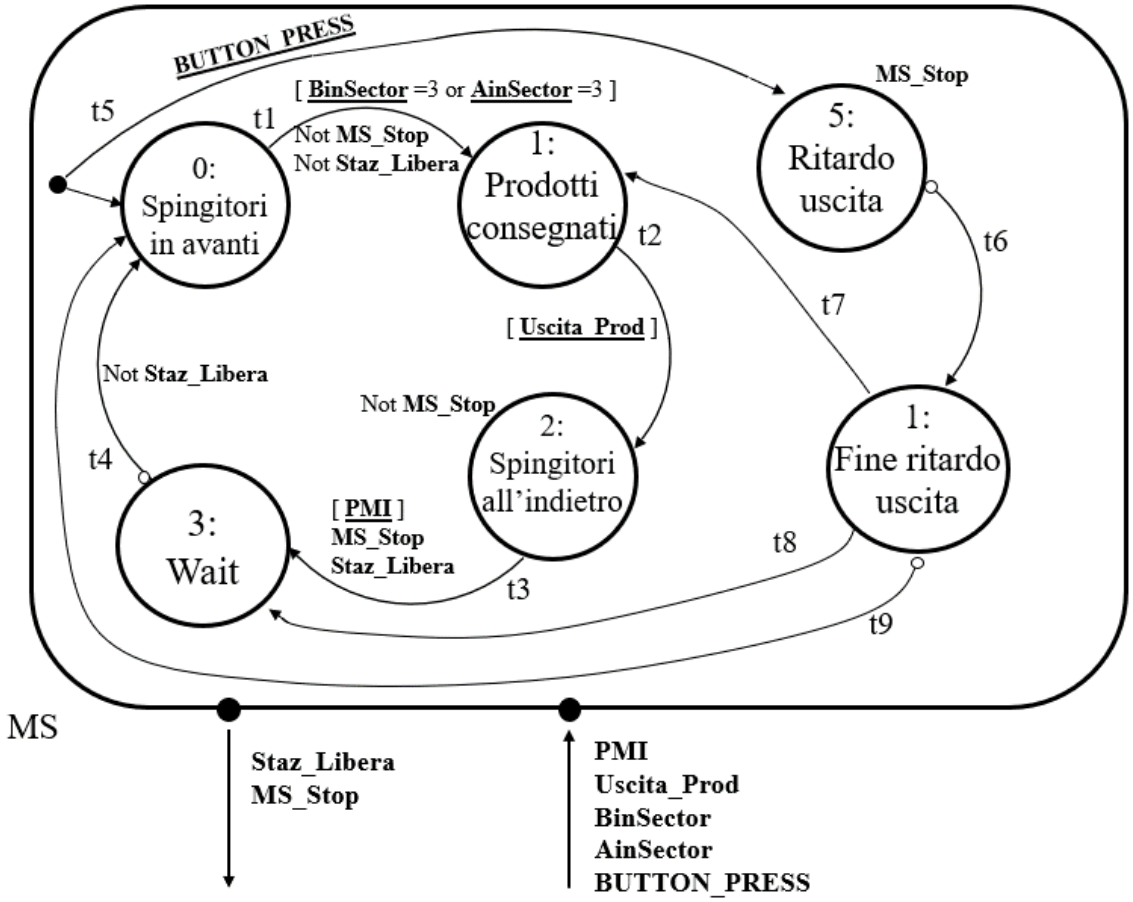


Figura 162 - Interfaccia dell'olone MS con aggiunta di fermo macchina

La programmazione della logica a stati del modello è terminata. Ogni configurazione avrà la stessa logica di funzionamento e la stessa lista di istruzioni. Le uniche configurazioni a presentare alcune differenze sono la 4x40 e la 4x40 con robot di caricamento a passo 60 mm:

- Il *model script MASTER_BLU* (e il *model script* analogo *MASTER_ARANC*), rispetto alla configurazione 8x20, hanno tre *stati* in più;
- Le variabili *BinSector* e *AinSector* assumono i valori *{0 1 2 3 4 5 6}*;
- Gli input *Opz* assumono i valori *{1 2 3 4 5 6}*.

Il resto delle istruzioni è uguale alla configurazione 8x20. Il motivo di queste differenze è dovuto al fatto che gli equipaggi, essendo più lunghi, necessitano di più operazioni di scaricamento e caricamento e di conseguenza vengono aggiunti i corrispettivi stati. L'interfaccia dell'olone MB della configurazione 4x40 e 4x40 con

robot di caricamento a 60 mm è visualizzato in Figura 163. Gli stati aggiuntivi del *model script* sono visualizzabili in Figura 164 e Figura 165.

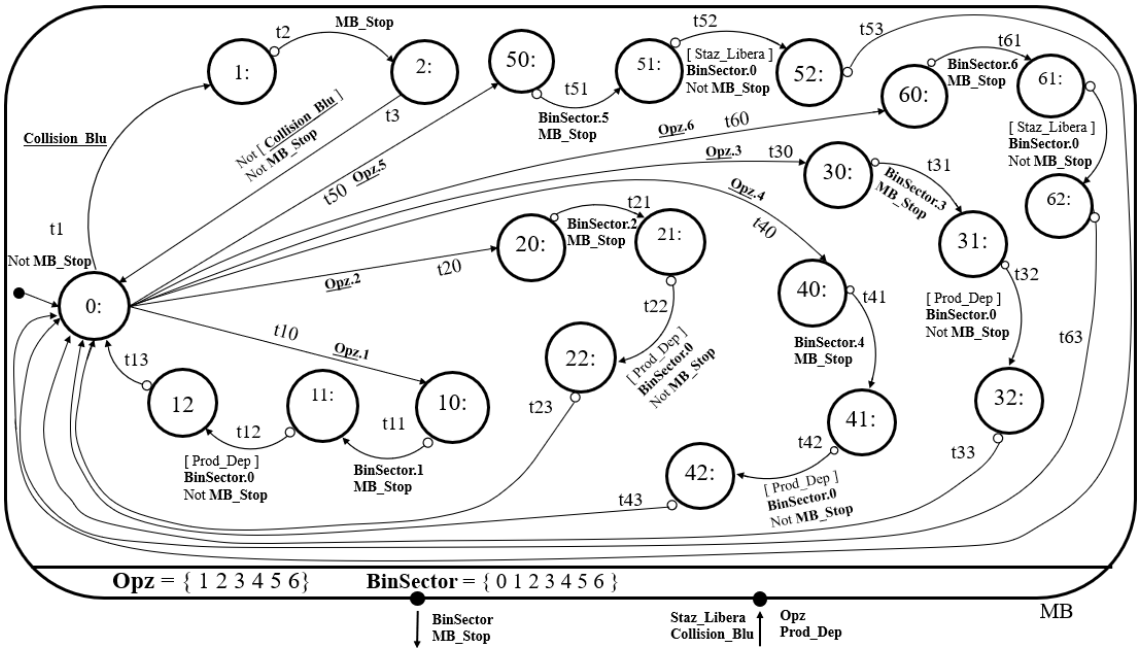


Figura 163 - Interfaccia dell'olone MB nella configurazione 4x40

```
CASE STATO OF
    0: //DECISIONE SETTORE
       IF OPZ1 THEN
           STATO:=10;
       ELSIF OPZ2 THEN
           STATO:=20;
       ELSIF OPZ3 THEN
           STATO:=30;
       ELSE
           MASTER_BLU_STOP:=FALSE;
       END_IF

    1: //COLLISION STOP
       MASTER_BLU_STOP:=TRUE;
       STATO:=2;
    2: //NON C'è PIù COLLISION
       IF NOT COLLISION_BLU THEN
           MASTER_BLU_STOP:=FALSE;
           STATO:=0;
       END_IF

    10: //EQUIPAGGIO IN SETTORE 1
        BINSECTOR1:=TRUE;
        MASTER_BLU_STOP:=TRUE;

    11: //ASSE IN MOVIMENTO
        IF PROD_DEPOSIT THEN
            MASTER_BLU_STOP:=FALSE;
            BINSECTOR1:=FALSE;
            STATO:=12;
        END_IF

    12: //WAIT
        IF COUNT=5 THEN
            COUNT:=0;
            STATO:=0;
        ELSE
            COUNT:=COUNT+1;
        END_IF

    20: //EQUIPAGGIO IN SETTORE 2
        BINSECTOR2:=TRUE;
        MASTER_BLU_STOP:=TRUE;
        STATO:=21;

    21: //ASSE IN MOVIMENTO
        IF PROD_DEPOSIT THEN
            MASTER_BLU_STOP:=FALSE;
            BINSECTOR2:=FALSE;
            STATO:=22;
        END_IF

    22: //WAIT
        IF COUNT=5 THEN
            COUNT:=0;
            STATO:=0;
        ELSE
            COUNT:=COUNT+1;
        END_IF
    30: //EQUIPAGGIO IN SETTORE 3
        BINSECTOR3:=TRUE;
        MASTER_BLU_STOP:=TRUE;
        STATO:=31;

    31: //ASSE IN MOVIMENTO
        IF PROD_DEPOSIT THEN
            MASTER_BLU_STOP:=FALSE;
            BINSECTOR3:=FALSE;
            STATO:=32;
        END_IF

    32: //WAIT
        IF COUNT=5 THEN
            COUNT:=0;
            STATO:=0;
        ELSE
            COUNT:=COUNT+1;
        END_IF
```

Figura 164 – Parte 1 del model script di MASTER_BLU nella configurazione 4x40

```
40: //EQUIPAGGIO IN SETTORE 4
    BINSECTOR4:=TRUE;
    MASTER_BLU_STOP:=TRUE;
    STATO:=41;

41: //ASSE IN MOVIMENTO
    IF PROD_DEPOSIT THEN
        MASTER_BLU_STOP:=FALSE;
        BINSECTOR4:=FALSE;
        STATO:=42;
    END_IF

42: //WAIT
    IF COUNT=5 THEN
        COUNT:=0;
        STATO:=0;
    ELSE
        COUNT:=COUNT+1;
    END_IF

50: //EQUIPAGGIO IN SETTORE 5
    BINSECTOR5:=TRUE;
    MASTER_BLU_STOP:=TRUE;
    STATO:=51;

51: //ASSE IN MOVIMENTO
    IF STAZ_LIBERA THEN
        MASTER_BLU_STOP:=FALSE;
        BINSECTOR5:=FALSE;
        STATO:=52;
    END_IF

52: //WAIT
    IF COUNT=5 THEN
       COUNT:=0;
       STATO:=0;
    ELSE
       COUNT:=COUNT+1;
    END_IF

60: //EQUIPAGGIO IN SETTORE 6
    BINSECTOR6:=TRUE;
    MASTER_BLU_STOP:=TRUE;
    STATO:=61;

61: //ASSE IN MOVIMENTO
    IF STAZ_LIBERA THEN
        MASTER_BLU_STOP:=FALSE;
        BINSECTOR6:=FALSE;
        STATO:=62;
    END_IF

62: //WAIT
    IF COUNT=5 THEN
       COUNT:=0;
       STATO:=0;
    ELSE
       COUNT:=COUNT+1;
    END_IF
```

Figura 165 - Parte 2 del model script di MASTER_BLU nella configurazione 4x40

5.3. Simulazione del modello e confronto delle configurazioni

Il modello viene simulato eseguendo il comando *run*. Al primo istante tutti gli azionamenti si mettono nella loro posizione iniziale e rimangono fermi in attesa del segnale di start. Inizialmente i prodotti non sono presenti nella zona di prelievo perché non sono ancora stati generati in numero sufficiente e nessun componente inizia il movimento.

Una volta che dieci prodotti giungono nella zona di prelievo il robot riceve il segnale di start e inizia le operazioni seguendo la logica a stati predefinita.

Contemporaneamente al robot si è scelto di far iniziare il movimento degli spingitori. Far partire gli spingitori in un certo istante piuttosto che un altro influenza la costruzione della carta delle operazioni ma solo fino alla conclusione del periodo di warm-up: se la sequenza di alzate è stata studiata correttamente il modello si sincronizza automaticamente rivelando la potenza dello strumento simulativo. Lo stesso ragionamento vale anche se vengono inseriti dei ritardi in maniera casuale: il modello dopo un tempo iniziale variabile si risincronizza automaticamente.

Quando il robot deposita i primi dieci prodotti, una catena di equipaggi inizia il movimento; quando gli spingitori completano la loro alzata un'altra catena di equipaggi inizia il movimento.

In questa fase iniziale le alzate impostate nella legge di moto complessiva e i segnali ricevuti dai sensori potrebbero far avanzare in modo desincronizzato gli equipaggi con il rischio di collisione tra le due catene. Tuttavia, la funzione di *collision detection* impedisce la compenetrazione e permette lo start and stop delle catene di equipaggi. Dopo un certo tempo gli equipaggi si troveranno automaticamente in una posizione relativa tale che le alzate saranno sempre eseguite in modo ripetitivo e uguale. Quando si giunge a questa situazione si può dire che è finito il periodo di warm-up e le macchine sono tutte sincronizzate.

La sequenza di operazioni susseguente determina il tempo ciclo del sistema sincro-dinamico. Prendendo come riferimento l'asse master di una delle due catene di equipaggi, il tempo ciclo è determinato dal tempo intercorso da quando l'asse inizia la prima delle alzate a quando conclude l'ultima.

Dunque, dalla simulazione si ottengono il tempo ciclo del sistema sincro-dinamico e la sequenza di operazioni, compresi i tempi di operazione comune per il carico/scarico

prodotti e i tempi di ozio. Tutti questi tempi possono essere utilizzati per costruire facilmente la carta delle operazioni.

Eseguendo tutte le simulazioni si ottengono sei carte delle operazioni (Figura 166, Figura 167, Figura 168, Figura 169, Figura 170 e Figura 171). Le caselle lasciate di colore bianco indicano la presenza di un tempo di ozio nella sequenza delle operazioni.

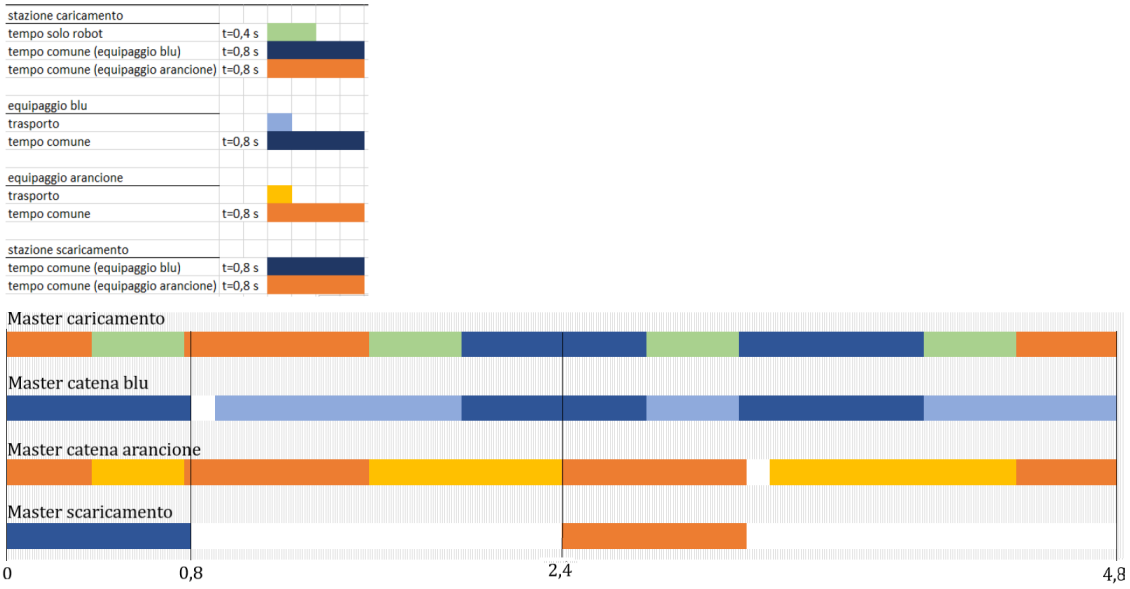


Figura 166 - Carta operazioni configurazione 8x20. Valori dell'asse delle ascisse in secondi

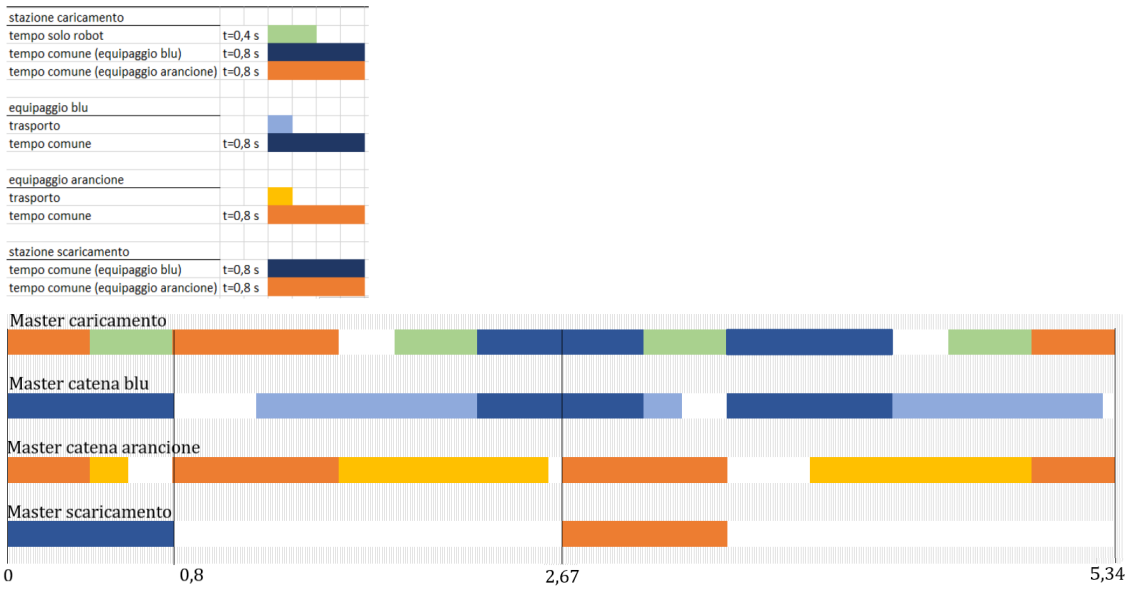


Figura 167 - Carta operazioni configurazione 8x20 con caricamento a passo 60 mm. Valori dell'asse delle ascisse in secondi

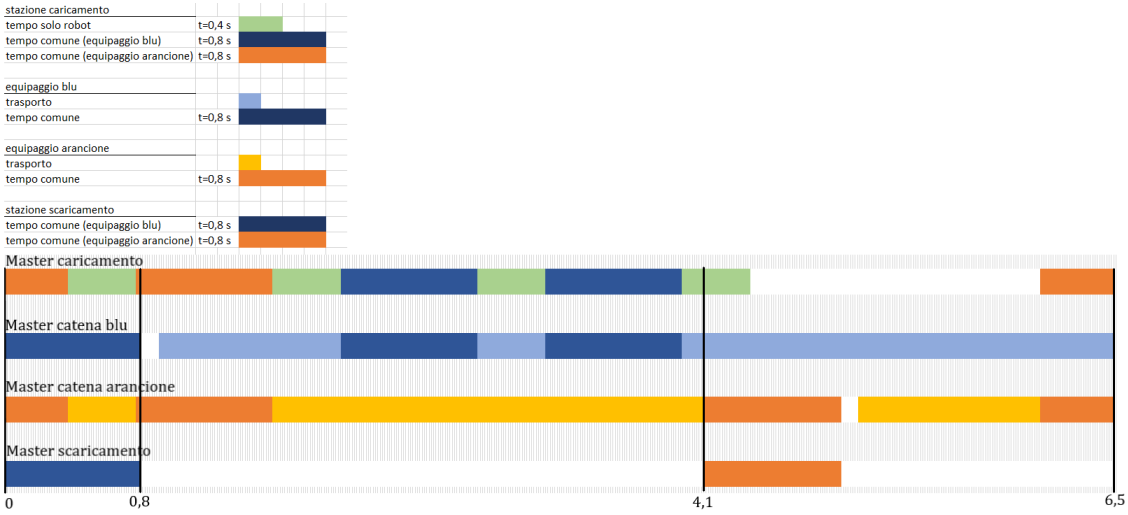


Figura 168 - Carta operazioni configurazione 4x20. Valori dell'asse delle ascisse in secondi

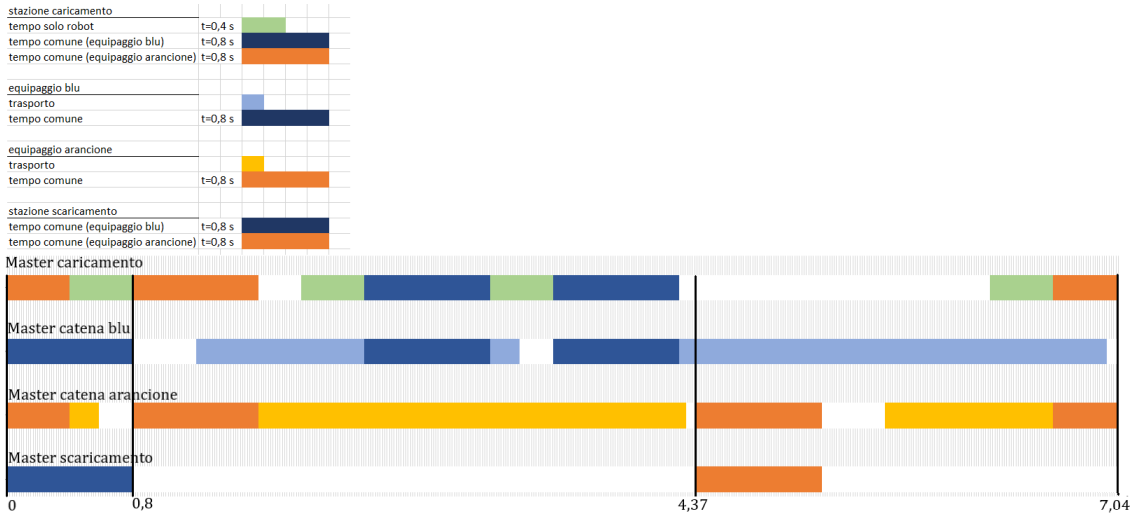


Figura 169 - Carta operazioni configurazione 4x20 con caricamento a passo 60 mm. Valori dell'asse delle ascisse in secondi

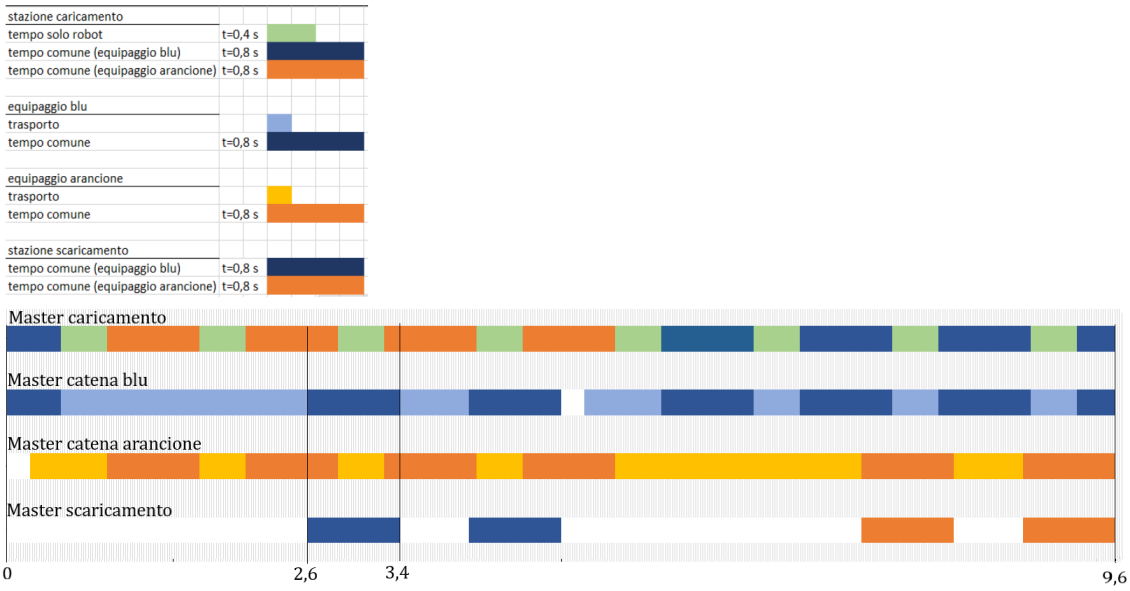


Figura 170 - Carta operazioni configurazione 4x40. Valori dell'asse delle ascisse in secondi

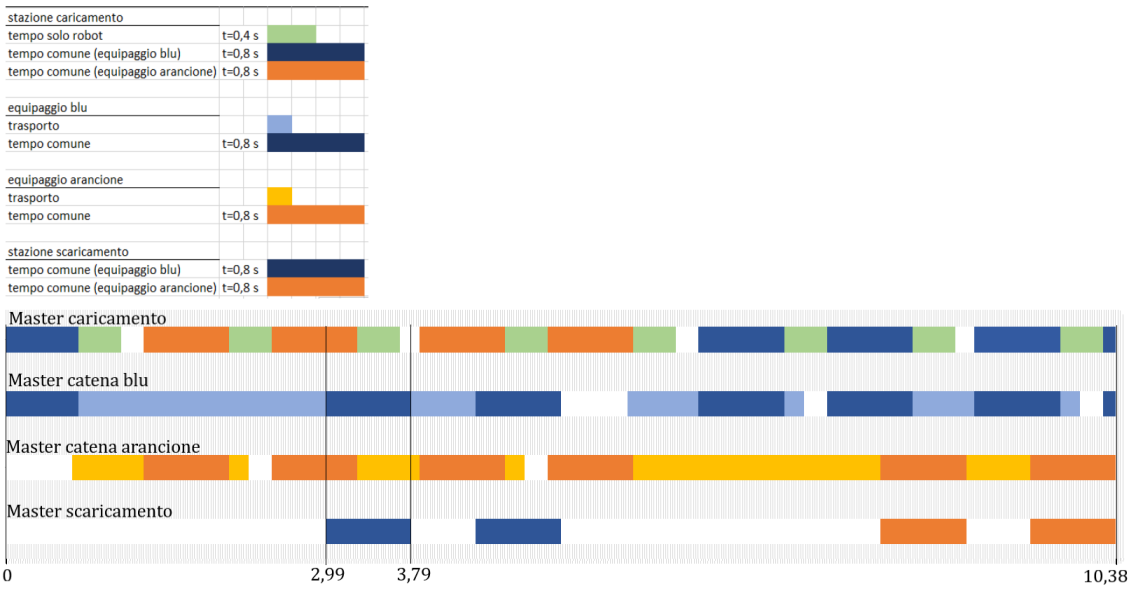


Figura 171 - Carta operazioni configurazione 4x40 con caricamento a passo 60 mm. Valori dell'asse delle ascisse in secondi

Dal confronto di queste carte è possibile immediatamente decretare il tempo ciclo totale delle configurazioni e la produttività.

Guardando la sequenza di operazioni dell'asse "Master scaricamento" si può notare che oltre al tempo di ozio le uniche operazioni eseguite sono quelle del tempo comune per la consegna dei prodotti. Pertanto, il numero di consegne effettuate nel ciclo complessivo è facilmente ricavabile guardando all'asse Master scaricamento, da cui è possibile ricavare la produttività delle configurazioni (Figura 172).

Configurazione 8x20			
Tempo ciclo [s]	n° operazioni di scaricamento nel ciclo	Prodotti scaricati in una operazione	Produtività [Prodotti/min]
4,8	2	20	500

Configurazione 8x20 con caricamento a passo 60 mm			
Tempo ciclo [s]	n° operazioni di scaricamento nel ciclo	Prodotti scaricati in una operazione	Produtività [Prodotti/min]
5,34	2	20	449,4382022

Configurazione 4x20			
Tempo ciclo [s]	n° operazioni di scaricamento nel ciclo	Prodotti scaricati in una operazione	Produtività [Prodotti/min]
6,5	2	20	369,2307692

Configurazione 4x20 con caricamento a passo 60 mm			
Tempo ciclo [s]	n° operazioni di scaricamento nel ciclo	Prodotti scaricati in una operazione	Produtività [Prodotti/min]
7,04	2	20	340,9090909

Configurazione 4x40			
Tempo ciclo [s]	n° operazioni di scaricamento nel ciclo	Prodotti scaricati in una operazione	Produtività [Prodotti/min]
9,6	4	20	500

Configurazione 4x40 con caricamento a passo 60 mm			
Tempo ciclo [s]	n° operazioni di scaricamento nel ciclo	Prodotti scaricati in una operazione	Produtività [Prodotti/min]
10,38	4	20	462,4277457

Figura 172 - Confronto produttività configurazioni

Dalla Figura 172 si evince che le configurazioni con caricamento a passo 60 mm non garantiscono la produttività di 500 prodotti al minuto e pertanto non sono configurazioni accettabili. Anche la configurazione 4x20 non soddisfa i requisiti. Le configurazioni 8x20 e 4x40 invece soddisfano entrambe la produttività richiesta a pari merito.

Tra queste due configurazioni è possibile fare un ulteriore confronto guardando al tempo di ozio degli assi "*Master catena blu*" e "*Master catena arancione*".

Nella configurazione 8x20 il tempo di ozio è pari a 0,1 secondi (Figura 173), mentre nella configurazione 4x40 è pari a 0,2 secondi (Figura 174).

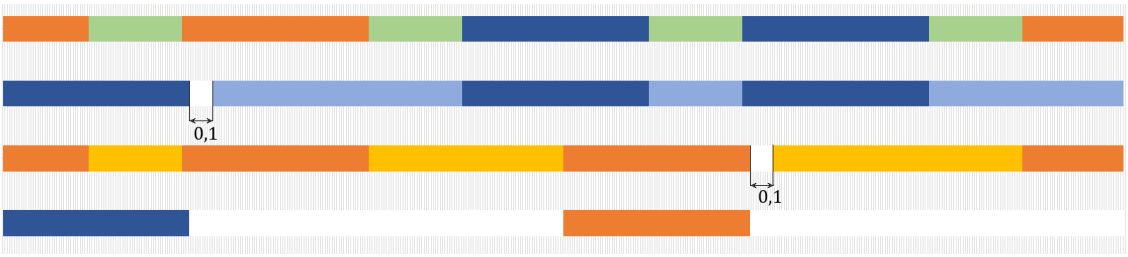


Figura 173 - Configurazione 8x20; Tempo di ozio negli assi "Master catena blu" e "Master catena arancione"

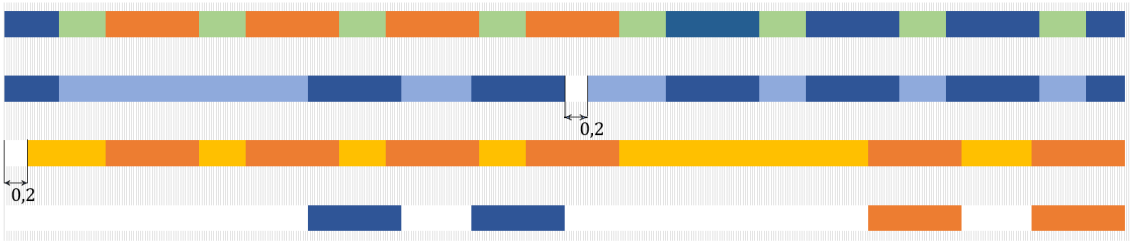


Figura 174 - Configurazione 4x40; Tempo di ozio negli assi "Master catena blu" e "Master catena arancione"

Il tempo di ozio della configurazione 4x40, all'interno di un ciclo completo, è maggiore del tempo ciclo della configurazione 8x20.

Questa informazione ci dice due cose:

- Il tempo di ozio è presente solo dopo una operazione di scaricamento. Questo vuol dire che solo il movimento dopo lo scaricamento ha del tempo in più per essere seguito, mentre gli altri movimenti dei due assi sono necessariamente da eseguire con una legge di moto a tempo minimo, raggiungendo il più in fretta possibile la velocità massima consentita dai motori.
- Il tempo di ozio della configurazione 4x40, essendo più grande, permette di eseguire l'alzata corrispondete in un tempo maggiore.

Se si considera nuovamente una legge di moto $f(p)$ con alzata e corsa unitaria e si confronta con essa una generica legge di moto $q(p)$, si ottiene:

$$q(p) = h\ f\left(\frac{p}{T}\right)$$

dove h è l'alzata e T la corsa della legge di moto $q(p)$. Considerando le derivate di ordine superiore si ha:

$$\dot{q}(p) = \frac{h}{T}\ \dot{f}\left(\frac{p}{T}\right)$$

$$\ddot{q}(p) = \frac{h}{T^2}\ \ddot{f}\left(\frac{p}{T}\right)$$

$$\dddot{q}(p) = \frac{h}{T^3}\ \dddot{f}\left(\frac{p}{T}\right)$$

e quindi, aumentando o riducendo la corsa totale di un fattore T, la velocità varia di un fattore T, mentre l'accelerazione varia di un fattore T^2. In base a questa considerazione si può affermare che è più conveniente sviluppare l'alzata sulla massima corsa possibile, in modo da massimizzare T, e quindi ridurre velocità ed accelerazioni massime (nell'ultimo caso di un fattore quadratico) [19].

Si utilizza quindi il tempo di ozio come indice per stabilire quale tra le configurazioni può massimare la corsa T di una determinata alzata.

Consideriamo $k = \frac{h}{T^2}$ e confrontiamo le configurazioni 8x20 e 4x40:

- Configurazione 8x20

  Alzata a tempo minimo: $k'_{8x20} = \frac{1300}{1{,}067^2} = 1218{,}37$

  Alzata con tempo di ozio: $k''_{8x20} = \frac{1300}{1{,}167^2} = 1113{,}97$

  $c_{8x20} = \frac{k'_{8x20}}{k''_{8x20}} = 1{,}094$

- Configurazione 4x40

  Alzata a tempo minimo: $k'_{4x40} = \frac{700}{0{,}667^2} = 1049{,}48$

  Alzata con tempo di ozio: $k''_{4x40} = \frac{700}{0{,}867^2} = 807{,}38$

$$c_{4x40} = \frac{k'_{4x40}}{k''_{4x40}} = 1{,}3$$

Si ottiene che $c_{4x40} > c_{8x20}$, che vuol dire che nella configurazione 4x40 l'accelerazione di picco di quella particolare alzata può essere diminuita del 30% rispetto al 9,4% della configurazione 8x20.

Sulla base di queste considerazioni si può affermare che la configurazione ottima del sistema risulta essere la configurazione 4x40.

Conclusioni

Lo scopo di questo elaborato di tesi è di ottimizzare un sistema sincro-dinamico partendo dai dati di progetto forniti dall'azienda Marchesini Group.

In questo elaborato di tesi si è studiato il sistema in tutte le sue parti: dal punto di vista dell'ottimizzazione con la formulazione di un algoritmo che genera le configurazioni di macchina e con la simulazione virtuale, dal punto di vista meccanico con lo studio delle leggi di moto, dal punto di vista logistico con l'analisi delle operazioni e la loro sincronizzazione e infine, dal punto di vista dell'automazione con la programmazione a stati del sistema.

Per ottimizzare al meglio il sistema sono stati utilizzati i paradigmi introdotti con l'avvento dell'industria 4.0. Si è costruito un modello simulativo che permette la valutazione della prestazione di una configurazione di macchina e tale modello è stato costruito in maniera modulare: è possibile applicare facilmente delle modifiche, aggiungere o eliminare altri sistemi alla simulazione. Questo aspetto dona un enorme valore aggiunto al modello simulativo realizzato.

La simulazione prende in input delle varianti di macchina generate a partire dall'algoritmo di generazione delle configurazioni, sviluppato appositamente per l'ottimizzazione del sistema. Questa metodologia permette di sfruttare al meglio il modello simulativo e può adattarsi in maniera efficace ad una realtà aziendale di alto livello.

Il modello simulativo sfrutta la tecnologia del virtual commissioning, permettendo di eseguire la simulazione del software che controlla la macchina sincro-dinamica. Questo studio può essere utilizzato come base per eseguire successive migliorie al software di altre macchine che sono simili al sistema studiato, e di accelerare il processo di messa in opera di tali sistemi.

La programmazione del sistema ha seguito il formalismo della logica a stati ed in particolare si è sfruttato il linguaggio grafico dei diagrammi di stato parte-intero, utilizzati nell'ambito dell'*internet of things*, in questo modo il sistema è in grado di sincronizzarsi attraverso la comunicazione tra gli oloni che gestiscono i vari componenti della macchina sincro-dinamica. La modularità di questo approccio garantisce una enorme flessibilità per

la simulazione di nuove configurazioni o per l'implementazione di altre modifiche al sistema.

Lo studio di questo sistema sincro-dinamico ha portato alla generazione di sei differenti configurazioni di macchina.

Ognuna di queste configurazioni è stata simulata attraverso l'implementazione di leggi di moto nella simulazione virtuale e come risultato delle simulazione si sono ottenute le carte delle operazioni di ciascuna configurazione.

Dallo studio delle carte delle operazioni si è potuto osservare che i sistemi che adottano un robot di caricamento con passo di 60 mm e la configurazione 4x20 non riescono a soddisfare la produttività nominale di progetto. Invece, le configurazioni 8x20 e 4x40 soddisfano i requisiti di progetto e attraverso uno studio dettagliato dei tempi di ozio si è potuto decretare che la configurazione 4x40 lavora in condizione più vantaggiose rispetto alla configurazione 8x20.

La configurazione 4x40 viene dichiarata come la configurazione ottima con cui far lavorare il sistema sincro-dinamico.

Bibliografia

[1] H. Kagermann, R. Anderl, J. Gausemeier, G. Schuh e W. Wahlster, «Industrie 4.0 in a Global Context: Strategies for Cooperating with International Partners,» acatech, 2016.

[2] C. R. Harrell e D. A. Hicks, «Simulation software component architecture for simulation-based enterprise applications,» in *Proceedings of the 1998 Winter Simulation Conference*, 1998.

[3] B. Rodič, «Industry 4.0 and the New Simulation Modelling Paradigm,» *Organizacija,* vol. 50, n. 3, pp. 193-207, 2017.

[4] Z. Liu, N. Suchold e C. Diedrich, «Virtual Commissioning of Automated Systems,» *Automation,* pp. 131-148, 2012.

[5] M. P. Groover, Fundamentals of Modern Manifacturing - Material, Processes and Systems, Hoboken, NJ: John Wiley & Sons, Inc., 2010.

[6] D. de Champeaux, Object-Oriented System Development, Addison-Wesley, 1993.

[7] T. L. Booth, Sequential Machines and Automata Theory, John Wiley and Sons, Inc., 1967.

[8] J. E. Hopcroft, R. Motwani e J. D. Ullman, Introduction to Automata Theory, Languages, and Computations, Pearson, Addison-Wesley, 2006.

[9] D. Harel, «Statecharts: a visual formalism for complex systems,» *Science of Computer Programming,* n. 8, pp. 231-274, 1987.

[10] L. Pazzi e M. Pellicciari, «From the Internet of Things to Cyber-Physical Systems: the Holonic Perspective,» *Procedia Manufacturing,* n. 11, pp. 989-995, 2017.

[11] R. Matteo, «Implementazione di costrutti modulari a stati per automatizzare il funzionamento di una pinza autocentrante e relativo aggancio ad un robot mediante il software TwinCAT 3,» Università degli studi di Modena e Reggio Emilia, Modena, 2020.

[12] L. M. De Nardo, «packagingobserver,» 5 Luglio 2013. [Online]. Available: https://www.packagingobserver.com. [Consultato il giorno 13 Agosto 2021].

[13] A. Nosarka, «Safeguard your profits with filling accuracy,» *SOUTH AFRICAN PHARMACEUTICAL & COSMETIC REVIEW,* vol. 45, n. 9, pp. 44-45, 2018.

[14] A. Messac, Optimization in Practice with MATLAB® for Engineering Students and Professionals, New York: Cambridge University Press, 2015.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest e C. Stein, Introduction to Algorithms, Cambridge, Massachusetts: The MIT Press, 2009.

[16] The MathWorks, Inc., «Matlab Help Center,» MathWorks, 2021. [Online]. Available: https://it.mathworks.com/help/matlab/matlab_prog/vectorization.html. [Consultato il giorno 22 Ottobre 2021].

[17] S. Konz e S. Johnson, Work Design, London: CRC Press, 2016.

[18] A. Pareschi, Impianti industriali - Criteri di scelta, progettazione e realizzazione, Bologna: Società editrice Esculapio s.r.l., 2007.

[19] G. Canini e C. Fantuzzi, Controllo del moto per macchine automatiche, Bologna: Innopak s.r.l., 2001.

[20] L. Biagiotti e C. Melchiorri, Trajectory Planning for Automatic Machines and Robots, Modena: Springer, 2008.