POLITECNICO DI TORINO

Corso di Laurea Magistrale

in Ingegneria Meccanica indirizzo Produzione

Tesi di Laurea Magistrale

Sviluppo di un modello di simulazione a eventi discreti per la valutazione delle performance di un magazzino automatico in grado di svolgere operazioni di picking interno

Relatori:

Franco Lombardi

Giulia Bruno

Alberto Faveto

Traini Emiliano

Candidato:

Simone Ferrari

ANNO ACCADEMICO 2020/2021

SOMMARIO

1. Abstract....7
2. Introduzione agli argomenti trattati....8
2.1. Logistica: definizione ed evoluzione....8
2.2. Sistemi di immagazzinamento automatizzati e picking....8
2.2.1. Magazzini automatici con trasloelevatori o carrelli a guida autonoma....10
2.2.2. Magazzini automatici verticali, orizzontali e dispenser....11
2.2.3. Magazzini automatici basati sull'utilizzo di AGV....13
2.2.4. Magazzini automatici basati sull'utilizzo di navette e analisi del sistema navetta-satellite ESMARTSHUTTLE®....13
2.3. Simulazione ad eventi discreti....14
2.3.1. Flexsim®....15
3. Assunzioni sul modello e approccio alla simulazione....16
3.1. Approccio flessibile alla creazione del modello di simulazione ad eventi discreti....16
3.2. Modifiche necessarie proposte al sistema ESMARTSHUTTLE®....17
4. Costruzione del modello....18
4.1. Creazione della zona delle scaffalature....19
4.2. Creazione della zona parcheggi....19
4.3. Creazione degli ESMARTSHUTTLE®....20
5. Logiche di simulazione....21
5.1. Modulo logico di inizializzazione del magazzino....21
5.1.1. Sotto-modulo di creazione dei pallet in magazzino....23
5.2. Modulo logico di gestione degli ordini....24
5.2.1. Sub-routine di movimentazione....25
5.2.2. Sub-routine di picking....28
5.3. Modulo logico di rifornimento del magazzino....30

6. Confronto con il benchmark 32
7. Ottimizzazione con analisi di sensitività 35
7.1. Studio dell'influenza del numero di corridoi e livelli 35
7.2. Studio dell'andamento degli indicatori al variare del numero di navette 37
7.2.1. Regressione lineare 40
7.2.2. Regressione polinomiale di secondo grado 41
8. Conclusioni e possibili sviluppi futuri del modello 45
8.1. Conclusioni 45
8.2. Possibili sviluppi futuri del modello 45
8.2.1. Studio della prestazione legata al posizionamento in magazzino e studi legati alle logiche di rifornimento 45
8.2.2. Analisi regressive per individuazione del tempo medio 46
9. Allegato 1 – codice generazione modello 47
11. Allegato 3 – tabella analisi della varianza numero ideale di navette 68
12. Allegato 4 – confronto regressione logistica e lineare 69
13. Riferimenti bibliografici 72

INDICE DELLE FIGURE

Figura 1 – schema riassuntivo dei magazzini automatizzati 10
Figura 2 - sistema AS/RS con trasloelevatore [4] 10
Figura 3 - magazzino automatico orizzontale [4] 11
Figura 4 - magazzino automatico verticale 11
Figura 5 - sistema automatico basato su dispenser [4] 12
Figura 6 - RCSRS prodotto da AutoStore ™ 12
Figura 7 - magazzino automatico con RMFS [5] 13
Figura 8 - sistema ESMARTSHUTTLE® [6] 14
Figura 9 - sistema ESMARTSHUTTLE® modificato 17
Figura 10 - schema del modello auto generato tramite codice 18
Figura 11 - modulo logico di inizializzazione dati 22
Figura 12 - sotto-modulo di creazione pallet iniziali 23
Figura 13 – logica di gestione degli ordini 24
Figura 14 - sub-routine di prima movimentazione 28
Figura 15 - sub-routine di picking 29
Figura 16 - modulo logico di rifornimento 31
Figura 17 - configurazione singolo corridoio 33
Figura 18 - lista degli ordini benchmark 33
Figura 19 - andamento di lead time e throughput 36
Figura 20 - andamento degli indicatori al variare del numero di navette 38
Figura 21 - distribuzione residuo regressione lineare lungo dataset di test 40
Figura 22 - risultato studio normalità del residuo regressione lineare 41
Figura 23 - confronto distribuzione residuo lungo dataset di test 42
Figura 24 - risultato studio normalità del residuo regressione polinomiale secondo grado 43
Figura 25 - confronto errore assoluto 44
Figura 26 - confronto stime del dataset di test 44

1. ABSTRACT

La logistica è un campo nel quale la competizione tra aziende è cresciuta molto nell'ultimo periodo. Questa evoluzione ha portato a un aumento della qualità del servizio offerto al cliente ma anche a un aumento della complessità dei sistemi di immagazzinamento. I magazzini automatizzati ricoprono un ruolo sempre più fondamentale nella gestione di inventari con un grande numero di prodotti differenti e di sistemi di ordinazione sempre più efficienti e complessi.

Da questa complessità dei sistemi nasce l'esigenza di munirsi di strumenti capaci di simulare le varie soluzioni disponibili in modo affidabile, soprattutto per quei sistemi che non dispongono ancora di modelli matematici in grado di descriverli in modo accettabile.

In questa tesi viene discussa la modellizzazione ad eventi discreti di un magazzino automatico basato sul sistema ESMARTSHUTTLE®. Sono state proposte alcune modifiche al sistema attualmente in commercio in modo da poter creare un magazzino capace di effettuare operazioni di *order picking* internamente senza la necessità di creare un'area di *picking* esterna. Il sistema è stato inoltre sviluppato in modo da gestire anche lo *storage* e il *retrieval* di pallet interi parallelamente alle attività di *picking*.

Il modello è stato sviluppato tramite l'ambiente di simulazione Flexsim®. Il modello è completamente auto generante sulla base di alcuni parametri chiave, in questo modo si ha la massima compatibilità con la funzione *experimenter* di Flexsim® che permette il lancio in parallelo di simulazioni su configurazioni differenti.

Una volta sviluppato il modello e implementate le logiche di lavoro si è effettuata una valutazione sulle performance del sistema. Si sono confrontate le prestazioni del modello sviluppato con un benchmark reperito in letteratura per poi passare a un'analisi di sensitività utile per individuare il numero ideale di sistemi ESMARTSHUTTLE® da installare.

2. INTRODUZIONE AGLI ARGOMENTI TRATTATI

2.1. LOGISTICA: DEFINIZIONE ED EVOLUZIONE

Il concetto di logistica ha origini antiche radicate nella storia dell'organizzazione e dell'arte militare. Il termine *logistique* venne coniato nel 1838 dall'ufficiale militare e storico franco svizzero Antoine-Henri Jomini che lo definì come l'arte di ordinare un esercito, di combinare bene l'ordine delle truppe in colonne, gli orari della loro partenza, il loro itinerario e i mezzi di comunicazione necessari per assicurare il loro arrivo nel punto prestabilito. [1]

Bisognerà aspettare fino agli anni '60 per vedere applicati i concetti di logistica al campo industriale. Inizialmente le aziende focalizzavano le loro analisi logistiche solamente alla distribuzione e all'immagazzinamento.

La successiva evoluzione della logistica si ebbe dagli anni '80 quando si cominciarono ad eseguire studi di ottimizzazione in modo da poter offrire prodotti di qualità a costi contenuti. Si cominciò a vedere come obiettivo della logistica quello di assicurare un supporto ottimale alle attività di produzione e vendita al costo totale minore possibile. Complici di questa evoluzione furono la crescente competitività tra aziende e la diffusione sempre più capillare di logiche gestionali rivoluzionarie quali il *material requirements planning* e l'approccio *just in time*.

Attualmente si potrebbe definire la logistica come l'insieme delle attività che governano i flussi fisici del valore aggiunto, il flusso informativo sui fabbisogni e i flussi relativi ai sistemi di controllo della produzione. Nel mondo industriale odierno risulta fondamentale avere una completa integrazione dello studio logistico su tutta la *supply chain* sia interna che esterna. L'approccio attualmente più diffuso è quello di creare una competizione tra i sistemi logistici, sia esistenti che in sviluppo, in modo da individuare il sistema ottimale per ogni specifica applicazione.

2.2. SISTEMI DI IMMAGAZZINAMENTO AUTOMATIZZATI E PICKING

La notevole crescita delle grandi compagnie di *e-commerce* ha reso i sistemi di immagazzinamento degli ottimi candidati per i processi di automazione. Una completa automazione del magazzino comporterebbe per queste aziende un vantaggio su molteplici

fronti: sarebbe infatti estremamente complicato gestire la quantità sempre crescente di ordini giornalieri e di oggetti unici utilizzando un magazzino classico; si ha poi che il lavoro in un magazzino tradizionale è ripetitivo, generalmente poco ergonomico ed è difficile garantire delle condizioni di lavoro accettabili mantenendo competitivi i costi di magazzino. Queste condizioni favorirono la diffusione prima degli *Automated Storage and Retrieval Systems* AS/RS, poi degli *Autonomous Vehicle–based Storage and Retrieval Systems* AVS/RS e infine nei sistemi di ultima generazione basati su *Automated Guided Vehicles* AGV e dei sistemi che cercano di automatizzare anche la fase di *picking*.[2]

La tendenza attuale è infatti quella di studiare un metodo per integrare il sistema di *retrieval* con un robot capace di eseguire il *picking* (anche internamente al magazzino) e con dei sistemi robotici automatizzati di *packaging*. In questo modo ogni ordine potrebbe essere gestito interamente dal sistema automatizzato. Inoltre, unendo questo tipo di magazzini con i sistemi di *delivery* automatizzati attualmente in studio, si potrebbe convergere in futuro ad una condizione di automatizzazione dell'intera *supply chain*.[3]

Il *picking* è la rottura di un'unità di carico in modo da eseguire un prelievo parziale del suo contenuto al fine di ricostruire raggruppamenti di materiali che soddisfino ordini di dimensioni ridotte o di prodotti misti. I materiali soggetti a *picking* dovrebbero soddisfare alcuni prerequisiti per rendere l'operazione facilmente attuabile: i colli dovrebbero infatti essere rigidi, facilmente conteggiabili e di dimensioni che permettano la movimentabilità, essi inoltre non dovrebbero essere imballati se non singolarmente o in piccole quantità.

Le logiche di *picking* possono essere divise in due famiglie principali: logiche di *order picking* e logiche di *batch picking*. Nell'*order picking* i materiali raccolti verranno direttamente inoltrati alla confezione finale mentre nel *batch picking* si esegue l'evasione di un lotto di ordini che verranno poi formati successivamente in un'area di *sorting*.

Il *picking* può essere svolto manualmente con diverse modalità a seconda della conformazione del magazzino e dei colli trattati (operatore verso materiali, materiali verso operatore, *pick and sort*, *pick to box*, *pick to light*) o in modo automatico tramite i sistemi indicati nel seguente schema riassuntivo.

Figura 1 – schema riassuntivo dei magazzini automatizzati

2.2.1. Magazzini automatici con trasloelevatori o carrelli a guida autonoma

I più diffusi sistemi AS/RS sono i magazzini autoportanti con trasloelevatore. Questi sistemi vennero introdotti già negli anni '60 ma si sono evoluti nel tempo con un aumento delle prestazioni e del grado di automazione.[2] Questi sistemi si sono dimostrati particolarmente interessanti specialmente per la possibilità di realizzare magazzini aventi altezze sotto filo catena superiori a 12-15m e di poter eseguire contemporaneamente le traslazioni lungo i due assi. Il sistema più semplice realizzabile è quello a semplice profondità in cui viene disposto un solo pallet per *slot*. Per realizzare sistemi a doppia profondità è necessario l'utilizzo di trasloelevatori con forche retrattili più lunghe e resistenti. Per i magazzini per i quali si cercano elevatissimi coefficienti di sfruttamento superficiale si può creare un sistema multi profondità nel quale occorre però uno scaffale *push back* o uno *shuttle* di supporto per avvicinare l'unità di carico al lato corridoio in modo che il trasloelevatore possa raggiungerla senza problemi. La variante di questi sistemi per unità di

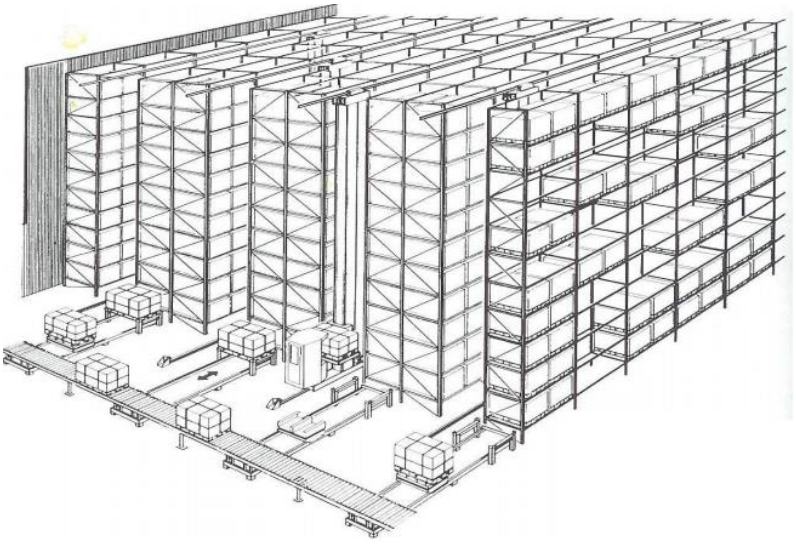

Figura 2 - sistema AS/RS con trasloelevatore [4]

carico di piccole dimensioni viene chiamata *miniload* e permette di stoccare oggetti già rimossi dai loro pallet di origine. Questi sistemi possono essere corredati di un sistema automatizzato per lo scarico delle unità di carico ritirate dal trasloelevatore su un sistema di movimentazione che le dirige verso l'area di picking creando così un sistema completamente automatico.

In alternativa si può pensare di utilizzare un sistema AVS/RS che funziona in modo del tutto similare ma nel quale si utilizzano dei carrelli elevatori a guida autonoma che operano in un magazzino tradizionale.

2.2.2. Magazzini automatici verticali, orizzontali e dispenser

In un magazzino automatico verticale si ha generalmente un guscio chiuso che contiene una struttura che permette la traslazione verticale di vari ripiani. Su ogni ripiano vengono posizionati i contenitori dai quali effettuare il *picking*. All'interno del magazzino sono contenuti anche sistemi automatici che possono estrarre o inserire i singoli contenitori da e sulla baia dalla quale avverrà il deposito o il prelievo. Questi sistemi possono essere affiancati da robot che svolgono le attività di *picking* o di *kitting* in modo da avere una completa automazione del processo. I principali vantaggi di questi sistemi sono la riduzione dell'impronta a terra con uno sfruttamento ottimizzato dell'altezza disponibile nel magazzino e la flessibilità derivante dalla possibilità di cambiare la disposizione interna dei contenitori a seconda della merce immagazzinata e delle condizioni di prelievo. Uno dei principali problemi di questi sistemi è l'impossibilità di inserire nel sistema pezzi di dimensioni troppo elevate.

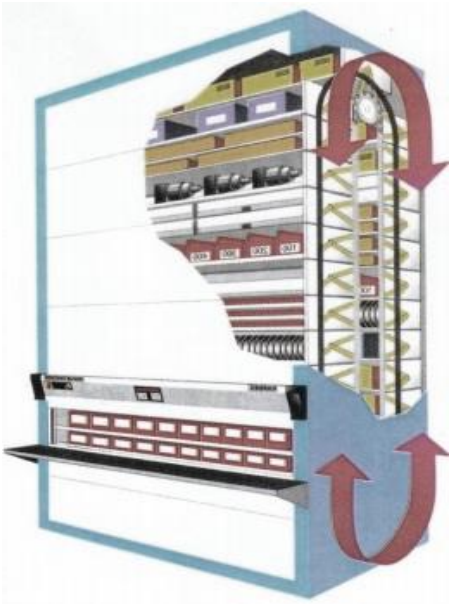

Figura 4 - magazzino automatico verticale

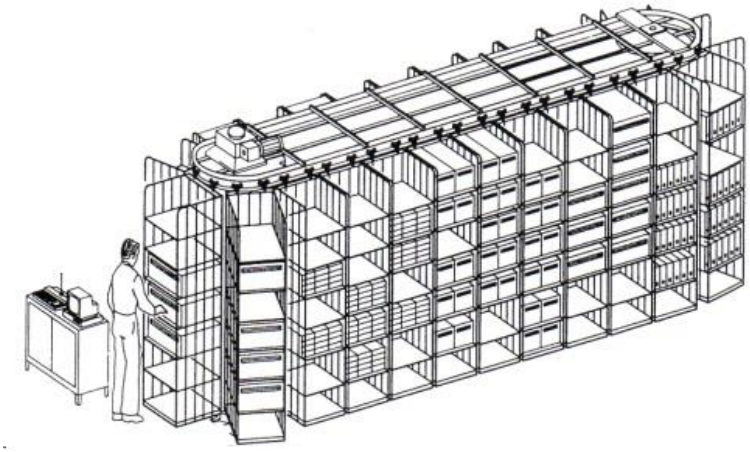

Figura 3 - magazzino automatico orizzontale [4]

In un magazzino automatico orizzontale a carosello si hanno una serie di contenitori fissati su un sistema di movimentazione che può ruotare lungo un *loop* chiuso. Il sistema si occuperà di portare in prossimità dell'area di *picking* tutti i contenitori in cui si trovano gli oggetti utili all'operazione. In questo sistema sono tutte le baie interessate a muoversi contemporaneamente verso l'area di picking riducendo notevolmente i tempi di spostamento che si avrebbero con scaffalature tradizionali. I principali vantaggi ottenibili usando questa tecnologia sono quindi una riduzione dei tempi medi di completamento di un ordine e la flessibilità dovuta alla possibilità di spostare a piacimento i contenitori lungo il sistema di movimentazione a seconda delle esigenze. Naturalmente i pezzi inseriti in questo tipo di sistemi devono essere già stati precedentemente rimossi dai pallet di origine e non devono essere di dimensioni eccessive.

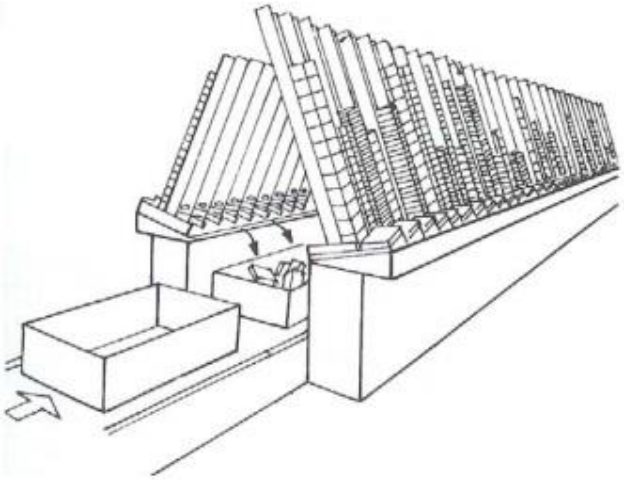

Figura 5 - sistema automatico basato su dispenser [4]

In un sistema automatico basato su *dispenser* i colli vengono caricati su delle rastrelliere per lo stoccaggio temporaneo; al momento dell'ordine l'impianto provoca la caduta di tutti i colli necessari che atterrano su un nastro trasportatore il quale li indirizza verso la zona di *picking*. Anche in questo caso i prodotti inseriti nelle rastrelliere devono essere leggeri e dalle dimensioni particolarmente contenute.

Un'evoluzione di questi sistemi è rappresentato dai *Robot-based Compact Storage and Retrieval System* RCSRS. Questi sistemi sono basati su griglie nelle quali i prodotti vengono incolonnati all'interno di box appositi. Saranno quindi dei piccoli robot ad occuparsi del ritiro e del deposito dall'alto di questi box muovendosi orizzontalmente su tutta la superficie della griglia.

Figura 6 - RCSRS prodotto da AutoStore ™

2.2.3. Magazzini automatici basati sull'utilizzo di AGV

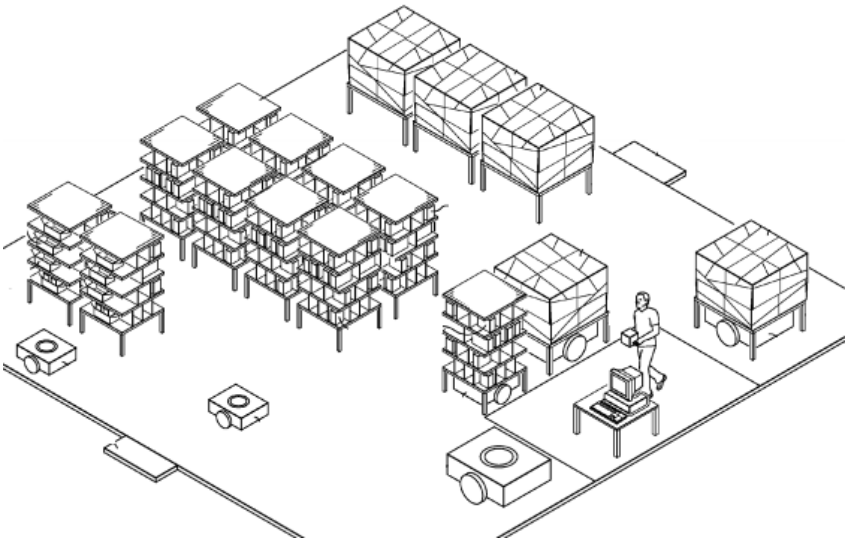

Figura 7 - magazzino automatico con RMFS [5]

Uno dei problemi presenti nei magazzini dei grandi *e-commerce* è il vasto assortimento di prodotti differenti; in un sistema di *picking* manuale, nonostante lo studio approfondito del posizionamento degli oggetti in magazzino, gli operatori si trovano costretti a fare numerosi e lunghi spostamenti tra i corridoi; questa condizione comporta problemi sia di carattere ergonomico che a livello di ottimizzazione in quanto questi eccessivi tempi di viaggio in magazzino non hanno nessun valore aggiunto intrinseco e rappresentano quindi uno spreco. Per far fronte a questa situazione sono stati sviluppati dei *Robotic Mobile Fulfillment System* RMFS; questi sistemi sono composti da AGV e scaffalature mobili contenenti i prodotti. Saranno quindi gli AGV a spostare le scaffalature contenenti i prodotti interessati verso la zona di *picking* in cui un operatore potrà svolgere l'operazione in piena ergonomia. [2]

In alternativa a questo sistema si può pensare di utilizzare un AGV in grado di muoversi in un magazzino per supportare l'operatore durante i viaggi tra i corridoi o per eseguire il *picking* in autonomia direttamente dagli scaffali. Sono stati infatti studiati negli ultimi anni diversi sistemi in grado di eseguire le operazioni di *picking* in modo autonomo direttamente in magazzino; questi sistemi sono in grado di trovare il prodotto grazie a sistemi di riconoscimento e di geolocalizzazione; sono inoltre forniti di bracci robotici per il recupero di oggetti singoli o di sistemi per il recupero di box standardizzati. [3]

2.2.4. Magazzini automatici basati sull'utilizzo di navette e analisi del sistema navetta-satellite ESMARTSHUTTLE®

L'utilizzo di magazzini automatici basati su trasloelevatori comporta alcuni problemi di ottimizzazione: ogni corsia può infatti essere percorsa da un solo trasloelevatore alla volta

e nel caso si volesse usare un solo trasloelevatore per coprire più corridoi si avrebbero tempi di cambio corsia lunghi. Per questo motivo negli anni '90 ci fu uno sviluppo di nuove tecnologie basate sull'utilizzo di navette. Le navette sono poco ingombranti e si muovono lungo le corsie viaggiando su rotaie o verticalmente utilizzando degli ascensori appositi. Ogni navetta può muoversi liberamente verso qualsiasi *slot* del magazzino senza occupare l'intero corridoio, in questo modo si possono avere diverse navette che svolgono la loro missione su più piani dello stesso corridoio contemporaneamente.[2]

Il sistema ESMARTSHUTTLE® di Eurofork S.p.A. è caratterizzato dalla presenza di due navette indipendenti: *Eshuttle* (navetta madre) ed *Esatellite* (navetta figlia). Con questo sistema l'*Eshuttle* si muoverà lungo i corridoi e tra i diversi piani come negli impianti classici ma si aggiunge il movimento lungo tutta la lunghezza dello slot eseguibile dall'*Esatellite*; in questo modo il sistema ESMARTSHUTTLE® può essere utilizzato anche in magazzini multi profondità in quanto la navetta figlia può entrare nello slot per portare le unità di carico sulla navetta madre.

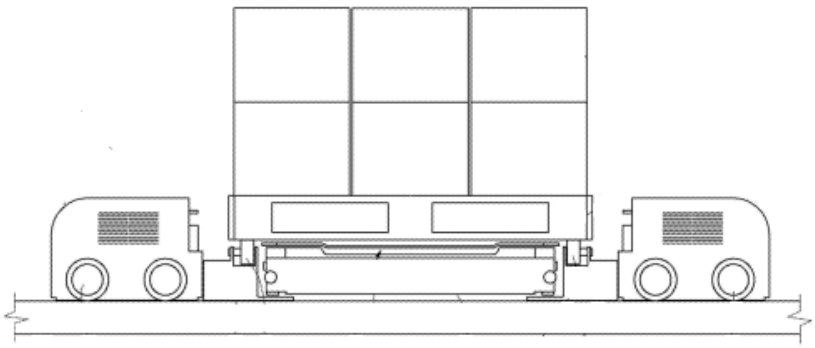

Figura 8 - sistema ESMARTSHUTTLE® [6]

Il sistema ESMARTSHUTTLE® permette la movimentazione di pallet di dimensione massima 1000x1200mm con un peso fino a 2000kg. Sia l'*Eshuttle* che l'*Esatellite* sono motorizzati e si possono quindi muovere in autonomia con velocità a carico fino a 150m/min.

2.3. SIMULAZIONE AD EVENTI DISCRETI

La simulazione è uno strumento sempre più diffuso nelle aziende produttive; complice di tale diffusione è la crescente semplicità nel reperire *hardware* alto prestazionale e lo sviluppo delle reti di calcolatori.

In una simulazione viene creato un modello del processo che si ha intenzione di studiare e si analizza il suo comportamento fornendogli *input* opportuni. In questo modo si possono analizzare diverse opzioni senza la necessità di possedere fisicamente le componenti del processo in analisi.

In un sistema dinamico a eventi discreti si ha un susseguirsi di tutti gli eventi in ordine cronologico. Questo tipo di approccio simulativo venne sviluppato negli anni '60 e si basa su tre componenti fondamentali: le entità, lo schema logico e le risorse. Il simulatore contiene anche un cronometro interno che parte nell'istante dell'inizio della simulazione, ogni evento ha un istante di partenza prestabilito che, quando viene raggiunto dal cronometro, permette l'inizio dello svolgimento dell'evento stesso. Un evento può essere un'azione svolta da una risorsa, un aggiornamento di una variabile, la creazione di un oggetto o una qualsiasi variazione dello stato attuale del sistema. In definitiva si può vedere una simulazione ad eventi discreti come una serie di entità e risorse che, al passare del tempo, vengono spinte lungo lo schema logico innescando i vari eventi che le richiedono.[7]

2.3.1. Flexsim®

Flexsim® è il *software* di simulazione sviluppato da *Flexsim Software Products Inc*. Questo simulatore permette di creare un modello *3D* del sistema da analizzare sia tramite codice (in linguaggio *Flexscript* o *C++*) sia utilizzando una libreria *drag and drop* implementata nel *software*. Un modello *3D* risulta particolarmente efficace perché permette di visualizzare il sistema creato in tempo reale in modo da poter individuare facilmente eventuali errori e possibili migliorie effettuabili.

Una volta costruito il modello si possono implementare tutte le funzioni logiche tramite il *process flow*, si crea uno schema a blocchi che ricostruisce il processo in studio utilizzando blocchi con logiche precostruite o blocchi personalizzati funzionanti tramite codice.

Flexsim® contiene inoltre funzionalità di analisi delle prestazioni, di ottimizzazione e dà la possibilità di testare scenari multipli in parallelo tramite la funzione *experimenter*.

3. ASSUNZIONI SUL MODELLO E APPROCCIO ALLA SIMULAZIONE

In questo capitolo verranno approfonditi alcuni punti fondanti della tesi già espressi nell'abstract. Questi punti sono fortemente caratterizzanti del lavoro svolto e dell'ottica con cui il modello è stato sviluppato.

3.1. APPROCCIO FLESSIBILE ALLA CREAZIONE DEL MODELLO DI SIMULAZIONE AD EVENTI DISCRETI

In letteratura la simulazione ad eventi discreti viene spesso utilizzata per valutare magazzini di ogni tipo, nella maggior parte di queste pubblicazioni il magazzino viene però visto come un elemento fisso della simulazione; vengono infatti fatte spesso analisi al variare delle risorse o dei flussi in magazzini prestabiliti e immutabili. Come già espresso nell'abstract una delle caratteristiche fondamentali del modello trattato in questa tesi è la capacità di auto generare il magazzino a inizio simulazione a seconda di alcuni parametri. Questi parametri sono sia di carattere geometrico (indicati nel capitolo successivo) sia legati all'inizializzazione del magazzino quali:

- Il numero di pallet stoccati in magazzino a inizio simulazione;
- Il numero di prodotti in ogni pallet iniziale a inizio simulazione;
- Il numero di codici differenti stoccati in magazzino;

Questo tipo di approccio rende il modello molto flessibile e capace di simulare il medesimo processo su tutti i possibili magazzini dello stesso tipo. I principali vantaggi riscontrabili usando questo metodo sono:

- Maggior numero di parametri da poter variare durante un'analisi si sensitività per renderla più completa;
- Miglior sfruttamento della funzione *experimenter* in quanto è possibile legare una specifica configurazione di magazzino a ogni esperimento lanciato in parallelo;
- Riutilizzabilità del modello per studi su sistemi simili con il relativo risparmio di tempo;
- Facilità del ritocco del modello in corso d'opera nel caso in cui ci fosse la necessità di variare alcune caratteristiche costruttive del magazzino;

3.2. MODIFICHE NECESSARIE PROPOSTE AL SISTEMA ESMARTSHUTTLE®

Come già espresso nell'abstract lo scopo del modello è quello di simulare delle operazioni di *order picking* gestito internamente al magazzino; per rendere possibile questo punto è necessario apportare alcune modifiche teoriche al sistema ESMARTSHUTTLE® attualmente in commercio.

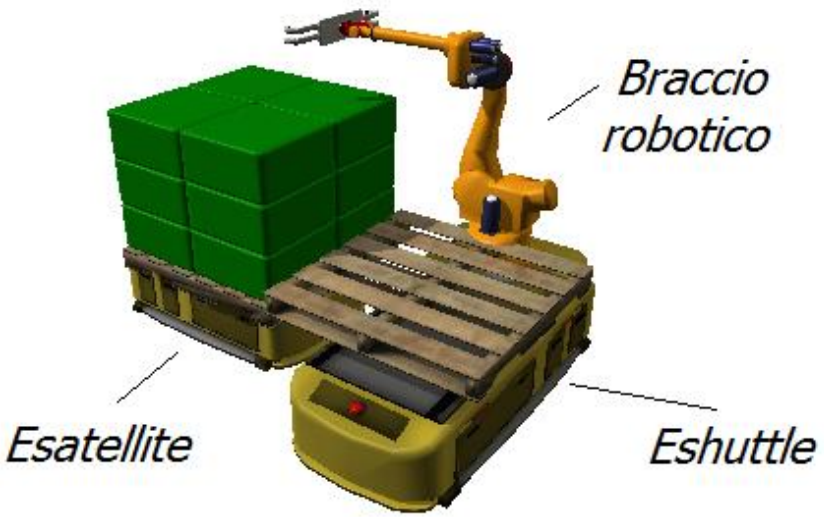

Figura 9 - sistema ESMARTSHUTTLE® modificato

La modifica più importante è quella di inserire un braccio robotico capace di svolgere le operazioni di *picking* direttamente sulla navetta *Eshuttle*. Questo braccio robotico deve essere in grado di raggiungere il pallet immagazzinato, portato sul fronte del magazzino dalla navetta *Esatellite*, in modo da spostare i pezzi desiderati sul pallet da completare precedentemente posizionato sulla navetta madre.

Un'ulteriore modifica necessaria per l'ottimizzazione del sistema e la creazione di uno spazio apposito sulla navetta *Eshuttle* per il posizionamento di pallet vuoti. Durante le operazioni di *picking* interno al magazzino può infatti succedere di svuotare completamente un pallet immagazzinato; per evitare lunghe operazioni specifiche di *retrieval* di pallet vuoti, che comporterebbero grossi sprechi di tempo e renderebbero alcuni slot momentaneamente inutilizzabili, è interessante pensare di predisporre una zona, magari nella parte posteriore della navetta madre, dove il braccio robotico possa posizionare momentaneamente il pallet vuoto per poi scaricarlo in concomitanza della consegna del pallet misto.

Entrambe queste modifiche proposte al sistema ESMARTSHUTTLE® sono implementate nel modello in modo da poter effettuare lo studio del sistema applicato alle operazioni di *picking* interno.

4. COSTRUZIONE DEL MODELLO

Per la costruzione del modello si usa un programma scritto in linguaggio *Flexscript*. Il modello viene generato e scalato a seconda dei seguenti parametri:

- Numero dei corridoi della scaffalatura;
- Numero dei livelli della scaffalatura;
- Numero delle baie di ogni livello della scaffalatura;
- Numero degli slot di ogni baia della scaffalatura;
- Larghezza baia;
- Profondità baia;
- Altezza livello;
- Larghezza corridoi tra le scaffalature;
- Coordinate posizione in ingresso del punto di accesso del primo corridoio;
- Numero di ESMARTSHUTTLE®;
- Distanza unitaria per la costruzione della zona dei parcheggi;

Per favorire l'utilizzo della funzione *experimenter* il programma viene scritto sul *trigger* del *reset* del modello. In questo modo quando si lanceranno le simulazioni in parallelo ognuna di esse eseguirà il *reset* che comporterà la generazione del modello specifico per quella stessa simulazione. Sempre per favorire l'utilizzo della funzione *experimenter* verrà incluso in testa al programma un ciclo di cancellazione di tutte le parti del modello precedentemente generato.

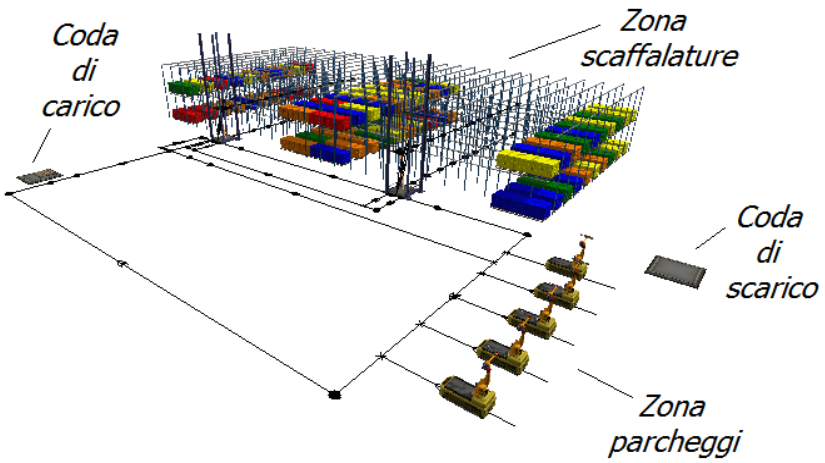

Figura 10 - schema del modello auto generato tramite codice

Il codice completo relativo alla costruzione del modello è visualizzabile nell'allegato 1.

4.1. CREAZIONE DELLA ZONA DELLE SCAFFALATURE

Per ogni corridoio vengono generate due scaffalature *drive-in* con il lato di ingresso rivolto verso il corridoio. Ad ognuna di queste scaffalature viene assegnato un sistema di indirizzi in grado di far rintracciare al simulatore il singolo *slot* tramite un codice numerico nel formato 1-1-01-1-1 (corridoio-lato-baia-livello-slot). Per ogni scaffalatura viene indicato l'ordine con cui numerare e riempire gli *slot*. Ogni scaffalatura viene quindi spostata nella zona corretta.

Per ogni corridoio viene generata una rotaia *two-way* per ogni livello che permette alla navetta di percorrere il lato di accesso delle scaffalature. Per ognuna di queste rotaie vengono inseriti i *control point* che permettono alla navetta di fermarsi in prossimità di ogni baia e i *control point* che permettono di accedere agli ascensori.

In testa ad ogni corridoio viene inserito un ascensore che permette di accedere ai livelli superiori. Gli ascensori vengono gestiti tramite uno schema logico pre-implementato in Flexsim® che evita i blocchi dell'ascensore dando le priorità alle varie navette a seconda dell'istante di prenotazione dell'ascensore stesso. I *control point* legati al sistema di ascensori verranno indicati e connessi all'ascensore di riferimento tramite codice.

4.2. CREAZIONE DELLA ZONA PARCHEGGI

La parte principale della zona parcheggi è quella comune a tutti gli ESMARTSHUTTLE®. Questa zona è composta da un grande percorso ad anello che mette in comunicazione i singoli parcheggi, l'accesso alle code di ritiro e deposito e l'accesso agli ascensori (e quindi alle scaffalature). In concomitanza dei vari ingressi ai corridoi sono inseriti dei percorsi ad anello più piccoli che permettono un movimento più agile tra le scaffalature durante le operazioni di *picking*. Questi anelli sono completati da diversi *control point* utili per rendere più semplice il controllo della simulazione e possibile l'accesso agli ascensori.

In prossimità della zona delle scaffalature viene creata una zona di *buffer* in cui le navette possono aspettare la liberazione del corridoio a cui devono accedere senza intasare il traffico.

Vengono create due code per caricare o scaricare i pallet. Nella coda di arrivo vengono generati i pallet vuoti su cui posizionare i pezzi raccolti durante il *picking* e i pallet pieni per rifornire il materiale sulle scaffalature. Nella coda di uscita vengono scaricati i pallet interi o i pallet misti appena raccolti o formati in magazzino.

L'ultima parte relativa alla zona dei parcheggi è quella dei tratti personali di ogni ESMARTSHUTTLE®. Per ogni navetta verrà infatti generato un tratto di rotaia *two-way* per entrare e uscire dal tratto comune corredato da un *control point* in cui la navetta può posizionarsi durante i momenti di inattività.

4.3. CREAZIONE DEGLI ESMARTSHUTTLE®

Per ogni sistema ESMARTSHUTTLE® vengono creati i seguenti componenti:

- Una navetta *Eshuttle* a cui vengono assegnati un nome univoco e tutte le caratteristiche reali utili per la simulazione;
- Una navetta *Esatellite* a cui vengono assegnati un nome univoco e tutte le caratteristiche reali utili per la simulazione. La navetta figlia viene posizionata sopra la relativa navetta madre e il suo sistema di riferimento viene rimpostato per avere l'origine in questa posizione;
- Un braccio robotico posizionato sulla navetta *Eshuttle*;

Ogni sistema ESMARTSHUTTLE® viene poi legato al *control point* del relativo parcheggio in modo che a inizio simulazione ogni navetta si trovi già su rotaia posizionata nel punto più idoneo per attendere istruzioni.

5. LOGICHE DI SIMULAZIONE

In questo capitolo verranno analizzati gli schemi logici che permettono alla simulazione di procedere. Lo schema è suddiviso in tre funzioni fondamentali: inizializzazione di magazzino, gestione degli ordini e rifornimento del magazzino. I moduli logici verranno presentati nella loro versione generica alla quale verranno cambiati alcuni particolari per le varie applicazioni specifiche.

5.1. MODULO LOGICO DI INIZIALIZZAZIONE DEL MAGAZZINO

Il modulo logico di inizializzazione dati è il più semplice e serve per rendere il modello appena costruito pronto per la simulazione.

Il *token* viene generato nell'istante iniziale della simulazione e la sua creazione è il primo evento ad avvenire.

Il *token* appena generato passa poi nel sotto-modulo di assegnazione *SKU*. Questo sotto-modulo è un blocco di codice personalizzato nel quale vengono definiti quale tipologia di oggetti verrà stoccata in ogni *slot* del magazzino. Il codice contenuto nel blocco può essere scritto a proprio piacimento per poter simulare ogni volta una distribuzione differente degli oggetti in magazzino.

Il sotto-modulo successivo è quello relativo alla compilazione delle liste e dell'assegnazione dei parametri. Questo sotto-modulo è composto da un blocco di codice personalizzato nel quale tutti gli *Eshuttle*, gli *Esatellite* e i bracci robotici vengono inseriti in delle liste create in precedenza e opportunamente impostate. L'inserimento di queste risorse in delle apposite liste permetterà di richiamare ogni risorsa facilmente ed eviterà di richiamare una risorsa già impegnata in un'altra operazione. Questo blocco contiene anche il codice grazie al quale possono essere assegnate le corrette proprietà cinematiche alle navette, questa operazione è particolarmente importante per avere un comportamento del simulacro che sia il più simile possibile a quello del sistema reale.

Il sotto-modulo principale di questo blocco è quello che si occupa della creazione dei pallet che, per via della sua complessità superiore, verrà analizzato più approfonditamente di seguito.

Le ultime operazioni che svolgerà il *token*, prima di andare nel blocco di distruzione, saranno quelle relative all'azzeramento di alcuni contatori. Questa operazione viene svolta da un blocco di codice personalizzato molto semplice ma di fondamentale importanza per alcune funzioni di *background* del modello.

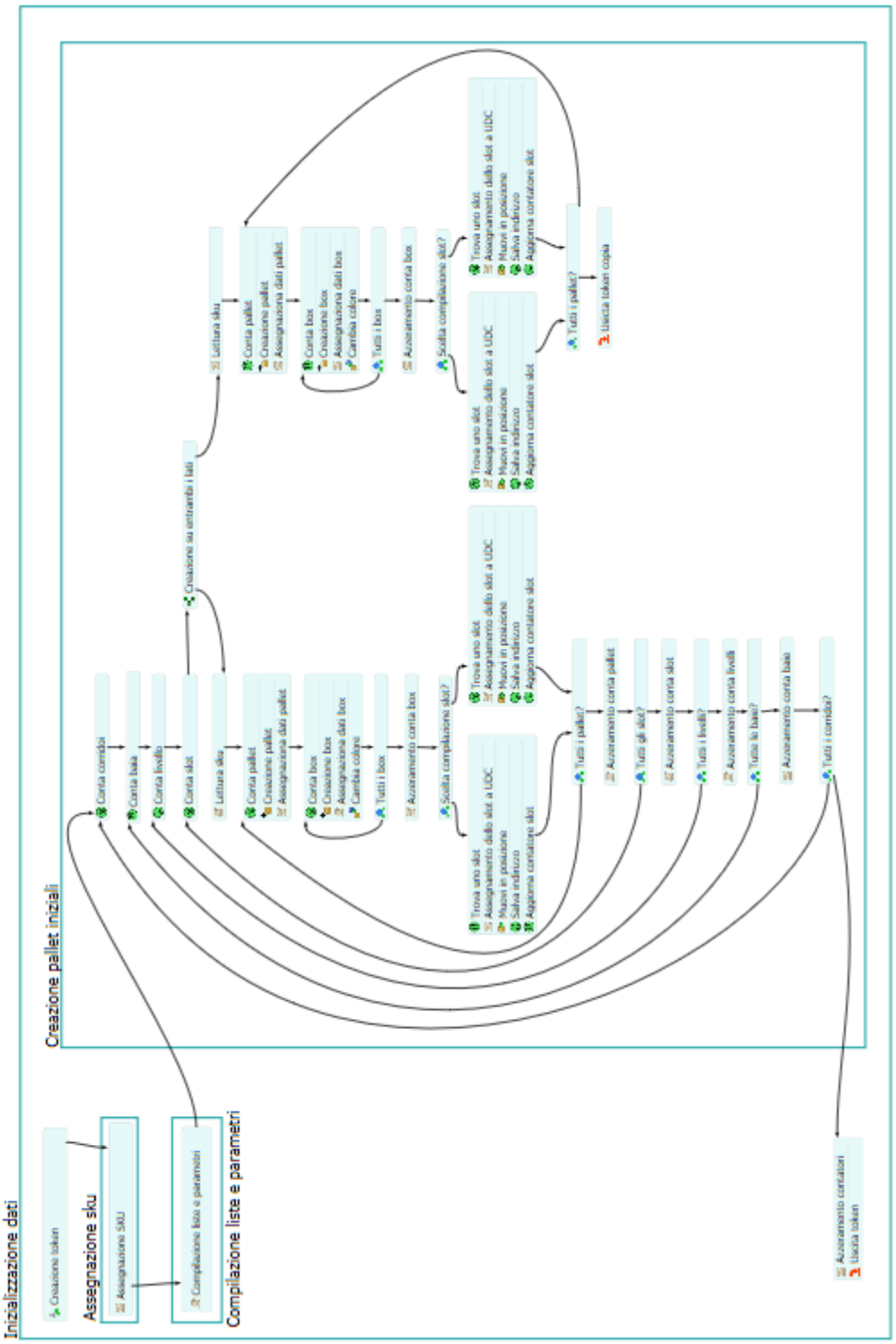

Figura 11 - modulo logico di inizializzazione dati

5.1.1. Sotto-modulo di creazione dei pallet in magazzino

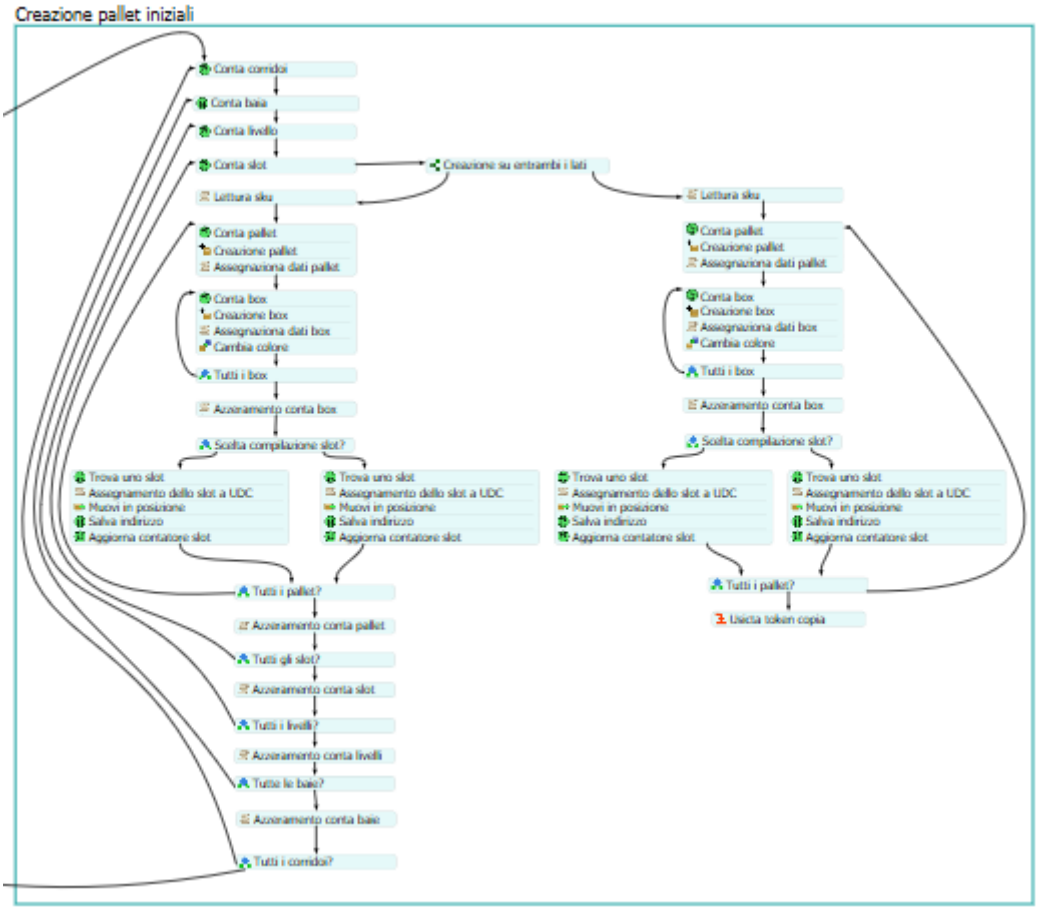

Figura 12 - sotto-modulo di creazione pallet iniziali

Il sotto-modulo è contraddistinto da un ciclo *for* nel quale vengono contati i corridoi, le baie, i livelli e gli *slot*. Per ogni conteggio il *token* viene sdoppiato in modo da poter generare i pallet su entrambi i lati del corridoio senza appesantire ulteriormente il ciclo *for*. Per ognuno dei *token* viene effettuata la lettura del relativo *slot* in modo da valutare quale oggetto creare in quella posizione. Vengono quindi creati il pallet e i relativi oggetti accatastati su di essi; tutte le entità appena create vengono poi rinominate e colorate per rendere più facile il loro riconoscimento. Lo *slot* viene poi identificato tramite lo schema di indirizzi e, con l'ausilio di un blocco di codice personalizzato, ogni pallet creato viene stoccato nel relativo *slot*.

Per semplificare le operazioni di *picking,* di *storage* e di *retrieval* su ogni pallet vengono salvate le seguenti informazioni fondamentali:

- Indirizzo nel quale il pallet è attualmente stoccato;
- Attuale contenuto del pallet;
- Ora di inserimento del pallet in magazzino;

Come si vedrà più avanti questi dati saranno letti e riscritti ogni volta che il pallet verrà modificato o spostato.

Una volta generati tutti i pallet iniziali di quello specifico *slot* i due *token* gemelli subiranno due trattamenti diversi: uno dei due verrà distrutto mentre l'altro verrà utilizzato per far avanzare il conteggio del ciclo *for* e per la generazione dei nuovi token gemelli.

5.2. MODULO LOGICO DI GESTIONE DEGLI ORDINI

Il modulo logico di gestione degli ordini è il più complesso e permette l'evasione di un qualsiasi ordine sia esso di pallet misti o di pallet interi.

Il primo sotto-modulo è quello di generazione degli ordini; questo sotto-modulo può essere modificato ogni volta a seconda delle esigenze a patto che si rispetti il metodo di scrittura degli ordini. Lo schema logico è infatti creato in modo da interpretare gli ordini forniti sotto forma di vettore contenente il numero di oggetti voluti per ogni tipologia di oggetto; il vettore sarà quindi lungo quanto il numero di tipi di oggetto immagazzinati e per ogni indice del vettore sarà indicato il numero di oggetti che bisognerà caricare per quello specifico tipo di prodotto.

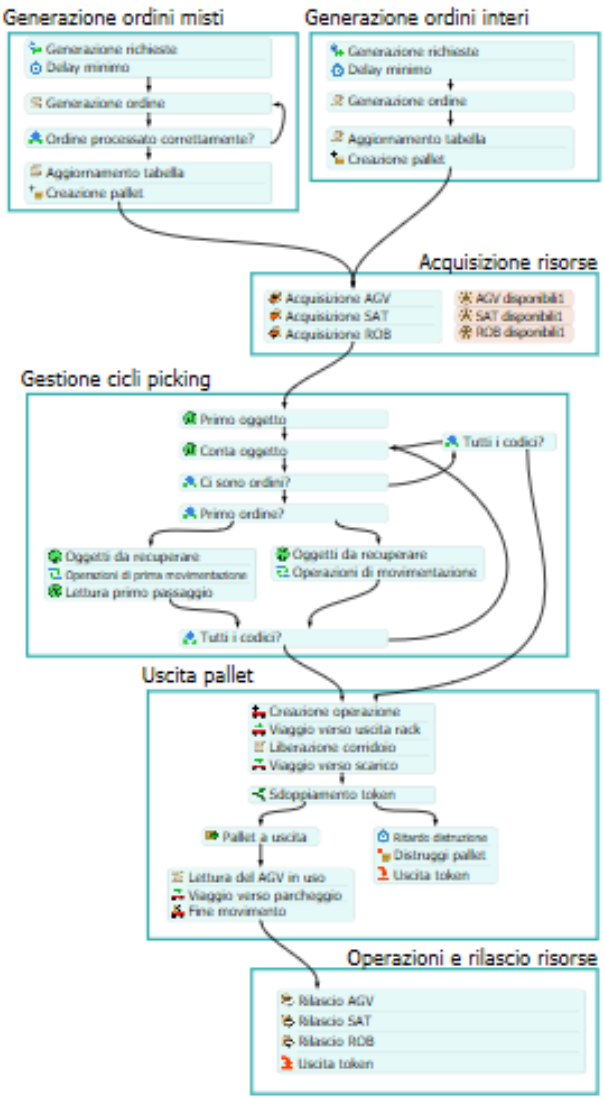

Figura 13 – logica di gestione degli ordini

Una volta ricevuto l'ordine si passa all'acquisizione delle risorse; in questo sotto-modulo verranno sottratti dalle liste,

precedentemente generate e riempite, tutte le risorse necessarie per il completamento dell'ordine. Verrà quini sottratto un *Eshuttle* con relativo satellite e braccio robotico.

Si passerà quindi al sotto-modulo che si occupa della gestione dei cicli di *picking*. Questo sotto-modulo è un ciclo *for* che si assicurerà che tutti i vari tipi di oggetti vengano recuperati saltando quelle tipologie di oggetti per le quali è stata segnata un'ordinazione di zero unità. Nel caso del primo tipo di oggetto da raccogliere il *token* verrà indirizzato alla *sub-routine* relativa, in tutti gli altri casi il *token* sarà invece dirottato verso una *sub-routine* di movimentazione più semplice spogliata di tutte le operazioni di inizio *picking*.

Una volta finite le operazioni di *picking* il token passerà attraverso il sotto-modulo nel quale la navetta con l'ordine completo verrà direzionata verso l'uscita del *rack* e quindi verso la coda di scarico dove completerà la sua missione. Una volta portata a termine l'operazione, la navetta ritornerà al suo parcheggio e tutte le risorse verranno reinserite nelle relative liste pronte per essere richiamate.

5.2.1. Sub-routine di movimentazione

Nel caso della *sub-routine* di prima movimentazione la prima operazione è il viaggio verso la coda di carico; una volta raggiunta la destinazione partiranno le operazioni che vedono la navetta *Esatellite* caricare il pallet vuoto per poi scaricarlo sulla navetta *Eshuttle*.

Completate le operazioni di carico il *token* innescherà l'operazione di viaggio in direzione della zona di *buffer*. Raggiunto lo snodo che porta al buffer verrà letto il tipo di materiale da recuperare e verrà individuato lo *slot* da raggiungere tra quelli che contengono quella tipologia di pezzi.

Per evitare blocchi dell'impianto, il sistema è gestito mediante un sistema di prenotazioni; una navetta che intende interagire con uno slot dovrà infatti prenotare sia il singolo slot che l'intero livello del corridoio; in questo modo si evitano tutte quelle situazioni nelle quali due navette sono in missione per recuperare il medesimo oggetto e le situazioni in cui due navette contromano bloccano l'intero flusso delle operazioni.

Nel momento di ricerca di uno *slot* si seguirà il seguente sistema di priorità:

1. Se possibile si sceglie uno *slot* raggiungibile tramite un livello di corridoio non attualmente prenotato da altre navette, se ci sono più *slot* di questo tipo viene scelto lo *slot* che contiene attualmente più pezzi;
2. Se non esistono *slot* attualmente raggiungibili viene scelto lo slot che contiene attualmente più pezzi tra gli *slot* occupati;
3. Se non esistono *slot* che contengono pezzi di quella tipologia la navetta si sposta momentaneamente fuori dal flusso delle operazioni aspettando la ricarica del magazzino, questo avviene tramite un *loop* con il quale si controlla periodicamente se quella tipologia di merce è stata reimmessa in magazzino;

Una volta trovato uno *slot* che corrisponde alle esigenze questo viene prenotato e viene individuato lo specifico pallet con cui interagire (secondo la logica LIFO).

Viene quindi valutata l'occupazione del corridoio tramite un blocco di codice personalizzato e si hanno quindi due scenari possibili:

- Se il livello del corridoio è vuoto esso viene prenotato e la navetta viene indirizzata verso il *control point* in prossimità dello *slot* da raggiungere;
- Se il livello del corridoio è occupato la navetta viene indirizzata verso la zona di buffer dove attenderà la liberazione del corridoio grazie ad un *loop* di controllo.

Una volta raggiunto il *control point* relativo allo *slot* si procederà con le logiche di raccolta degli oggetti. Nel caso si voglia raccogliere un pallet intero e il pallet individuato sia ancora nelle sue condizioni originali si avrà una semplice operazione in cui si caricherà direttamente sulla navetta *Eshuttle* tutto il pallet senza utilizzare il braccio robotico. In tutti gli altri casi si procederà con la *sub-routine* di *picking* analizzata nel capitolo seguente.

Dopo aver effettuato il *picking* o il carico del pallet intero si avranno due scenari possibili:

- Si sono raccolti tutti i pezzi richiesti per quello specifico codice, lo *slot* verrà quindi liberato e la *sub-routine* risulterà completata potendo procedere oltre nel modulo logico di gestione degli ordini;
- I pezzi contenuti il quel pallet non erano sufficienti e occorrerà trovarne un altro per completare la raccolta;

Se si dovesse ricadere nella seconda casistica la ricerca del nuovo pallet seguirà questo sistema di priorità:

1. Se lo *slot* attuale contiene altri pallet verrà scelto il primo di quelli secondo la logica LIFO, questo permette il proseguire le operazioni di *picking* senza effettuare alcuno spostamento;
2. Se lo *slot* attuale non contiene altri pallet si cercherà il nuovo pallet in uno *slot* che si affaccia sul medesimo livello di corridoio, in questo modo si evitano tutte le operazioni legate alla prenotazione di un nuovo corridoio, all'utilizzo dell'ascensore e di spostamento tra un corridoio e l'altro;
3. Nel caso non fossero più presenti pallet del tipo desiderato lungo l'intero livello di corridoio si procederà con la liberazione del corridoio in uso e con una ricerca del nuovo *slot* seguendo la scala di priorità già vista per le operazioni di ricerca del primo *slot*;

Questa operazione verrà ripetuta fino a ricadere nella prima casistica completando la *sub-routine*.

La *sub-routine* per le movimentazioni successive è estremamente simile a quella appena analizzata; le differenze principali sono:

- Non vi è più la possibilità di caricare l'intero pallet direttamente sulla navetta *Eshuttle* in quanto sono già state effettuate delle operazioni di *picking* e ci saranno già dei pezzi caricati;
- La logica partirà direttamente con la scelta dello *slot* tenendo conto dell'attuale posizione in cui ci si trova; si agirà quindi in modo simile a quanto visto per la ricerca dei pallet successivi al primo della *sub-routine* di prima movimentazione;
- Sono eliminate tutte le operazioni relative alla coda di carico e alla raccolta del pallet vuoto iniziale;

Durante lo svolgimento di queste operazioni risulta fondamentale aggiornare i dati di ogni *slot* e pallet con cui si interagisce in modo da avere sotto controllo la situazione delle scorte in magazzino in tempo reale.

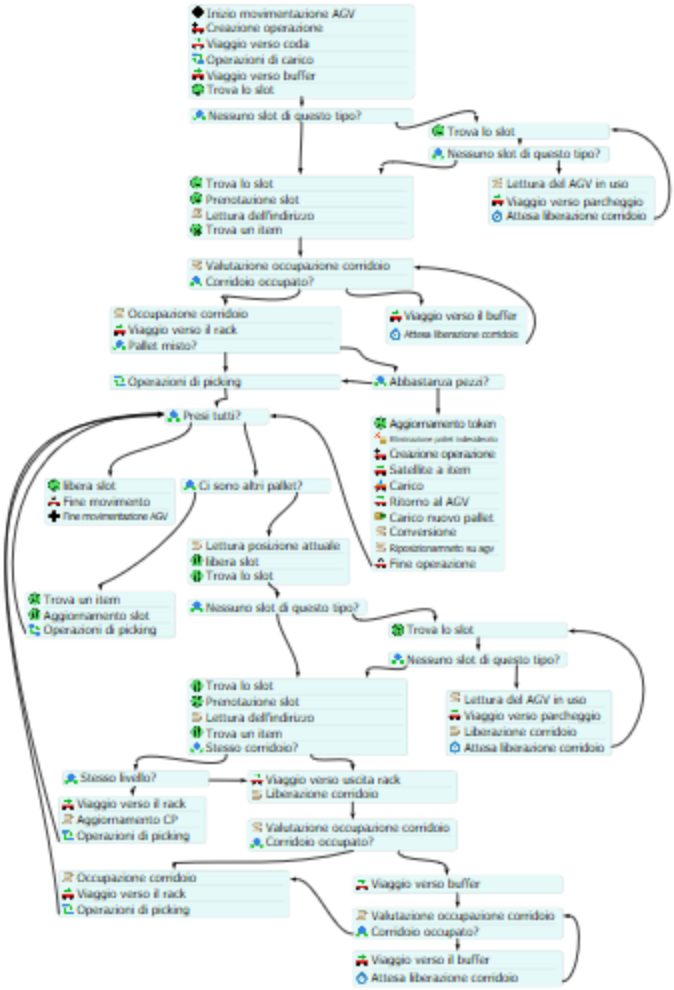

Figura 14 - sub-routine di prima movimentazione

5.2.2. Sub-routine di picking

Questa sub-routine inizia con il viaggio della navetta *Esatellite* lungo la profondità scaffalatura *drive-in* per raggiungere il pallet. Esso viene caricato e portato in prossimità della navetta *Eshuttle*.

Raggiunto questo punto ci si può trovare in tre diverse situazioni:

- Il pallet recuperato contiene più pezzi di quelli necessari; in questo caso il braccio robotico sposta i pezzi dal pallet appena recuperato al pallet misto fino al raggiungimento dell'obbiettivo; il pallet viene poi riportato alla sua posizione di origine dalla navetta *Esatellite*;

- Il pallet recuperato contiene solo i pezzi necessari o un numero minore e questo è il primo pallet con cui si interagisce; in questo caso non si ha nessuna operazione di *picking* e l'intero pallet appena recuperato viene caricato sulla navetta *Eshuttle*;
- Il pallet recuperato contiene solo i pezzi necessari o un numero minore ma questo non è il primo pallet con cui si interagisce; in questo caso il braccio robotico sposta tutti i pezzi dal pallet appena recuperato al pallet misto; il pallet vuoto viene poi preso dal braccio robotico per essere caricato nella zona apposita della navetta madre (per semplificare la logica di simulazione il pallet vuoto viene semplicemente eliminato);

Dopo ognuna di queste operazioni vi è sempre una parte di codice utile all'aggiornamento di tutti gli *slot* e i pallet con cui si è entrati in contatto.

Al termine di queste operazioni la navetta figlia ritorna in posizione sulla relativa navetta madre e si ha il termine della *sub-routine*.

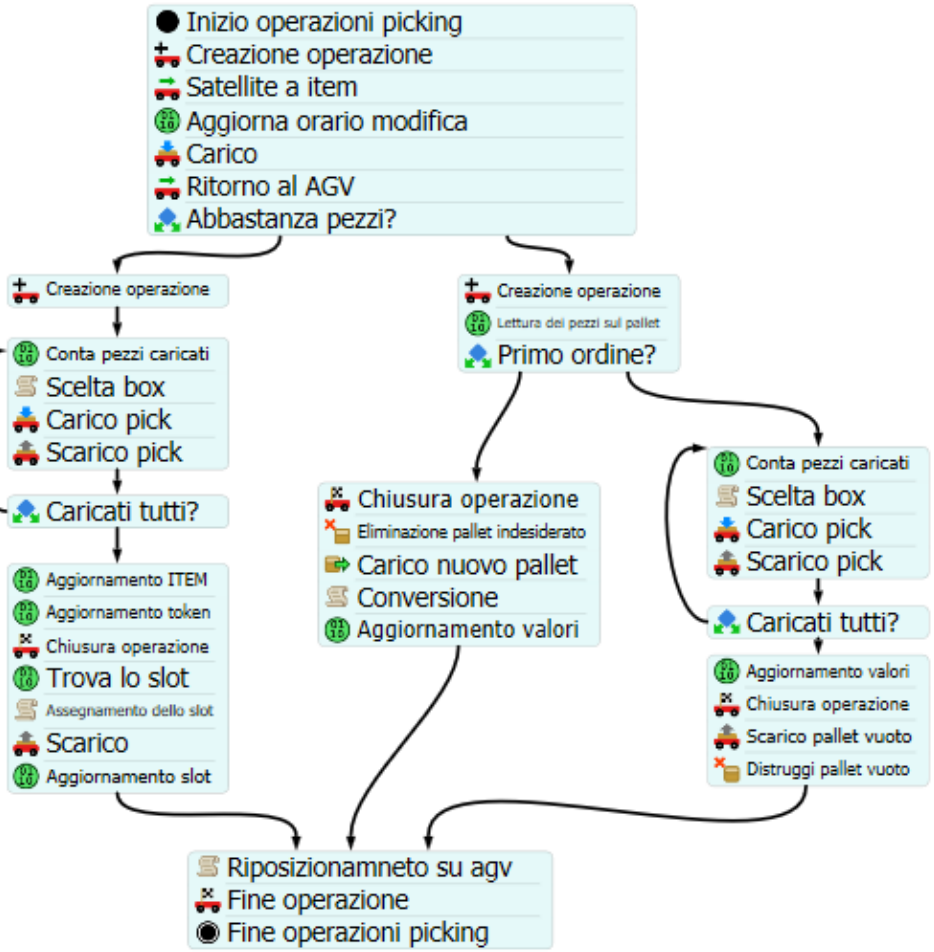

Figura 15 - sub-routine di picking

5.3. MODULO LOGICO DI RIFORNIMENTO DEL MAGAZZINO

Il modulo logico di rifornimento del magazzino regola tutte quelle attività legate alle unità di carico intere provenienti dai fornitori da inserire nel magazzino. Anche questa attività viene svolta dalle navette.

Il modulo parte con la generazione del *token* che innesca le operazioni di verifica di necessità dell'ordine. Queste operazioni sono liberamente inseribili a seconda della politica di gestione delle scorte che si vuole attuare. Nel caso in cui ci fosse la necessità di rifornimenti, si salvano nel *token* le informazioni relative alla tipologia e alla quantità di pallet da rifornire.

Il sotto-modulo successivo è quello relativo alla creazione dei pallet. In questo sotto-modulo vengono generati i pallet del tipo necessario e nella quantità utile; questi vengono poi inseriti nella coda degli arrivi pronti per essere recuperati.

Si ha quindi un sotto-modulo di acquisizione risorse molto simile a quello già visto in precedenza; in questo sotto-modulo non viene però richiamato il braccio robotico in quanto non è necessario pe questo tipo di operazione.

Si ha quindi una *sub-routine* di movimentazione nella quale si ha il sistema navetta che, come prima operazione, si muove verso la coda di ingresso; una volta raggiunta la coda si ha un'operazione di carico del pallet del tutto uguale a quella già vista in precedenza per il carico dei pallet vuoti. Si ha poi un viaggio verso la zona di *buffer* raggiunta la quale viene individuato lo *slot* da rifornire. Ci si può quindi trovare in due situazioni:

- Lo *slot* è raggiungibile; si occupa il livello di corridoio e ci si muove in direzione del relativo *control point* posto di fronte allo *slot*;
- Il livello di corridoio relativo allo *slot* è momentaneamente occupato; ci si sposta verso la zona di buffer dove viene lanciato un *loop* che controlla periodicamente la liberazione del corridoio da raggiungere.

Una volta raggiunto il *control point* si hanno le operazioni di scarico con i relativi aggiornamenti dei dati di tutti gli slot e i pallet interessati. Il nuovo pallet viene sempre posizionato in testa al corridoio in accordo con una politica LIFO. La fase di scarico termina con il riposizionamento della navetta *Esatellite* sulla propria navetta *Eshuttle*.

Una volta terminate le operazioni di scarico dei pallet si ha il movimento di uscita dal corridoio, la liberazione dello stesso e il viaggio di ritorno verso il posteggio.

Le ultime operazioni, prima della distruzione del *token*, sono quelle relative al reinserimento delle risorse nelle relative liste in modo da rendere il sistema navetta nuovamente disponibile.

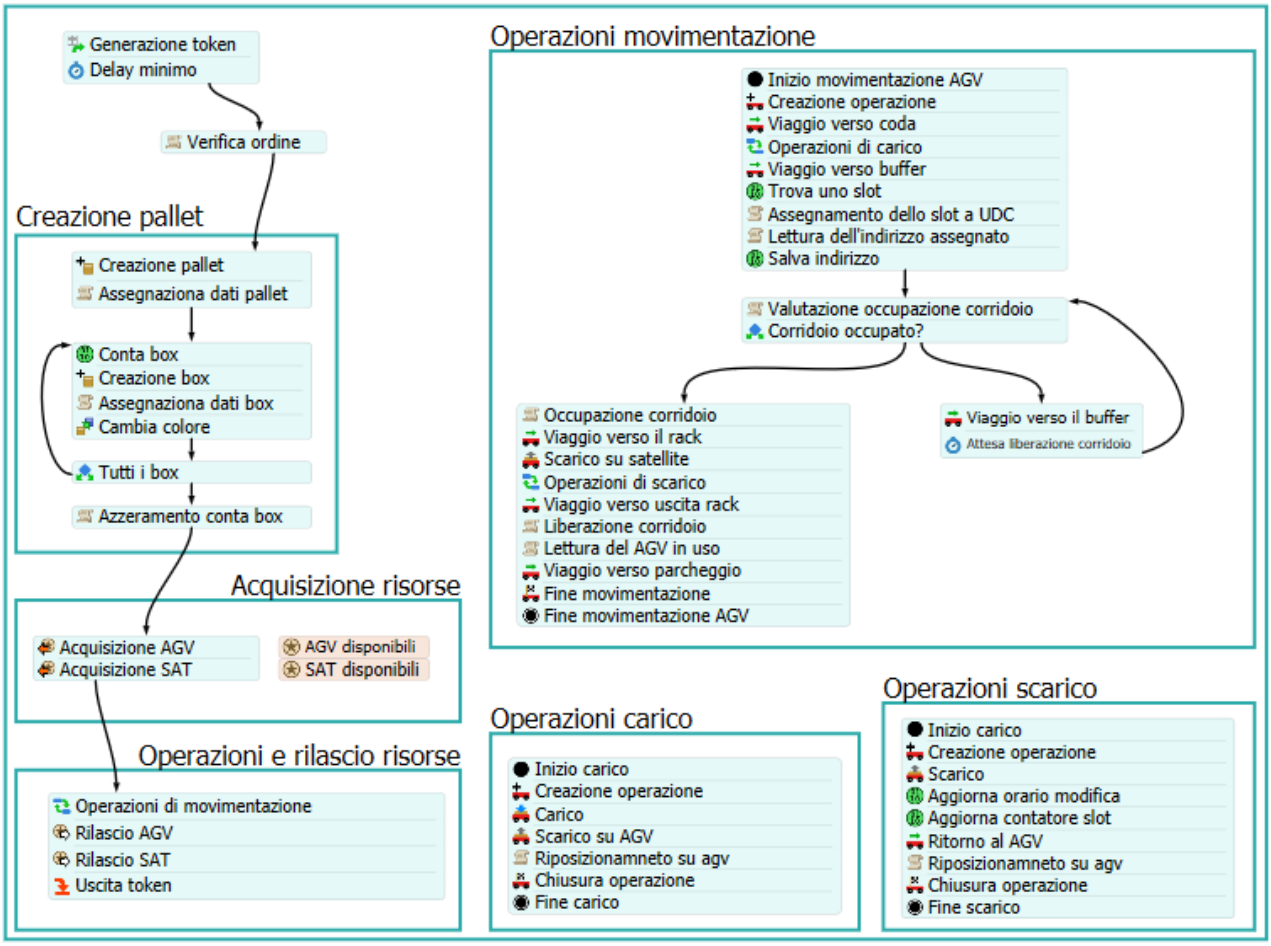

Figura 16 - modulo logico di rifornimento

6. CONFRONTO CON IL BENCHMARK

Non avendo a disposizione dati relativi a un sistema reale si è cercato in letteratura un sistema di *order picking* per cui erano state analizzate le relative *performance* in modo da poter fare un confronto tra i sistemi. Il *paper* scelto per il confronto[8] tratta il sistema di *order picking* più diffuso in Europa[9] ovvero un sistema *picker to parts*.

Nel sistema di *benchmark* l'operatore usa un carrello elevatore *stock picker Jungheinrich EKX515k* con il quale percorre un corridoio lungo 150m e composto da scaffalature alte 14,5m. Lungo il corridoio l'operatore può trovare tutti le 10 tipologie di oggetti che possono comporre un pallet misto.

Le *performance* del sistema vengono studiante variando la percentuale di tempo di inattività sul tempo totale di lavoro; per questo confronto verrà preso come *benchmark* il sistema senza tempi di blocco.

Le prestazioni del sistema *benchmark* vengono valutate sulla base di una lista di ordini predefinita. La lista è composta da una serie di vettori che esplicitano il numero di oggetti di ogni tipo che devono comporre il pallet misto da consegnare. Il sistema di *benchmark* evade questa specifica lista di ordini in 26,53h.

Il modello da testare è stato quindi impostato per eseguire la medesima lista di ordini. Non avendo informazioni in merito si sono immaginate 10 tipologie di oggetti delle stesse dimensioni che permettono la formazione di pallet interi da 100 pezzi. Essendo che tutti i pezzi hanno la stessa possibilità di essere richiesti non è stata studiata nessuna disposizione particolare dei pallet nel magazzino. Proprio come nel sistema di *benchmark* le scorte in magazzino all'istante iniziale sono state studiate in modo da avere abbastanza pezzi per rispondere a tutti gli ordini senza ricorrere alle operazioni di rifornimento.

L'analisi è stata fatta su un sistema delle stesse dimensioni di quello di *benchmark* con 75 slot larghi 2m (150m totali) e 5 livelli altri 2,9m (14,5m totali). Non avendo informazioni in merito gli SKU sono stati distribuiti casualmente per mettersi nella situazione peggiore. La simulazione è stata eseguita variando il numero di sistemi ESMARTSHUTTLE® installati.

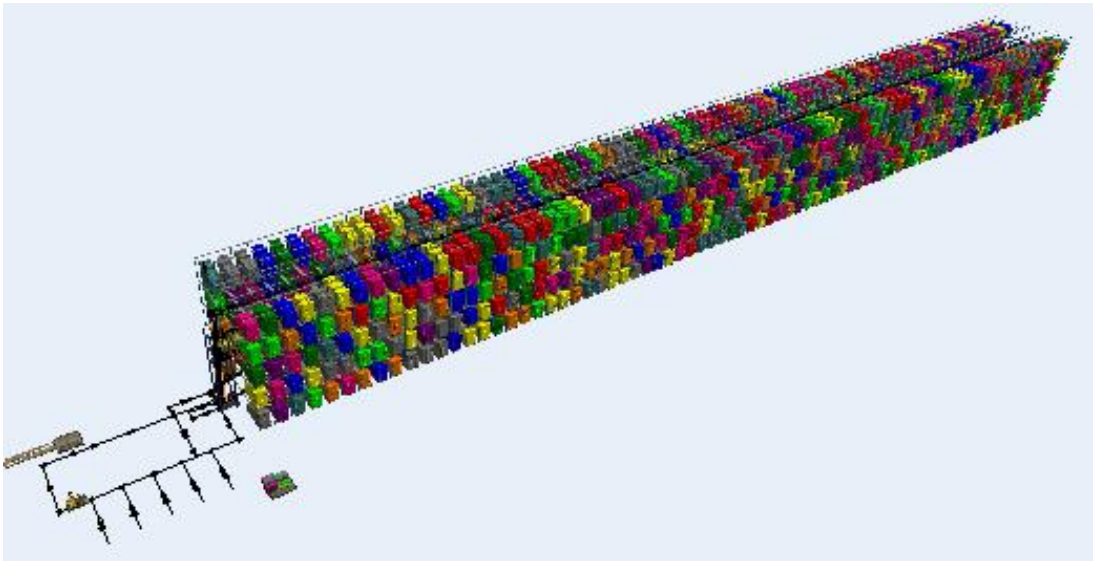

Figura 17 - configurazione singolo corridoio

j	$\{p_{j1}, p_{j2}, p_{j3}, p_{j4}, p_{j5}, p_{j6}, p_{j7}, p_{j8}, p_{j9}, p_{j10}\}$	j	$\{p_{j1}, p_{j2}, p_{j3}, p_{j4}, p_{j5}, p_{j6}, p_{j7}, p_{j8}, p_{j9}, p_{j10}\}$	j	$\{p_{j1}, p_{j2}, p_{j3}, p_{j4}, p_{j5}, p_{j6}, p_{j7}, p_{j8}, p_{j9}, p_{j10}\}$
1	{5,9,10,6,2,8,8,0,3,3}	35	{4,7,5,0,4,5,1,2,9,9}	69	{7,4,8,9,0,1,4,2,5,4}
2	{9,4,8,7,9,1,4,6,4,8}	36	{1,9,10,7,1,7,9,7,6,4}	70	{0,2,2,6,7,1,5,4,6,4}
3	{0,6,10,6,8,10,3,1,7,4}	37	{3,2,6,9,1,8,6,0,7,6}	71	{1,3,6,3,7,9,2,9,2,2}
4	{3,5,5,3,9,9,4,6,4,1}	38	{2,9,4,10,0,9,9,2,5,2}	72	{10,3,9,8,8,0,9,9,2,8}
5	{1,9,2,6,8,8,8,3,9,0}	39	{3,5,3,0,9,8,1,2,4,3}	73	{9,6,9,9,3,2,10,3,2,8}
6	{7,1,10,6,9,2,6,0,0,3}	40	{9,3,5,5,9,1,1,7,2,5}	74	{4,1,8,4,1,5,9,3,1,2}
7	{3,2,7,3,3,9,3,6,6,5}	41	{10,2,10,8,9,0,5,2,1,4}	75	{1,6,2,6,6,10,2,6,3,8}
8	{0,4,4,5,6,6,1,6,6,2}	42	{4,3,4,6,9,1,8,10,0,8}	76	{8,6,0,4,3,3,6,3,1,5}
9	{0,6,6,10,7,6,1,10,2,4}	43	{2,6,10,3,2,6,9,6,8,7}	77	{2,10,1,10,6,5,6,3,7,6}
10	{5,8,2,9,1,10,0,7,1,4}	44	{6,2,3,1,2,6,2,0,5,3}	78	{0,1,2,8,2,5,10,5,1,7}
11	{9,0,1,7,3,8,9,9,6,8}	45	{1,6,3,4,1,5,4,4,4,3}	79	{7,9,6,2,5,2,9,7,0,8}
12	{9,7,2,7,9,10,4,5,6,5}	46	{1,1,4,7,4,4,6,8,1,7}	80	{5,3,7,9,7,3,7,6,9,9}
13	{6,1,5,7,2,5,6,2,3,9}	47	{8,6,0,0,4,5,9,6,7,0}	81	{3,5,2,4,9,8,7,7,6,4}
14	{6,1,4,5,7,10,5,7,9,6}	48	{4,5,0,2,9,8,4,5,1,6}	82	{4,6,0,5,5,8,6,4,6,1}
15	{8,4,6,7,9,9,8,8,4,7}	49	{1,9,6,2,8,2,5,4,6,3}	83	{8,5,7,9,1,0,4,5,1,3}
16	{2,9,10,3,4,5,2,10,5,3}	50	{3,7,3,9,7,9,7,2,2,9}	84	{8,7,1,7,4,9,2,0,3,9}
17	{5,5,4,6,10,2,6,4,5,1}	51	{5,5,9,8,7,8,4,0,7,2}	85	{8,2,4,8,10,7,3,6,9,3}
18	{2,6,7,9,2,0,1,4,0,5}	52	{5,7,3,8,9,9,1,4,6,7}	86	{7,4,8,9,8,4,2,7,1,3}
19	{3,2,4,9,5,4,5,8,4,2}	53	{7,9,8,1,2,8,6,9,5,5}	87	{7,8,9,2,7,9,1,9,9,7}
20	{3,5,3,1,5,8,9,5,3,9}	54	{2,3,3,10,9,2,8,9,4,5}	88	{1,3,9,7,6,4,7,4,10,0}
21	{10,5,5,5,8,1,9,7,10,2}	55	{9,0,0,9,2,8,6,0,6,2}	89	{4,9,3,5,3,5,5,3,3,1}
22	{10,10,5,7,1,2,2,5,6,7}	56	{6,5,8,7,5,5,2,6,10,2}	90	{8,8,6,9,4,9,3,4,0,4}
23	{2,3,8,4,7,6,6,8,5,8}	57	{8,4,3,3,4,7,7,9,8,7}	91	{3,9,4,9,0,1,10,6,9,0}
24	{4,8,5,3,10,6,4,4,2,3}	58	{5,4,6,7,1,1,7,8,6,8}	92	{8,8,2,7,5,4,7,1,4,6}
25	{7,3,7,9,0,2,3,5,2,0}	59	{3,0,9,9,4,1,2,0,3,8}	93	{1,6,8,8,4,7,9,8,9,6}
26	{6,7,2,6,2,2,4,2,6,9}	60	{8,3,7,0,6,9,10,3,8,9}	94	{4,2,5,8,2,8,6,8,9,4}
27	{7,5,5,4,7,9,1,0,1,5}	61	{8,9,7,6,5,6,8,4,7,2}	95	{2,0,5,4,0,10,2,7,3,5}
28	{4,1,4,5,4,6,4,3,9,8}	62	{4,10,5,2,7,10,1,9,8,3}	96	{1,6,9,0,4,1,10,6,2,5}
29	{5,7,8,8,2,8,6,6,3,5}	63	{9,9,4,1,6,7,8,3,8,4}	97	{7,10,10,1,8,2,3,5,3,8}
30	{3,6,0,5,2,9,0,1,7,4}	64	{7,9,2,7,9,3,5,8,7,7}	98	{6,1,1,7,4,1,8,0,1,10}
31	{6,2,8,9,8,4,3,8,2,9}	65	{4,0,9,6,5,5,5,6,9,9}	99	{8,7,3,9,7,7,2,6,2,2}
32	{10,9,7,7,2,2,10,4,6,5}	66	{7,8,4,1,1,3,8,6,3,3}	100	{3,8,2,10,2,10,6,2,5,8}
33	{4,5,9,1,1,2,4,1,10,10}	67	{1,4,6,3,7,2,4,2,8,1}	-	-
34	{5,8,3,8,6,9,6,2,1,3}	68	{2,8,1,5,2,10,3,1,8,8}	-	-

Figura 18 - lista degli ordini benchmark

Dai risultati si può notare come un singolo sistema ESMARTSHUTTLE® possa mantenere una prestazione assimilabile a quella ottenibile con il sistema di *benchmark*. Si può notare anche come un aumento delle navette installate porti a un calo drastico dei tempi per completare la lista ordini.

numero di navette	tempo totale [h]	confronto benchmark
1	27,49	+4%
2	13,89	-48%
3	10,31	-61%
4	7,13	-73%
5	6,09	-77%

Si può quindi dire che il grande vantaggio portato dall'utilizzo degli ESMARTSHUTTLE® sia il poter utilizzare più sistemi in parallelo nello stesso spazio in cui solitamente può operare un solo *picker*. Le prestazioni possono quindi essere incrementate ulteriormente ripensando all'utilizzo degli spazzi di magazzino e applicando configurazioni multi corridoio nelle quali è possibile far lavorare ancora più sistemi in parallelo a parità di volume occupato.

7. OTTIMIZZAZIONE CON ANALISI DI SENSITIVITÀ

7.1. STUDIO DELL'INFLUENZA DEL NUMERO DI CORRIDOI E LIVELLI

Dai dati raccolti per il confronto precedente si è potuto notare come l'aumentare del numero di sistemi ESMARTSHUTTLE® installati comporti un miglioramento della prestazione. Questo aumento della prestazione verrà mitigato nel momento in cui il numero di navette supererà il numero di livelli di corridoio accessibili in contemporanea in quanto le navette in eccesso dovranno attendere nella zona di *buffer* la liberazione di un corridoio. Si può quindi definire una prima regola di progettazione del magazzino:

$$cl \geq n$$

Dove c è il numero di corridoi, l in numero di livelli e n il numero di sistemi ESMARTSHUTTLE®.

Fissata questa regola si vuole definire quale variabili agiscono significativamente sulla prestazione del sistema; il *kpi* scelto per fare queste valutazioni è il tempo totale per completare tutti gli ordini nella lista.

Per questa prova si sono andati a fissare alcuni fattori:

- Numero di sistemi ESMARTSHUTTLE® installati pari a 1;
- Lista di 100 ordini consecutivi da 10 prodotti diversi generata casualmente;
- Disposizione degli *SKU*;

Si andranno invece a valutare le seguenti variabili:

- Numero di corridoi c;
- Numero di livelli l;
- Coefficiente della legge esponenziale di interarrivo k;
- Numero di oggetti medio per ordine m;

Per l'analisi della varianza si è impostato un semplice esperimento fattoriale 2^4 utilizzando le variabili elencate in precedenza. L'analisi dell'esperimento fattoriale è stata svolta in forma completa utilizzando l'algoritmo di Yates. La tabella di risoluzione completa è riportata nell'allegato 2. Si imposta un livello di significatività del 1%.

Gli unici fattori che risultano significativi sono k, m e la loro interazione:

fattore	SS	MS	F-ratio	p-value
c	19147219	19147219	2,02	0,174
l	36647501	36647501	3,87	0,067
cl	65205345	65205345	6,88	0,018
k	1,21E+10	1,21E+10	1281,72	0,000
ck	10427603	10427603	1,10	0,310
lk	43052240	43052240	4,54	0,049
clk	16743238	16743238	1,77	0,202
m	2,19E+10	2,19E+10	2313,59	0,000
cm	17893658	17893658	1,89	0,188
lm	32302713	32302713	3,41	0,083
clm	2957920	2957920	0,31	0,584
km	9,76E+09	9,76E+09	1029,47	0,000
ckm	28457854	28457854	3,00	0,102
lkm	10056491	10056491	1,06	0,318
clkm	8504719	8504719	0,90	0,358

Si può quindi dire che il numero di corridoi e di livelli non sono significativi a patto che i livelli di corridoio siano almeno pari al numero di sistemi ESMARTSHUTTLE® installati.

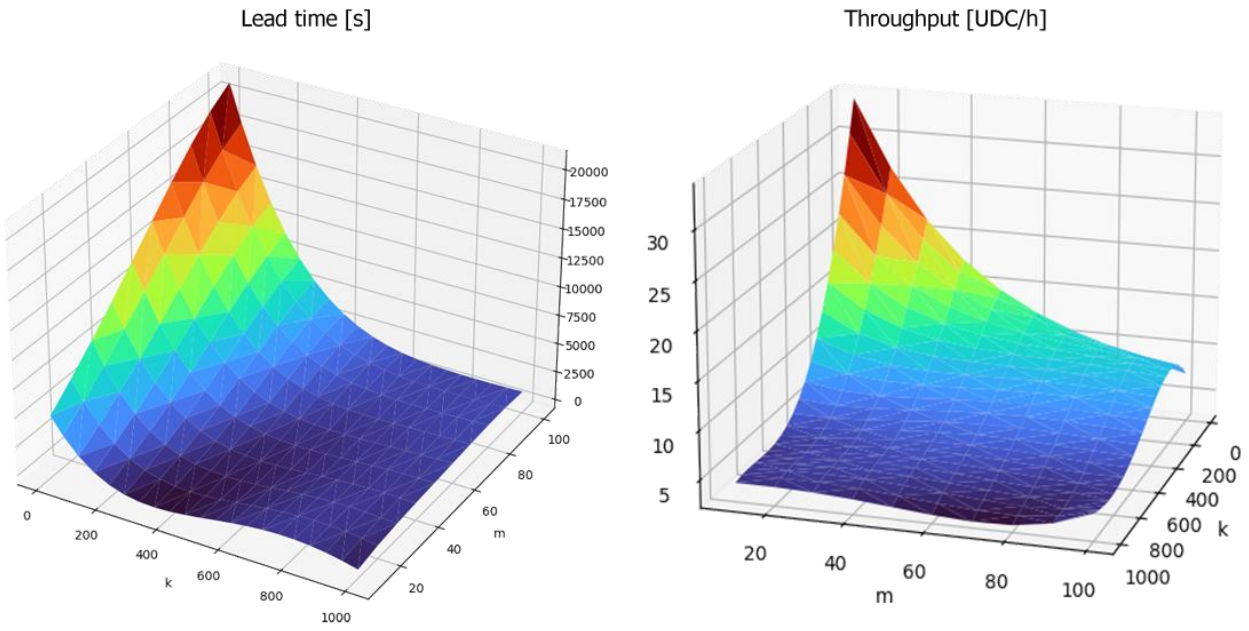

Figura 19 - andamento di lead time e throughput

7.2. STUDIO DELL'ANDAMENTO DEGLI INDICATORI AL VARIARE DEL NUMERO DI NAVETTE

Per studiare le varie configurazioni si sono scelti i seguenti indicatori:

- Tempo medio per evadere l'ordine (*lead time*): il tempo che passa dall'arrivo dell'ordine alla consegna del pallet misto alla coda di uscita;
- Scarto quadratico medio del tempo medio per evadere l'ordine;
- Rateo di uscita (*throughput*): la quantità di ordini evasi in un'ora;

Il sistema è stato testato andando a variare il coefficiente k della legge esponenziale dell'arrivo degli ordini (espresso in secondi) e il numero di sistemi ESMARTSHUTTLE® presenti n. Variando n sono stati cambiati anche il numero di corridoi e di livelli in modo da avere un numero di livelli di corridoio idoneo a sfruttare al massimo tutte le navette disponibili.

Da i grafici riportati di seguito si può notare che il tempo medio (in blu) e il suo scarto quadratico medio (in arancione) seguono lo stesso andamento a mazza da hockey in tutti i casi. Si nota quindi che, raggiunto un numero di navette adeguato a tenere il ritmo degli arrivi delle nuove ordinazioni, non vi è più alcun vantaggio nell'aumentare il numero di navette.

Lo stesso discorso può essere fatto per quanto riguarda il rateo di uscita (in grigio) che raggiunto il proprio valore ottimale, all'ulteriore aumentare di navette, oscillerà intorno a quel valore limite in sola funzione della variabilità degli ordini.

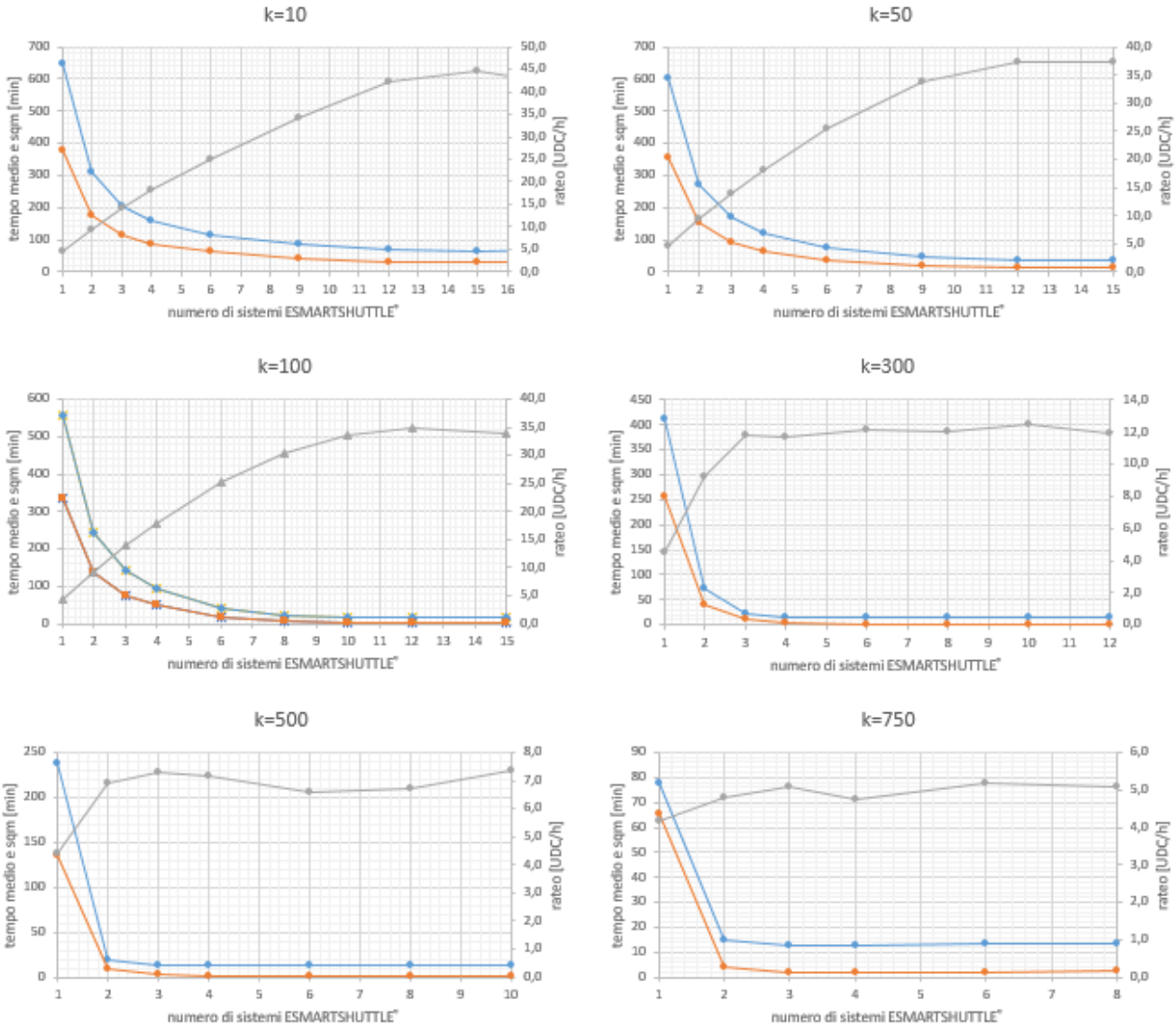

Figura 20 - andamento degli indicatori al variare del numero di navette

Si vuole studiare un metodo per definire quali fattori influenzano, in modo significativo, il numero di navette oltre il quale la diminuzione del tempo medio per evadere l'ordine scende sotto il 5%. Si vuole individuare questo numero in modo che, data una configurazione, ci sia un'indicazione sul numero di sistemi ESMARTSHUTTLE® da installare.

Per l'analisi della varianza si è impostato un semplice esperimento fattoriale 3^2 utilizzando come variabili il coefficiente della legge esponenziale di inter arrivo k e il numero medio di pezzi per il singolo ordine m. L'analisi dell'esperimento fattoriale è stata svolta in forma completa utilizzando l'algoritmo di Yates. La tabella di risoluzione completa è riportata nell'allegato 3. Si imposta un livello di significatività del 1%.

Risultano significative entrambe le variabili prese in considerazione, anche la loro combinazione risulta molto vicina al livello di significatività voluto:

	TSS	dof	MS	F	p-value
k	108,11	2	54,06	108,11	0,000
m	25,44	2	12,72	25,44	0,000
km	12,22	4	3,06	6,11	0,012
errore	4,50	9	0,50		
totale	150,28	17			

Si può quindi dire che, da quel che emerge da questa analisi, i parametri scelti sono tutti significativi per la rappresentazione di questo fenomeno. Verranno quindi inclusi tutti nella scelta del modello matematico.

Nell'ottica dell'analisi regressiva viene raccolto un *dataset* con valori di k casuali tra 1 e 1000 secondi e con valori casuali di m tra 10 e 100 pezzi. Questo dataset viene quindi diviso in un *dataset* di addestramento e in un *dataset* di *test*. Per la scelta del modello sono state prese in considerazione due tipi di regressione:

- Regressione logistica nella quale ogni classe rappresenta il numero di sistemi ESMARTSHUTTLE® da installare;
- Regressione lineare con un'approssimazione all'intero per valutare il numero di sistemi ESMARTSHUTTLE® da installare;

L'approccio tramite regressione logistica non risulta accettabile in quanto non possono essere individuate classi non presenti nel *dataset* di partenza, il modello perderebbe quindi di validità allontanandosi dai valori di k e m presenti nel *dataset*. Inoltre anche per i valori del *dataset* di *test* si ha un errore assoluto medio e un errore assoluto massimo molto superiore rispetto a quelli ottenuti con una regressione lineare. Viene di seguito riportato il riepilogo di confronto, il modello Python per la valutazione di questi modelli viene riportato nell'allegato 4.

	err med	err max
regressione logistica con algoritmo Newton conjugate gradient	2,15	5
regressione logistica con algoritmo limited Broyden–Fletcher–Goldfarb–Shanno	2,15	5
regressione logistica con algoritmo stochastic average gradient descent	2,06	6
regressione logistica con algoritmo stochastic average gradient ascent	2,05	6
regressione lineare minimi quadrati ordinari	0,93	4

Si vuole ora analizzare più nel dettaglio la regressione lineare per vederne caratteristiche e eventuali limiti. Si vuole inoltre applicare un preprocessore per trasformare l'analisi di regressione lineare in una regressione polinomiale in modo da poter studiare la possibilità di aumentare l'aderenza del modello alla realtà usando un polinomio di grado superiore.

7.2.1. Regressione lineare

L'analisi regressiva lineare restituisce l'equazione:

$$n' = 6{,}8889 - 0{,}0084k + 0{,}0301m$$

Dove k è espresso in secondi. Gli indicatori di varianza spiegata e di informazione persa per questo modello sono:

$$R^2 = 0{,}78 \qquad {R_{adj}}^2 = 0{,}77 \qquad AIC = 242 \qquad BIC = 249$$

Confrontando lo scarto tra valore reale e stimato lungo tutto lo spettro del *dataset* ci si può rendere conto che il modello tende alla sottostima per i valori estremi e alla sovrastima dei valori centrali; nonostante questo non si notano *pattern* estremamente evidenti sulla distribuzione dei residui lungo lo spettro del *dataset* di test.

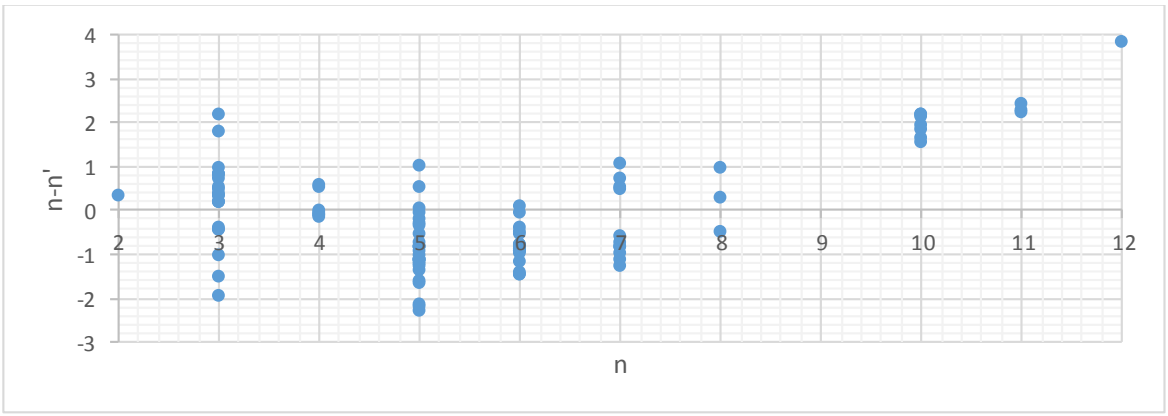

Figura 21 - distribuzione residuo regressione lineare lungo dataset di test

Si vuole valutare anche la normalità del residuo tramite l'input analyzer di *Rockwell Automation*. Vengono quindi divisi i residui in intervalli e viene eseguito sia il *chi-square test* sia il *Kolmogorov-Smirnov test*. Si può notare come la distribuzione in classi ottenuta non sia assimilabile a una distribuzione normale.

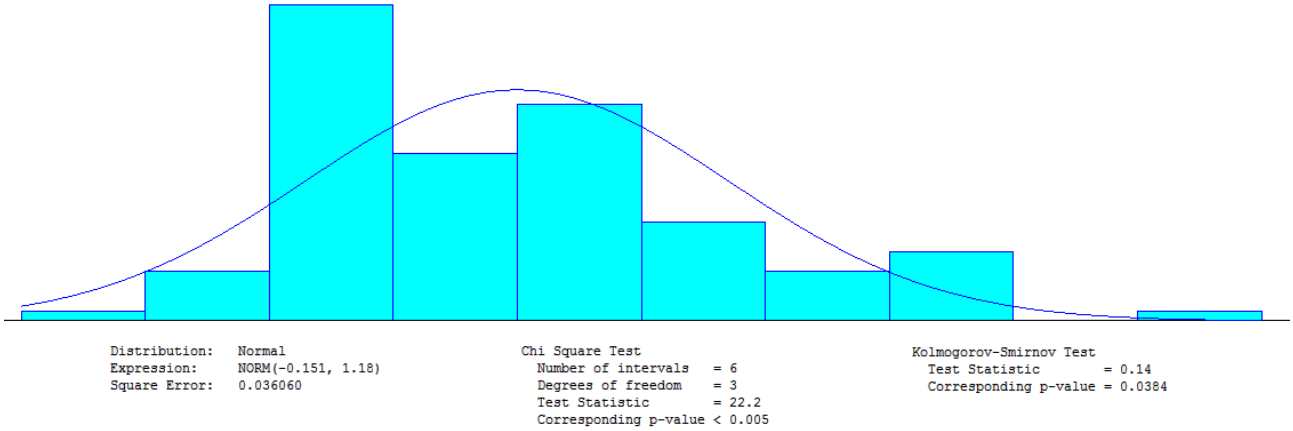

Figura 22 - risultato studio normalità del residuo regressione lineare

I risultati di questo modello regressivo vengono poi arrotondati secondo due filosofie:

- Arrotondamento all'intero più vicino: in questo modo si otterrà l'errore assoluto medio più basso possibile;
- Arrotondamento all'intero superiore: nell'ottica di un'istallazione reale, per assicurarsi di poter sostenere i carichi di lavoro, sarebbe opportuno utilizzare un'approssimazione che minimizza la sottostima;

Utilizzando l'arrotondamento all'intero più vicino si ottiene un errore assoluto medio di 0,93 sistemi ESMARTSHUTTLE®. Si è sottostimato il 24% del *dataset* di test e si è stimato perfettamente il 30% del *dataset* di test.

Utilizzando l'arrotondamento all'intero superiore si ha un incremento dell'errore assoluto medio che sale a 1,11 sistemi ESMARTSHUTTLE®. Si è sottostimato solo il 14% del *dataset* di test e si è stimato perfettamente il 25% del *dataset* di test.

7.2.2. Regressione polinomiale di secondo grado

Si nota come la regressione lineare, pur risultando accettabile, presenti margini di miglioramento, si applica quindi un'analisi regressiva polinomiale di secondo grado che restituisce l'equazione:

$$n' = 6{,}3651 - 0{,}0123k + 0{,}0516m + 1{,}0524\,k^2 10^{-5} - 8{,}1644km\,10^{-5} + 0{,}0002\,m^2$$

Dove k è espresso in secondi.

Gli indicatori di varianza spiegata e di informazione persa per questo modello sono:

$$R^2 = 0{,}84 \qquad {R_{adj}}^2 = 0{,}83 \qquad AIC = 221 \qquad BIC = 236$$

Questi dati indicano che l'aumento di complessità del modello comporta un miglioramento effettivo della bontà del modello stesso.

Confrontando lo scarto tra valore reale e stimato lungo tutto lo spettro del *dataset* ci si può rendere conto che il modello ha quasi del tutto perso l'andamento che contraddistingueva la regressione lineare mantenendo quasi lo stesso comportamento per tutta la lunghezza del *dataset* di test.

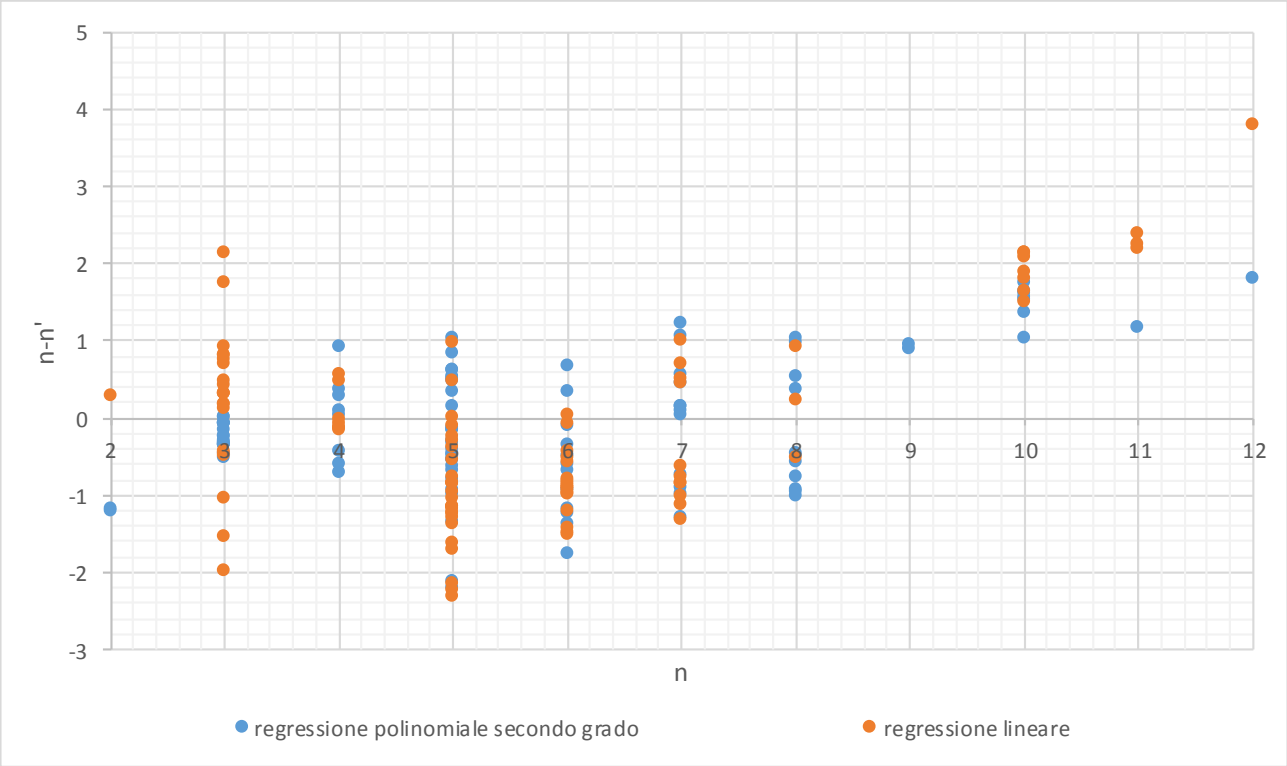

Figura 23 - confronto distribuzione residuo lungo dataset di test

Anche in questo caso si vuole valutare la normalità del residuo tramite l'input analyzer di *Rockwell Automation*. Vengono quindi divisi i residui in intervalli e viene eseguito sia il *chi-square test* si il *Kolmogorov-Smirnov test*. Si può notare come la distribuzione in classi ottenuta non sia ancora impeccabile ma risulti accettabilmente assimilabile a una distribuzione normale.

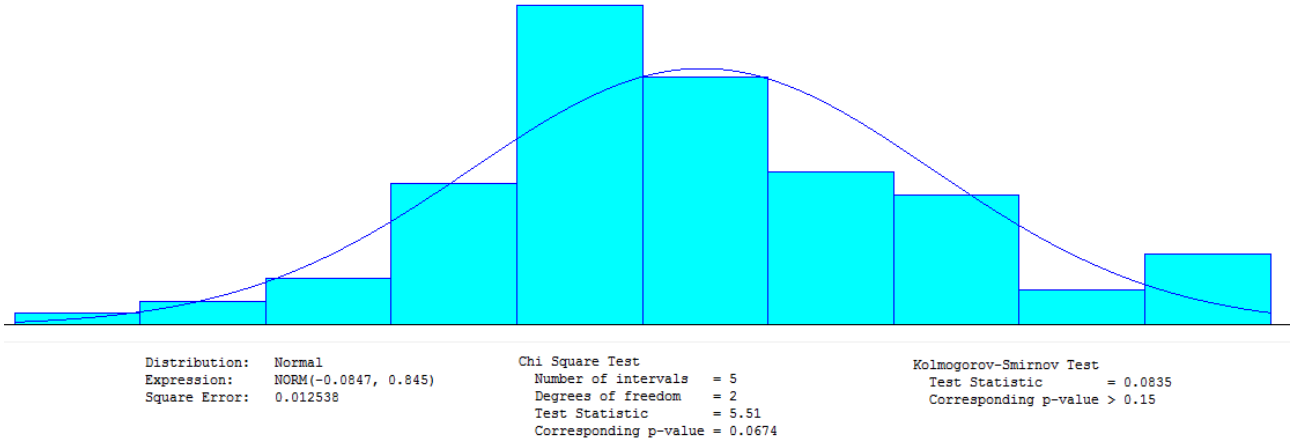

Figura 24 - risultato studio normalità del residuo regressione polinomiale secondo grado

I risultati di questo modello regressivo vengono poi arrotondati secondo due filosofie espresse in precedenza.

Utilizzando l'arrotondamento all'intero più vicino si ottiene un errore assoluto medio di 0,65 sistemi ESMARTSHUTTLE®. Si è sottostimato il 25% del *dataset* di test e si è stimato perfettamente il 44% del *dataset* di test.

Utilizzando l'arrotondamento all'intero superiore si ha un incremento dell'errore assoluto medio che sale a 0,84 sistemi ESMARTSHUTTLE®. Si è sottostimato solo il 13% del *dataset* di test e si è stimato perfettamente il 28% del *dataset* di test.

Si nota quindi che l'aumento di complessità per passare al modello polinomiale di secondo grado viene ripagato con una migliore stima del fenomeno e risulta pertanto conveniente adottare quest'equazione.

Si sono analizzati modelli polinomiali di grado superiore ma non si è riscontrato un miglioramento nella bontà della stima. Non è quindi conveniente aumentare ulteriormente la complessità del modello regressivo. Si può quindi affermare che il modello da adottare risulta quello regressivo polinomiale di secondo grado.

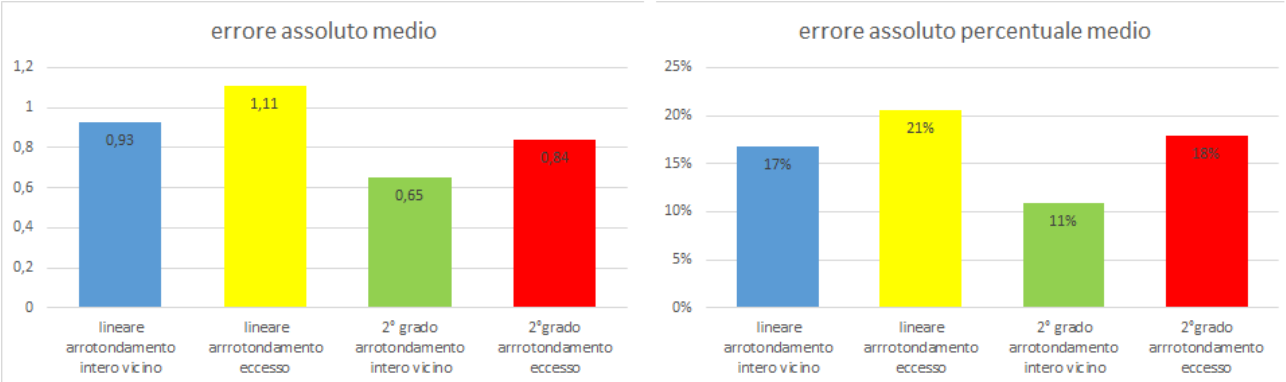

Figura 25 - confronto errore assoluto

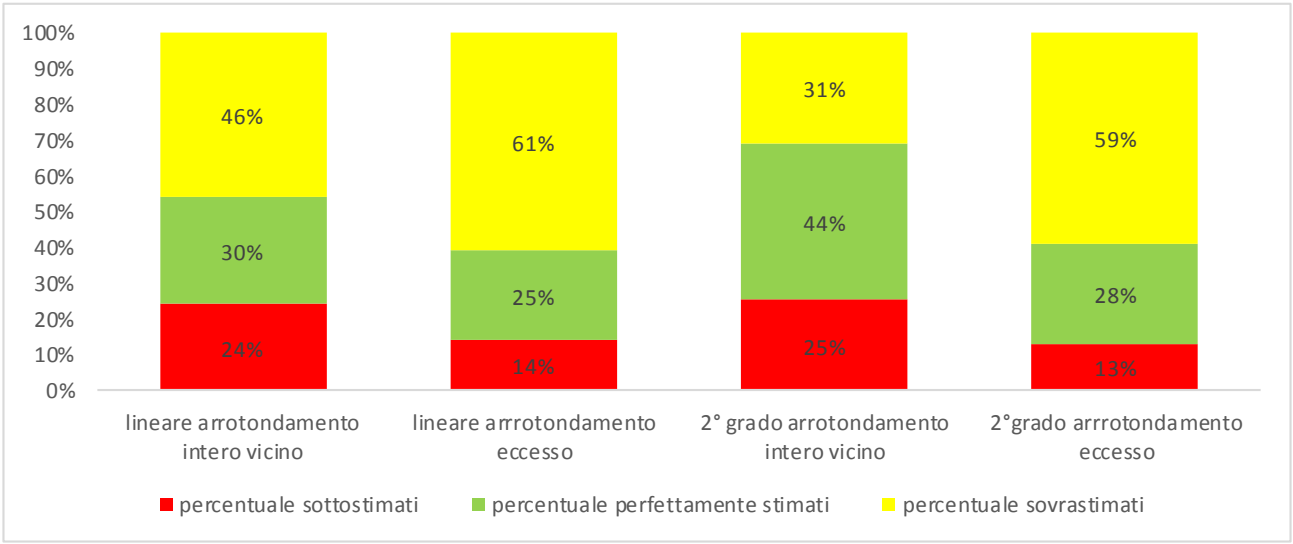

Figura 26 - confronto stime del dataset di test

8. CONCLUSIONI E POSSIBILI SVILUPPI FUTURI DEL MODELLO

8.1. CONCLUSIONI

In conclusione si è riusciti a sviluppare un modello auto-generante e personalizzabile. Sarà possibile quindi utilizzare tale modello per ulteriori analisi su questa specifica modifica dei sistemi ESMARTSHUTTLE® con braccio robotico. Il principale potenziale del modello sviluppato è la possibilità di creare un grande numero di configurazioni variando solamente dei parametri. Si ha inoltre la possibilità di integrare qualsiasi politica di stoccaggio e *picking* utilizzando dei blocchi personalizzati in linguaggio *C++* (o *Flexscript*) e inserendoli in moduli logici già sviluppati.

Si è poi dimostrato come il modello possa essere utilizzato per la raccolta di un *dataset* per eseguire un *virtual DOE* utile per individuare fattori significativi e eventuali equazioni regressive in grado di descrivere il sistema da un punto di vista analitico.

8.2. POSSIBILI SVILUPPI FUTURI DEL MODELLO

Il modello di simulazione a eventi discreti sviluppato si può prestare a ulteriori analisi che, pur andando oltre gli obiettivi di questa testi, potrebbero essere portate avanti utilizzando questa base.

8.2.1. Studio della prestazione legata al posizionamento in magazzino e studi legati alle logiche di rifornimento

Il modello è pronto per funzionare con qualunque distribuzione dei prodotti all'interno del magazzino. Si può quindi utilizzare il modello con un qualsiasi algoritmo di assegnamento delle SKU ai vari slot. In questo modo il modello può essere utilizzato per individuare la migliore logica di assegnazione SKU dato il sistema basato su ESMARTSHUTTLE® che si vuole studiare.

Il modello ha un modulo logico di rifornimento del magazzino che può generare e posizionare sulle scaffalature i pallet interi in modo realistico. Basterà quindi definire le regole per il ripristino del magazzino per svolgere analisi anche su questo punto. In questo modo il modello può essere utilizzato per individuare la migliore politica di ripristino scorte dato il sistema basato su ESMARTSHUTTLE® che si vuole studiare.

8.2.2. Analisi regressive per individuazione del tempo medio

In questa analisi si sono individuati 4 fattori che influenzano significativamente il tempo medio per completare un ordine. Non è stato però possibile valutare un modello regressivo adeguatamente realistico con i *dataset* raccolti. Il modello ha però un numero molto più amplio di fattori modificabili e scalabili a piacere e si presta quindi alla raccolta di *dataset* complessi. Sarà quindi possibile individuare altri fattori di varianza significativi e raccogliere una quantità di dati tale da permettere la formulazione di un'equazione regressiva più affidabile e che permetta di stimare il tempo medio per completare un ordine dato il sistema basato su ESMARTSHUTTLE® che si vuole studiare.

9. ALLEGATO 1 – CODICE GENERAZIONE MODELLO

```
/**********ATTUAZIONE CANCELLAZIONE PER EXPERIMENTER**********/
//Ciclo cancellazione modello
for (int n=content(model()); n>1; n--)
{
    treenode o=rank(model(), n);
    if (o.name=="AGVNetwork")
    {
       while (content(o))
       destroyobject(o.first);
    }
    if(o.selfCreate? == 1)
    {
       destroyobject(o);
    }
}

/*****************INIZIALIZZAZIONE VARIABILI*****************/
int numero_corridoi=Table("Parametri rack")[9][1];
int numero_baie=Table("Parametri rack")[1][1];
int numero_livelli=Table("Parametri rack")[2][1];
int numero_slot=Table("Parametri rack")[3][1];
double larghezza_baia=Table("Parametri rack")[4][1];
double altezza_livello=Table("Parametri rack")[5][1];
double profondita_baia=Table("Parametri rack")[6][1];
double larghezza_corridoi=Table("Parametri rack")[10][1];
double posizione_in_x=Table("Parametri rack")[7][1];
double posizione_in_y=Table("Parametri rack")[8][1];
double distanza=Table("Parametri percorso AGV")[1][1];
int numero_agv=Table("Parametri percorso AGV")[3][1];
double distanza_parcheggi=Table("Parametri percorso AGV")[4][1];

/********************CREAZIONE DEI RACK********************/
/*richiamo libreria*/
treenode rack=library().find("?DriveInRack");
for(int crack=1; crack<=numero_corridoi; crack++)
{
    /*creazione rack lato a*/
    //creazione oggetto
    treenode rack_a=createinstance(rack, model());
    rack_a.selfCreate=1;
    Object rackdamodificare=rack_a;
    //Assegnazione del nome/tipo al rack
```

```
string numero = string.fromNum(crack);
rackdamodificare.name = concat("rack ", numero,"A");
//assegnazione dello schema indirizzi-corridoio(zone)-lato(aisle)
rackdamodificare.getVariable("addressScheme") =
Model.find("Tools/StorageSystem>variables/addressSchemes/Assegnazione indirizzi");
rackdamodificare.getVariable("zoneID") = numtostring(crack);
rackdamodificare.getVariable("aisleID") = "1";
rackdamodificare.getVariable("bayProgression") = -1;
rackdamodificare.getVariable("slotProgression") = -1;
//assegnazione dei parametri al rack
function_s(rackdamodificare, "setNumBays", numero_baie);
function_s(rackdamodificare, "setNumLevels", numero_livelli);
function_s(rackdamodificare, "setNumSlots", numero_slot);
function_s(rackdamodificare, "setBaySize", larghezza_baia);
function_s(rackdamodificare, "setLevelSize", altezza_livello);
rackdamodificare.size.y=profondita_baia;
rackdamodificare.rotation.z=180;
//posizionamento rack
rackdamodificare.setLocation(posizione_in_x, posizione_in_y+(crack-
1)*(2*profondita_baia+larghezza_corridoi), 0);
//consolidamento attributi
rebindobjectattributes(rack_a);

/*creazione rack lato b*/
//creazione oggetto
treenode rack_b=createinstance(rack,model());
rack_b.selfCreate=1;
rackdamodificare=rack_b;
//Assegnazione del nome/tipo al rack
rackdamodificare.name = concat("rack ", numero,"B");
//assegnazione dello schema indirizzi-corridoio(zone)-lato(aisle)
rackdamodificare.getVariable("addressScheme") =
Model.find("Tools/StorageSystem>variables/addressSchemes/Assegnazione indirizzi");
rackdamodificare.getVariable("zoneID") = numtostring(crack);
rackdamodificare.getVariable("aisleID") = "2";
//assegnazione dei parametri al rack
function_s(rackdamodificare, "setNumBays", numero_baie);
function_s(rackdamodificare, "setNumLevels", numero_livelli);
function_s(rackdamodificare, "setNumSlots", numero_slot);
function_s(rackdamodificare, "setBaySize", larghezza_baia);
function_s(rackdamodificare, "setLevelSize", altezza_livello);
rackdamodificare.size.y=profondita_baia;
//posizionamento rack
rackdamodificare.setLocation(posizione_in_x, posizione_in_y+(crack-
1)*(2*profondita_baia+larghezza_corridoi)+profondita_baia+larghezza_corridoi, 0);
```

```
    //consolidamento attributi
    rebindobjectattributes(rack_b);
}

/***************CREAZIONE DEI PERCORSI LATO RACK**************/
/*richiamo libreria*/
treenode dritto=library().find("?StraightPath");
treenode curva=library().find("?CurvedPath");
treenode controlpoint=library().find("?ControlPoint");
for(int ccor=1; ccor<=numero_corridoi; ccor++)
{
    for(int cliv=1; cliv<=numero_livelli; cliv++)
    {
        /*creazione percorso corridoio*/
        //creazione oggetto
        treenode drit=createinstance(dritto, model());
        drit.selfCreate=1;
        Object percdamodificare=drit;
        //assegnazione del nome
        percdamodificare.name = concat("P",string.fromNum(ccor),"A",string.fromNum(cliv));
        //definizione start point
        function_s(percdamodificare, "dragEndpoint", posizione_in_x,
        posizione_in_y+distanza+larghezza_corridoi/2+(ccor-
        1)*(larghezza_corridoi+2*profondita_baia), 1, 2);
        //definizione altezza start point
        function_s(percdamodificare, "setZ", (cliv-1)*altezza_livello, 1);
        //definizione end point
        function_s(percdamodificare, "dragEndpoint",
        posizione_in_x+numero_baie*larghezza_baia,
        posizione_in_y+distanza+larghezza_corridoi/2+(ccor
        1)*(larghezza_corridoi+2*profondita_baia), 1, 1);
        //definizione altezza start point
        function_s(percdamodificare, "setZ", (cliv-1)*altezza_livello, 0);
        percdamodificare.getVariable("isTwoWay")=1;
        //consolidamento attributi
        rebindobjectattributes(drit);

        /*creazione percorso tratto comune 1*/
        //creazione oggetto
        drit=createinstance(dritto, model());
        drit.selfCreate=1;
        percdamodificare=drit;
        //assegnazione del nome
        percdamodificare.name = concat("TC1",string.fromNum(ccor),string.fromNum(cliv));
        //definizione start point
```

```
function_s(percdamodificare, "dragEndpoint", posizione_in_x-larghezza_corridoi,
posizione_in_y+distanza+(ccor-1)*(larghezza_corridoi+2*profondita_baia), 1, 2);
//definizione altezza start point
function_s(percdamodificare, "setZ", (cliv-1)*altezza_livello, 1);
//definizione end point
function_s(percdamodificare, "dragEndpoint", posizione_in_x,
posizione_in_y+distanza+(ccor-1)*(larghezza_corridoi+2*profondita_baia), 1, 1);
//definizione altezza start point
function_s(percdamodificare, "setZ", (cliv-1)*altezza_livello, 0);
//consolidamento attributi
rebindobjectattributes(drit);

/*creazione percorso tratto comune 2*/
//creazione oggetto
drit=createinstance(dritto, model());
drit.selfCreate=1;
percdamodificare=drit;
//assegnazione del nome
percdamodificare.name = concat("TC2",string.fromNum(ccor),string.fromNum(cliv));
//definizione start point
function_s(percdamodificare, "dragEndpoint", posizione_in_x,
posizione_in_y+larghezza_corridoi-distanza+(ccor-
1)*(larghezza_corridoi+2*profondita_baia), 1, 2);
//definizione altezza start point
function_s(percdamodificare, "setZ", (cliv-1)*altezza_livello, 1);
//definizione end point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-larghezza_corridoi,
posizione_in_y+larghezza_corridoi-distanza+(ccor-
1)*(larghezza_corridoi+2*profondita_baia), 1, 1);
//definizione altezza start point
function_s(percdamodificare, "setZ", (cliv-1)*altezza_livello, 0);
//consolidamento attributi
rebindobjectattributes(drit);

/*creazione percorso tratto comune 3*/
//creazione oggetto
drit=createinstance(dritto, model());
drit.selfCreate=1;
percdamodificare=drit;
//assegnazione del nome
percdamodificare.name = concat("TC3",string.fromNum(ccor),string.fromNum(cliv));
//definizione start point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-larghezza_corridoi,
posizione_in_y+larghezza_corridoi-distanza+(ccor-
1)*(larghezza_corridoi+2*profondita_baia), 1, 2);
```

```
//definizione altezza start point
function_s(percdamodificare, "setZ", (cliv-1)*altezza_livello, 1);
//definizione end point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-larghezza_corridoi,
posizione_in_y+distanza+(ccor-1)*(larghezza_corridoi+2*profondita_baia), 1, 1);
//definizione altezza start point
function_s(percdamodificare, "setZ", (cliv-1)*altezza_livello, 0);
//consolidamento attributi
rebindobjectattributes(drit);

/*creazione percorso tratto comune 4*/
//creazione oggetto
drit=createinstance(dritto, model());
drit.selfCreate=1;
percdamodificare=drit;
//assegnazione del nome
percdamodificare.name = concat("TC4",string.fromNum(ccor),string.fromNum(cliv));
//definizione start point
function_s(percdamodificare, "dragEndpoint", posizione_in_x,
posizione_in_y+distanza+(ccor-1)*(larghezza_corridoi+2*profondita_baia), 1, 2);
//definizione altezza start point
function_s(percdamodificare, "setZ", (cliv-1)*altezza_livello, 1);
//definizione end point
function_s(percdamodificare, "dragEndpoint", posizione_in_x,
posizione_in_y+larghezza_corridoi-distanza+(ccor-
1)*(larghezza_corridoi+2*profondita_baia), 1, 1);
//definizione altezza start point - dati da inserire z, 0
function_s(percdamodificare, "setZ", (cliv-1)*altezza_livello, 0);
//consolidamento attributi
rebindobjectattributes(drit);
/*creazione redirect control point*/
//creazione oggetto
treenode cp=createinstance(controlpoint, model());
cp.selfCreate=1;
Object CPdamodificare=cp;
//assegnazione del nome
CPdamodificare.name = concat("CPR", string.fromNum(ccor),string.fromNum(cliv));
//posizionamento CP
CPdamodificare.setLocation(posizione_in_x-larghezza_corridoi,
posizione_in_y+larghezza_corridoi/2+0.1+(ccor-
1)*(larghezza_corridoi+2*profondita_baia), (cliv-1)*altezza_livello);
CPdamodificare.floorZ=(cliv-1)*altezza_livello;
//consolidamento attributi
rebindobjectattributes(cp);
```

```
/*creazione floor control point*/
//creazione oggetto
cp=createinstance(controlpoint, model());
cp.selfCreate=1;
CPdamodificare=cp;
//assegnazione del nome
CPdamodificare.name = concat("CPF", string.fromNum(ccor),string.fromNum(cliv));
//posizionamento CP
CPdamodificare.setLocation(posizione_in_x-larghezza_corridoi,
posizione_in_y+larghezza_corridoi/2-0.1+(ccor-
1)*(larghezza_corridoi+2*profondita_baia), (cliv-1)*altezza_livello);
//consolidamento attributi
rebindobjectattributes(cp);

for(int cbai=1; cbai<=numero_baie; cbai++)
{
	for(int cslot=1; cslot<=numero_slot; cslot++)
	{
		/*creazione control point corridoi*/
		//creazione oggetto
		cp=createinstance(controlpoint, model());
		cp.selfCreate=1;
		cp.occupato=0;
		CPdamodificare=cp;
		//assegnazione del nome
		if (cbai < 10)
		{
		CPdamodificare.name =
		concat("CP",string.fromNum(ccor),string.fromNum(cliv),"0",string.from
		Num(cbai),string.fromNum(cslot));
		}
		else
		{
		CPdamodificare.name =
		concat("CP",string.fromNum(ccor),string.fromNum(cliv),string.fromNum(
		cbai),string.fromNum(cslot));
		}
	//posizionamento CP
	CPdamodificare.setLocation(posizione_in_x+(cbai-1)*larghezza_baia+(cslot-
	1)*2*larghezza_baia/(2*numero_slot)+larghezza_baia/(2*numero_slot),
	posizione_in_y+distanza+larghezza_corridoi/2+(ccor-
	1)*(larghezza_corridoi+2*profondita_baia), (cliv-1)*altezza_livello);
	//consolidamento attributi
	rebindobjectattributes(cp);
	}
```

```
        }
    }
}

/*****************CREAZIONE DEGLI ASCENSORI****************/
/*richiamo libreria*/
treenode ascensore=maintree().find("project/library/taskexecuters/Elevator");
for(int crack=1;crack<=numero_corridoi; crack++)
{
    /*creazione ascensore*/
    //creazione oggetto
    treenode asc=createinstance(ascensore, model());
    asc.selfCreate=1;
    Object ascdamodificare=asc;
    //assegnazione del nome all'ascensore
    ascdamodificare.name = concat("ascensore", numtostring(crack));
    //inserimento dell'ascensore nel process flow dedicato
    function_s(Model.find("Tools/ProcessFlow/AGV Elevator"), "attachObject",
    ascdamodificare);
    //posizionamento ascensore
    ascdamodificare.setLocation(posizione_in_x-2*larghezza_corridoi,
    posizione_in_y+larghezza_corridoi+(crack-1)*(larghezza_corridoi+2*profondita_baia),
    0);
    //consolidamento attributi
    rebindobjectattributes(asc);
}

/*****************CREAZIONE DEL PARCHEGGIO****************/
/*creazione percorso tratto 1*/
//creazione oggetto
treenode drit=createinstance(dritto, model());
drit.selfCreate=1;
Object percdamodificare=drit;
//assegnazione del nome
percdamodificare.name = "P1";
//definizione start point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi-
distanza_parcheggi-numero_agv*distanza_parcheggi, posizione_in_y-profondita_baia, 1, 1);
//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 1);
//definizione end point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi,
posizione_in_y-profondita_baia, 1, 2);
//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 0);
```

```
//consolidamento attributi
rebindobjectattributes(drit);

/*creazione percorso tratto 2*/
//creazione oggetto
drit=createinstance(dritto, model());
drit.selfCreate=1;
percdamodificare=drit;
//assegnazione del nome
percdamodificare.name = "P2";
//definizione start point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi,
posizione_in_y-
2*profondita_baia+(2*profondita_baia+larghezza_corridoi)*(numero_corridoi), 1, 2);
//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 1);
//definizione end point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi,
posizione_in_y-profondita_baia, 1, 1);
//definizione altezza start point - dati da inserire z, 0
function_s(percdamodificare, "setZ", 0, 0);
//consolidamento attributi
rebindobjectattributes(drit);

/*creazione percorso tratto 3*/
//creazione oggetto
drit=createinstance(dritto, model());
drit.selfCreate=1;
percdamodificare=drit;
//assegnazione del nome
percdamodificare.name = "P3";
//definizione start point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi-
distanza_parcheggi-numero_agv*distanza_parcheggi, posizione_in_y-profondita_baia, 1, 2);

//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 1);
//definizione end point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi-
distanza_parcheggi-numero_agv*distanza_parcheggi, posizione_in_y-
2*profondita_baia+(2*profondita_baia+larghezza_corridoi)*(numero_corridoi), 1, 1);
//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 0);
//consolidamento attributi
rebindobjectattributes(drit);
```

```
/*creazione percorso tratto 4*/
//creazione oggetto
drit=createinstance(dritto, model());
drit.selfCreate=1;
percdamodificare=drit;
//assegnazione del nome
percdamodificare.name = "P4";
//definizione start point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi-
distanza_parcheggi-numero_agv*distanza_parcheggi,
posizione_in_y+larghezza_corridoi+(2*profondita_baia+larghezza_corridoi)*(numero_corridoi
-1), 1, 2);
//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 1);
//definizione end point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi,
posizione_in_y+larghezza_corridoi+(2*profondita_baia+larghezza_corridoi)*(numero_corridoi
-1), 1, 1);
//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 0);
//consolidamento attributi
rebindobjectattributes(drit);

for(int cagv=1; cagv<=numero_agv; cagv++)
{
    /*creazione tratto personale parcheggio*/
    //creazione oggetto
    treenode drit=createinstance(dritto, model());
    drit.selfCreate=1;
    Object percdamodificare=drit;
    //assegnazione del nome
    percdamodificare.name = concat("TP", string.fromNum(cagv));
    //definizione start point
    function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi-
    cagv*distanza_parcheggi, posizione_in_y-profondita_baia, 1, 1);
    //definizione altezza start point
    function_s(percdamodificare, "setZ", 0, 1);
    //definizione end point
    function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi-
    cagv*distanza_parcheggi, posizione_in_y-profondita_baia-2*distanza_parcheggi, 1, 2);
    //definizione altezza start point
    function_s(percdamodificare, "setZ", 0, 0);
    percdamodificare.getVariable("isTwoWay")=1;
    //consolidamento attributi
```

```
    rebindobjectattributes(drit);

    /*creazione posteggio*/
    //creazione oggetto
    treenode parcheggio=createinstance(controlpoint, model());
    parcheggio.selfCreate=1;
    Object CPdamodificare=parcheggio;
    //Assegnazione del nome
    CPdamodificare.name = concat("Posteggio","",string.fromNum(cagv));
    /*posizionamento control point*/
    CPdamodificare.setLocation(posizione_in_x-2*larghezza_corridoi-
    cagv*distanza_parcheggi, posizione_in_y-profondita_baia-larghezza_corridoi-
    distanza_parcheggi, 0);
    //consolidamento attributi
    rebindobjectattributes(parcheggio);
}

for(int ccor=1; ccor<=numero_corridoi; ccor++)
{
    /*creazione redirect control point ingresso*/
    //creazione oggetto
    treenode ingresso=createinstance(controlpoint, model());
    ingresso.selfCreate=1;
    Object CPdamodificare=ingresso;
    //Assegnazione del nome
    CPdamodificare.name = concat("CPRI", string.fromNum(ccor));
    /*posizionamento AGV*/
    CPdamodificare.setLocation(posizione_in_x-2*larghezza_corridoi,
    posizione_in_y+larghezza_corridoi/2+0.1+(ccor-
    1)*(larghezza_corridoi+2*profondita_baia), 0);
    CPdamodificare.floorZ=0;
    //consolidamento attributi
    rebindobjectattributes(ingresso);

    /*creazione floor control point ingresso*/
    //creazione oggetto
    ingresso=createinstance(controlpoint, model());
    ingresso.selfCreate=1;
    CPdamodificare=ingresso;
    //Assegnazione del nome
    CPdamodificare.name = concat("CPFI", string.fromNum(ccor));
    /*posizionamento AGV*/
    CPdamodificare.setLocation(posizione_in_x-2*larghezza_corridoi,
    posizione_in_y+larghezza_corridoi/2-0.1+(ccor-
    1)*(larghezza_corridoi+2*profondita_baia), 0);
```

```
    //consolidamento attributi
    rebindobjectattributes(ingresso);

    /*creazione control point libera ascensore*/
    //creazione oggetto
    ingresso=createinstance(controlpoint, model());
    ingresso.selfCreate=1;
    CPdamodificare=ingresso;
    //Assegnazione del nome
    CPdamodificare.name = concat("CPL", string.fromNum(ccor));
    /*posizionamento AGV*/
    CPdamodificare.setLocation(posizione_in_x-2*larghezza_corridoi,
    posizione_in_y+(2*profondita_baia+larghezza_corridoi)*(ccor-1)-larghezza_corridoi/2,
    0);
    //consolidamento attributi
    rebindobjectattributes(ingresso);

    /*creazione control point supporto ascensore*/
    //creazione oggetto
    ingresso=createinstance(controlpoint, model());
    ingresso.selfCreate=1;
    CPdamodificare=ingresso;
    //Assegnazione del nome
    CPdamodificare.name = concat("CPS", string.fromNum(ccor));
    /*posizionamento AGV*/
    CPdamodificare.setLocation(posizione_in_x-2*larghezza_corridoi,
    posizione_in_y+(2*profondita_baia+larghezza_corridoi)*(ccor-1)-larghezza_corridoi/2-
    distanza_parcheggi, 0);
    //consolidamento attributi
    rebindobjectattributes(ingresso);
}

/*creazione punto intermedio 1*/
//creazione oggetto
treenode punto=createinstance(controlpoint, model());
punto.selfCreate=1;
Object CPdamodificare=punto;
//Assegnazione del nome
CPdamodificare.name = "CPINT1";
/*posizionamento AGV*/
CPdamodificare.setLocation(posizione_in_x-2*larghezza_corridoi-distanza_parcheggi-
numero_agv*distanza_parcheggi, posizione_in_y-profondita_baia, 0);
//consolidamento attributi
rebindobjectattributes(punto);
```

```
/*creazione punto intermedio 2*/
//creazione oggetto
punto=createinstance(controlpoint, model());
punto.selfCreate=1;
CPdamodificare=punto;
//Assegnazione del nome
CPdamodificare.name = "CPINT2";
/*posizionamento AGV*/
CPdamodificare.setLocation(posizione_in_x-2*larghezza_corridoi-distanza_parcheggi-
numero_agv*distanza_parcheggi,
posizione_in_y+larghezza_corridoi+(2*profondita_baia+larghezza_corridoi)*(numero_corridoi
-1), 0);
//consolidamento attributi
rebindobjectattributes(punto);

/*creazione punto intermedio 3*/
//creazione oggetto
punto=createinstance(controlpoint, model());
punto.selfCreate=1;
CPdamodificare=punto;
//Assegnazione del nome
CPdamodificare.name = "CPINT3";
/*posizionamento AGV*/
CPdamodificare.setLocation(posizione_in_x-2*larghezza_corridoi-distanza_parcheggi-
numero_agv*distanza_parcheggi+5,
posizione_in_y+larghezza_corridoi+(2*profondita_baia+larghezza_corridoi)*(numero_corridoi
-1), 0);
//consolidamento attributi
rebindobjectattributes(punto);

/*creazione punto intermedio 4*/
//creazione oggetto
punto=createinstance(controlpoint, model());
punto.selfCreate=1;
CPdamodificare=punto;
//Assegnazione del nome
CPdamodificare.name = "CPINT4";
/*posizionamento AGV*/
CPdamodificare.setLocation(posizione_in_x-2*larghezza_corridoi, posizione_in_y-
profondita_baia, 0);
//consolidamento attributi
rebindobjectattributes(punto);

for (int ct=1; ct<=numero_corridoi; ct++)
{
```

```
/*creazione loop ritorno 1*/
//creazione oggetto
treenode drit=createinstance(dritto, model());
drit.selfCreate=1;
Object percdamodificare=drit;
//assegnazione del nome
percdamodificare.name = concat("L1",string.fromNum(ct));
//definizione start point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi,
posizione_in_y+(2*profondita_baia+larghezza_corridoi)*(ct-1)-larghezza_corridoi/2, 1,
2);
//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 1);
//definizione end point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi-
distanza_parcheggi, posizione_in_y+(2*profondita_baia+larghezza_corridoi)*(ct-1)-
larghezza_corridoi/2, 1, 1);
//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 0);
//consolidamento attributi
rebindobjectattributes(drit);

/*creazione loop ritorno 2*/
//creazione oggetto
drit=createinstance(dritto, model());
drit.selfCreate=1;
percdamodificare=drit;
//assegnazione del nome
percdamodificare.name = concat("L2",string.fromNum(ct));
//definizione start point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi,
posizione_in_y+larghezza_corridoi/2+0.1+(ct-
1)*(larghezza_corridoi+2*profondita_baia), 1, 1);
//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 1);
//definizione end point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi-
distanza_parcheggi, posizione_in_y+larghezza_corridoi/2+0.1+(ct-
1)*(larghezza_corridoi+2*profondita_baia), 1, 2);
//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 0);
//consolidamento attributi
rebindobjectattributes(drit);

/*creazione del punto intermedio 6*/
```

```
    //creazione oggetto
    punto=createinstance(controlpoint, model());
    punto.selfCreate=1;
    CPdamodificare=punto;
    //Assegnazione del nome
    CPdamodificare.name = concat("PL",string.fromNum(ct));;
    /*posizionamento CP*/
    CPdamodificare.setLocation(posizione_in_x-2*larghezza_corridoi-distanza_parcheggi,
    posizione_in_y+(2*profondita_baia+larghezza_corridoi)*(ct-1)-larghezza_corridoi/2,
    0);
    //consolidamento attributi
    rebindobjectattributes(punto);
}

/*creazione loop ritorno 3*/
//creazione oggetto
drit=createinstance(dritto, model());
drit.selfCreate=1;
percdamodificare=drit;
//assegnazione del nome
percdamodificare.name = "L3";
//definizione start point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi-
distanza_parcheggi, posizione_in_y-larghezza_corridoi/2, 1, 2);
//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 1);
//definizione end point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi-
distanza_parcheggi, posizione_in_y+larghezza_corridoi/2+0.1+(numero_corridoi-
1)*(larghezza_corridoi+2*profondita_baia), 1, 1);
//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 0);
//consolidamento attributi
rebindobjectattributes(drit);

/*creazione loop ritorno 4*/
//creazione oggetto
drit=createinstance(dritto, model());
drit.selfCreate=1;
percdamodificare=drit;
//assegnazione del nome
percdamodificare.name = "L4";
//definizione start point
```

```
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi-
(numero_agv*distanza_parcheggi-2*distanza_parcheggi)/2, posizione_in_y-profondita_baia,
1, 2);
//definizione altezza start point - dati da inserire z, 1
function_s(percdamodificare, "setZ", 0, 1);
//definizione end point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi-
(numero_agv*distanza_parcheggi-2*distanza_parcheggi)/2,
posizione_in_y+larghezza_corridoi+(2*profondita_baia+larghezza_corridoi)*(numero_corridoi
-1), 1, 1);
//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 0);
//consolidamento attributi
rebindobjectattributes(drit);

/*creazione del punto intermedio 5*/
//creazione oggetto
punto=createinstance(controlpoint, model());
punto.selfCreate=1;
CPdamodificare=punto;
//Assegnazione del nome
CPdamodificare.name = "CPINT5";
/*posizionamento CP*/
CPdamodificare.setLocation(posizione_in_x-2*larghezza_corridoi-
(numero_agv*distanza_parcheggi-2*distanza_parcheggi)/2, posizione_in_y-
2*profondita_baia+(2*profondita_baia+larghezza_corridoi)*(numero_corridoi)-
distanza_parcheggi, 0);
//consolidamento attributi
rebindobjectattributes(punto);

/*creazione del punto intermedio 6*/
//creazione oggetto
punto=createinstance(controlpoint, model());
punto.selfCreate=1;
CPdamodificare=punto;
//Assegnazione del nome
CPdamodificare.name = "CPINT6";
/*posizionamento CP*/
CPdamodificare.setLocation(posizione_in_x-2*larghezza_corridoi-
(numero_agv*distanza_parcheggi-2*distanza_parcheggi)/2, posizione_in_y-
2*profondita_baia+(2*profondita_baia+larghezza_corridoi)*(numero_corridoi)-
2*distanza_parcheggi, 0);
//consolidamento attributi
rebindobjectattributes(punto);
```

```
/**************UNIONE ZONA PARCHEGGIO E ZONA RACK************/
for(int ccor=1; ccor<=numero_corridoi; ccor++)
{
    for(int cliv=1; cliv<=numero_livelli; cliv++)
    {
        /*creazione tratto personale parcheggio*/
        //creazione oggetto
        treenode drit=createinstance(dritto, model());
        drit.selfCreate=1;
        Object percdamodificare=drit;
        //assegnazione del nome
        percdamodificare.name = concat("TPR",string.fromNum(ccor),string.fromNum(cliv));
        //definizione start point
        function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi,
        posizione_in_y+distanza+(ccor-1)*(larghezza_corridoi+2*profondita_baia), 1, 1);
        //definizione altezza start point
        function_s(percdamodificare, "setZ", (cliv-1)*altezza_livello, 1);
        //definizione end point
        function_s(percdamodificare, "dragEndpoint", posizione_in_x-larghezza_corridoi,
        posizione_in_y+distanza+(ccor-1)*(larghezza_corridoi+2*profondita_baia), 1, 2);
        //definizione altezza start point
        function_s(percdamodificare, "setZ", 0, 0);
        percdamodificare.getVariable("isTwoWay")=1;
        //consolidamento attributi
        rebindobjectattributes(drit);
    }
}

/********************CREAZIONE DEGLI AGV*******************/
/*richiamo libreria*/
treenode agv=library().find("?TaskExecuter");
treenode robot=library().find("?Robot");
for(int cagv=1; cagv<=numero_agv; cagv++)
{
    /*creazione agv*/
    //creazione oggetto principale
    treenode agvi=createinstance(agv, model());
    agvi.selfCreate=1;
    Object AGVdamodificare=agvi;
    //Assegnazione del nome
    AGVdamodificare.name = concat("AGV","",string.fromNum(cagv));
    AGVdamodificare.nAGV = cagv;
    //posizionamento AGV
```

```
    AGVdamodificare.setLocation(posizione_in_x-2*larghezza_corridoi-
    cagv*distanza_parcheggi-(distanza_parcheggi/2), posizione_in_y-
    profondita_baia+distanza_parcheggi, 0);
    //consolidamento attributi
    rebindobjectattributes(agvi);
    //creazione satellite
    string padre=concat("AGV",string.fromNum(cagv));
    treenode satellite=createinstance(agv,Model.find(padre));
    satellite.selfCreate=1;
    Object satdamodificare=satellite;
    //Assegnazione del nome e dimensione
    satdamodificare.name = concat("sat","",string.fromNum(cagv));
    satdamodificare.size.x=1.4;
    satdamodificare.size.y=1;
    satdamodificare.size.z=1;
    satdamodificare.nSAT = cagv;
    //posizionamento satellite
    satdamodificare.setLocation(0, 0, 0);
    //consolidamento attributi
    rebindobjectattributes(satellite);
    //creazione robot
    padre=concat("AGV",string.fromNum(cagv));
    treenode robot=createinstance(robot,Model.find(padre));
    robot.selfCreate=1;
    Object robdamodificare=robot;
    //Assegnazione del nome e dimensione
    robdamodificare.name = concat("rob","",string.fromNum(cagv));
    //posizionamento robot
    robdamodificare.setLocation(0.3, -0.2, 0.7);
    robdamodificare.size.x=0.4;
    robdamodificare.nROB = cagv;
    //consolidamento attributi
    rebindobjectattributes(robot);
}

/*****************CREAZIONE DELLE CONNESSIONI****************/
/*creazione connessioni AGV-posteggi*/
for(int cagv=1; cagv<=numero_agv; cagv++)
{
    string posteggio=concat("Posteggio",string.fromNum(cagv));
    string agv=concat("AGV",string.fromNum(cagv));
    function_s(Model.find(posteggio), "addConnection", 0, Model.find(agv));
}
/*creazione connessioni control point-ascensore*/
for(int ccor=1; ccor<=numero_corridoi; ccor++)
```

```
{
    for(int cliv=1; cliv<=numero_livelli; cliv++)
    {
        string ascensore=concat("ascensore",string.fromNum(ccor));
        string floor=concat("CPF",string.fromNum(ccor),string.fromNum(cliv));
        string redirect=concat("CPR",string.fromNum(ccor),string.fromNum(cliv));
        //assegnazione floor control point
        function_s(Model.find(floor), "addConnection", 7, Model.find(ascensore));
        //assegnazione floor control point
        function_s(Model.find(redirect), "addConnection", 8, Model.find(ascensore));
    }
}
/*creazione connessioni ingresso-ascensore*/
for(int ccor=1; ccor<=numero_corridoi; ccor++)
{
    string ascensore=concat("ascensore",string.fromNum(ccor));
    string ingresso=concat("CPRI",string.fromNum(ccor));
    function_s(Model.find(ingresso), "addConnection", 8, Model.find(ascensore));
    ascensore=concat("ascensore",string.fromNum(ccor));
    ingresso=concat("CPFI",string.fromNum(ccor));
    function_s(Model.find(ingresso), "addConnection", 7, Model.find(ascensore));
}

/***************CREAZIONE ZONA ARRIVI PALLET****************/
/*creazione del control point*/
//creazione oggetto
treenode arrivi=createinstance(controlpoint, model());
arrivi.selfCreate=1;
CPdamodificare=arrivi;
//Assegnazione del nome
CPdamodificare.name = "arrivi";
/*posizionamento AGV*/
CPdamodificare.setLocation(posizione_in_x-2*larghezza_corridoi-distanza_parcheggi-
numero_agv*distanza_parcheggi+3,
posizione_in_y+larghezza_corridoi+(2*profondita_baia+larghezza_corridoi)*(numero_corridoi
-1), 0);
//consolidamento attributi
rebindobjectattributes(arrivi);

/*creazione della coda*/
//creazione oggetto
treenode coda=library().find("?Queue");
treenode codai=createinstance(coda, model());
codai.selfCreate=1;
Object codadamodificare=codai;
```

```
//Assegnazione del nome
codadamodificare.name = "CODA ARRIVI";
/*posizionamento AGV*/
codadamodificare.setLocation(posizione_in_x-2*larghezza_corridoi-distanza_parcheggi-
numero_agv*distanza_parcheggi+2,
posizione_in_y+larghezza_corridoi+3+(2*profondita_baia+larghezza_corridoi)*(numero_corrid
oi-1), 0);
codadamodificare.getVariable("placemode")=2;
//consolidamento attributi
rebindobjectattributes(codai);

/*********************CREAZIONE BUFFER*********************/
/*creazione buffer tratto 1*/
//creazione oggetto
drit=createinstance(dritto, model());
drit.selfCreate=1;
percdamodificare=drit;
//assegnazione del nome
percdamodificare.name = "B1";
//definizione start point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi,
posizione_in_y-
2*profondita_baia+distanza_parcheggi/2+(2*profondita_baia+larghezza_corridoi)*(numero_cor
ridoi), 1, 2);
//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 1);
//definizione end point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi,
posizione_in_y-
2*profondita_baia+(2*profondita_baia+larghezza_corridoi)*(numero_corridoi), 1, 1);
//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 0);
//consolidamento attributi
rebindobjectattributes(drit);

/*creazione buffer tratto 2*/
//creazione oggetto
drit=createinstance(dritto, model());
drit.selfCreate=1;percdamodificare=drit;
//assegnazione del nome
percdamodificare.name = "B2";
//definizione start point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi-
(numero_agv*distanza_parcheggi-2*distanza_parcheggi)/2, posizione_in_y-
```

```
2*profondita_baia+distanza_parcheggi/2+(2*profondita_baia+larghezza_corridoi)*(numero_cor
ridoi), 1, 2);
//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 1);
//definizione end point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi,
posizione_in_y-
2*profondita_baia+distanza_parcheggi/2+(2*profondita_baia+larghezza_corridoi)*(numero_cor
ridoi), 1, 1);
//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 0);
//consolidamento attributi
rebindobjectattributes(drit);

/*creazione buffer tratto 3*/
//creazione oggetto
drit=createinstance(dritto, model());
drit.selfCreate=1;
percdamodificare=drit;
//assegnazione del nome
percdamodificare.name = "B3";
//definizione start point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi-
(numero_agv*distanza_parcheggi-2*distanza_parcheggi)/2, posizione_in_y-
2*profondita_baia+(2*profondita_baia+larghezza_corridoi)*(numero_corridoi), 1, 2);
//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 1);
//definizione end point
function_s(percdamodificare, "dragEndpoint", posizione_in_x-2*larghezza_corridoi-
(numero_agv*distanza_parcheggi-2*distanza_parcheggi)/2, posizione_in_y-
2*profondita_baia+distanza_parcheggi/2+(2*profondita_baia+larghezza_corridoi)*(numero_cor
ridoi), 1, 1);
//definizione altezza start point
function_s(percdamodificare, "setZ", 0, 0);
//consolidamento attributi
rebindobjectattributes(drit);

/*creazione del control point ingresso buffer*/
//creazione oggetto
treenode buffer=createinstance(controlpoint, model());
buffer.selfCreate=1;
CPdamodificare=buffer;
//Assegnazione del nome
CPdamodificare.name = "ingresso buffer";
/*posizionamento AGV*/
```

```
CPdamodificare.setLocation(posizione_in_x-2*larghezza_corridoi-
(numero_agv*distanza_parcheggi-2*distanza_parcheggi)/2, posizione_in_y-
2*profondita_baia+(2*profondita_baia+larghezza_corridoi)*(numero_corridoi), 0);
//consolidamento attributi
rebindobjectattributes(buffer);

/*creazione control point buffer*/
for (int cagv=1;cagv<=numero_agv/2;cagv++)
{
    //creazione oggetto
    buffer=createinstance(controlpoint, model());
    buffer.selfCreate=1;
    CPdamodificare=buffer;
    //Assegnazione del nome
    CPdamodificare.name = concat("CPB",string.fromNum(cagv));
    /*posizionamento AGV*/
    CPdamodificare.setLocation(posizione_in_x-2*larghezza_corridoi-(cagv-
    1)*distanza_parcheggi, posizione_in_y-
    2*profondita_baia+distanza_parcheggi/2+(2*profondita_baia+larghezza_corridoi)*(numer
    o_corridoi), 0);
    //consolidamento attributi
    rebindobjectattributes(buffer);
 }

 /****************CREAZIONE ZONA USCITA PALLET****************/
 /*creazione della coda*/
 //creazione oggetto
 coda=library().find("?Queue");
 codai=createinstance(coda, model());
 codai.selfCreate=1;
 codadamodificare=codai;
 //Assegnazione del nome
 codadamodificare.name = "CODA USCITA";
 /*posizionamento AGV*/
 codadamodificare.setLocation(posizione_in_x-2*larghezza_corridoi, posizione_in_y-
 profondita_baia-larghezza_corridoi-2*distanza_parcheggi, 0);
 codadamodificare.getVariable("placemode")=2;
 //consolidamento attributi
 rebindobjectattributes(codai);
```

10. ALLEGATO 2 – TABELLA ANALISI DELLA VARIANZA CORRIDOI E LIVELLI

c	l	k	m	sample 1	sample 2	somma	1	2	3	4	SS	gdl	MS	F-ratio	p-value
0	0	0	0	28405	29569	57974	120136	237437	1066038	2969793					
1	0	0	0	31113	31049	62162	117301	828601	1903755	24753	1,91E+07	1	1,91E+07	2,02	0,174
0	1	0	0	27208	27378	54586	429482	935699	412	-34245	3,66E+07	1	3,66E+07	3,87	0,067
1	1	0	0	31550	31165	62715	399119	968056	24341	45679	6,52E+07	1	6,52E+07	6,88	0,018
0	0	1	0	106884	116774	223658	465714	12317	-33198	623521	1,21E+10	1	1,21E+10	1281,72	0,000
1	0	1	0	105441	100383	205824	469985	-11905	-1047	-18267	1,04E+07	1	1,04E+07	1,10	0,310
0	1	1	0	99662	96933	196595	486687	9193	27704	-37117	4,31E+07	1	4,31E+07	4,54	0,049
1	1	1	0	106234	96290	202524	481369	15148	17975	23147	1,67E+07	1	1,67E+07	1,77	0,202
0	0	0	1	117635	114755	232390	4188	-2835	591164	837717	2,19E+10	1	2,19E+10	2313,59	0,000
1	0	0	1	116144	117180	233324	8129	-30363	32357	23929	1,79E+07	1	1,79E+07	1,89	0,188
0	1	0	1	114920	115943	230863	-17834	4271	-24222	32151	3,23E+07	1	3,23E+07	3,41	0,083
1	1	0	1	120988	118134	239122	5929	-5318	5955	-9729	2,96E+06	1	2,96E+06	0,31	0,584
0	0	1	1	123829	118390	242219	934	3941	-27528	-558807	9,76E+09	1	9,76E+09	1029,47	0,000
1	0	1	1	120290	124178	244468	8259	23763	-9589	30177	2,85E+07	1	2,85E+07	3,00	0,102
0	1	1	1	115766	118469	234235	2249	7325	19822	17939	1,01E+07	1	1,01E+07	1,06	0,318
1	1	1	1	122955	124179	247134	12899	10650	3325	-16497	8,50E+06	1	8,50E+06	0,90	0,358
					tot	2969793				errore	1,52E+08	16	9,48E+06		
					med	92806				totale	4,43E+10	31			

11. ALLEGATO 3 – TABELLA ANALISI DELLA VARIANZA NUMERO IDEALE DI NAVETTE

k	m	sample 1	sample 2	somma	1	2	effetto				
0	0	6	6	12	25	97	5,39	divisore	SS	F	p-value
1	0	4	3	7	30	-33	-5,50	12	90,75	181,50	0,000
2	0	3	3	6	42	25	4,17	36	17,36	34,72	0,000
0	1	10	8	18	-6	17	2,83	12	24,08	48,17	0,000
1	1	4	3	7	-13	-8	-1,33	8	8,00	16,00	0,003
2	1	2	3	5	-14	8	1,33	24	2,67	5,33	0,046
0	2	11	12	23	4	7	1,17	36	1,36	2,72	0,133
1	2	5	5	10	9	6	1,00	24	1,50	3,00	0,117
2	2	4	5	9	12	-2	-0,33	72	0,06	0,11	0,747

12. ALLEGATO 4 – CONFRONTO REGRESSIONE LOGISTICA E LINEARE

```
import numpy as np

import pandas as pd

from sklearn import linear_model as lm

from sklearn.preprocessing import PolynomialFeatures

from sklearn.model_selection import train_test_split as split


'''in dati da excel'''

dati=pd.read_excel('dataset.xlsx')

V=dati[['k','m']]

y=dati['c']


'''pre regressione logistica'''

Vadde, Vtest, yadde, ytest = split(V,y,test_size=0.2)

Vadde=np.array(Vadde)

yadde=np.array(yadde)

yadde_=[[0]]*len(yadde)

yadde_=np.array(yadde_)

for i in range(len(yadde)):

    yadde_[i,0]=yadde[i]


'''regressione logistica'''

r=lm.LogisticRegression(max_iter=10000000,multi_class='multinomial',solver='newton-
cg').fit(Vadde, yadde) #Newton conjugate gradient algorithm

ncg=r.predict(Vtest)

r=lm.LogisticRegression(max_iter=10000000,multi_class='multinomial',solver='lbfgs').fit(V
adde, yadde) #limited Broyden – Fletcher – Goldfarb – Shanno algorithm

lb=r.predict(Vtest)

r=lm.LogisticRegression(max_iter=10000000,multi_class='multinomial',solver='sag').fit(Vad
de, yadde) #stochastic average gradient descent algorithm

sag=r.predict(Vtest)

r=lm.LogisticRegression(max_iter=10000000,multi_class='multinomial',solver='saga').fit(Va
dde, yadde) #stochastic average gradient ascent algorithm

saga=r.predict(Vtest)

for i in range(len(saga)):

    saga[i]=saga[i]+2

    sag[i]=sag[i]+2
```

```
    lb[i]=lb[i]+2
    ncg[i]=ncg[i]+2


'''salvataggio valori'''
ytest=np.array(ytest)
saga=np.array(saga)
sag=np.array(sag)
lb=np.array(lb)
ncg=np.array(ncg)
ytest_=np.array([[0]]*(len(ytest)))
saga_=np.array([[0]]*(len(ytest)))
sag_=np.array([[0]]*(len(ytest)))
lb_=np.array([[0]]*(len(ytest)))
ncg_=np.array([[0]]*(len(ytest)))
for i in range(len(ytest)):
    ytest_[i,0]=ytest[i]
    saga_[i,0]=saga[i]-ytest[i]
    sag_[i,0]=sag[i]-ytest[i]
    lb_[i,0]=lb[i]-ytest[i]
    ncg_[i,0]=ncg[i]-ytest[i]
risultato=np.hstack((ytest_,ncg_,lb_,sag_,saga_))
nomi=np.array([['valore','ncg','lbfg','sag','saga']])


conta=0
for i in range(4):
    '''pre regressione polinomiale'''
    Vadde_=PolynomialFeatures(degree=i+1, interaction_only=False,
include_bias=True).fit_transform(Vadde)
    Vtest_=PolynomialFeatures(degree=i+1, interaction_only=False,
include_bias=True).fit_transform(Vtest)
    '''regressione polinomiale'''
    r=lm.LinearRegression().fit(Vadde_,yadde) #regressione lineare minimi quadrati
ordinari
    ols=r.predict(Vtest_)
    r=lm.BayesianRidge().fit(Vadde_,yadde) #regressione lineare Bayesiana
    bay=r.predict(Vtest_)
    ols=np.array(ols)
    bay=np.array(bay)
```

```
    for k in range(len(ols)):
        ols[k]=round(ols[k]-ytest[k],0)
        bay[k]=round(bay[k]-ytest[k],0)
    ols_=np.array([[0]]*(len(ytest)))
    bay_=np.array([[0]]*(len(ytest)))
    for j in range(len(ytest)):
        ols_[j,0]=ols[j]
        bay_[j,0]=bay[j]
    '''salvataggio valori'''
    risultato=np.hstack((risultato,ols_,bay_))
    new=np.array([['ols '+str(i+1),'bay '+str(i+1)]])
    nomi=np.hstack((nomi,new))

'''salvataggio excel'''
risultato=pd.DataFrame(risultato)
risultato.columns=nomi[0]
print(risultato)
risultato.to_excel(risultati.xlsx')
```

13. RIFERIMENTI BIBLIOGRAFICI

1. Jomini, A.H. (1838). *Précis de l'art de la guerre*.
2. Azadeh, K., & De Koster, R., & Roy, D. (2019). Robotized and Automated Warehouse Systems: Review and Recent Developments. *Transportation Science*, *53*, 917–945.
3. Bogue, R. (2016). Growth in e-commerce boosts innovation in the warehouse robot market. *Industrial Robot: An International Journal*, *43* (3).
4. Monte, A. (1994). *Elementi di impianti industriali*. Edizioni libreria Cortina Torino.
5. Hoffman, A.E., & Mountz, M.C., & Barbehenn, M.T., & Allard, J.R., & Kimmel, M.E., & Santini, F., & Decker, M.H., & D'Andrea, R., & Wurman, P.R. (2014). *System and method for inventory management using mobile drive units*.
6. Traversa, M., & Fedi, C., & Garola, A. (2017). *Automated warehouses*.
7. Borshchev, A., & Filippov, A. (2004). *From System Dynamics and Discrete Event to Practical Agent Based Modeling: Reasons, Techniques, Tools*.
8. Kostrzewski, M. (2020). Sensitivity Analysis of Selected Parameters in the Order Picking Process Simulation Model, with Randomly Generated Orders. *Entropy*, *22* (4).
9. de Koster, R., & Le-Duc, T., & Roodbergen, K.J. Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, *182*, 481–501.