POLITECNICO DI TORINO

Corso di Laurea Specialistica in Ingegneria Meccanica

Modellazione e analisi di AVS/RS mediante simulazione a eventi discreti

Relatori
Prof. Franco Lombardi
Dott.ssa Giulia Bruno
Dott. Gianluca D'Antonio

Candidato
Enrico Dessì Manca

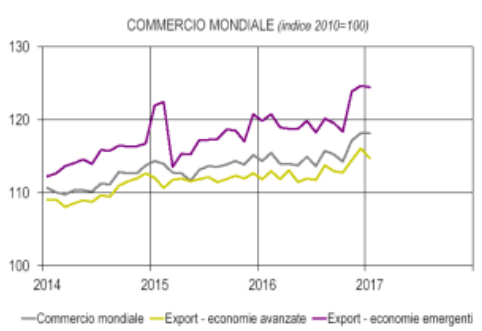
Luglio 2018

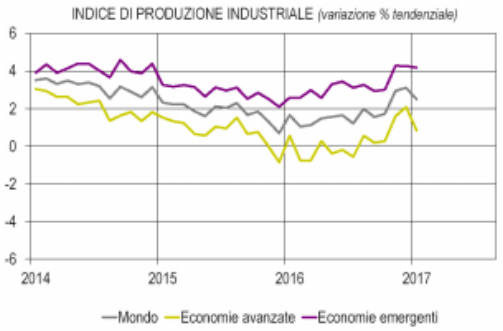
INDICE

INDICE DELLE FIGURE

1. Introduzione

Uno degli effetti più marcati della globalizzazione che ha caratterizzato i decenni passati è stato l'aumento costante dei volumi di merce esportata ogni anno dalla maggior parte dei paesi del mondo. Come evidenziato dal Centro Europa Ricerche[1], che da anni contribuisce al monitoraggio della macroeconomia mondiale per conto di istituzioni e società finanziarie, il commercio tra differenti aree del mondo è in costante crescita, con una tasso di produzione mondiale di beni di consumo sostanzialmente stabile.





In un mercato sempre più saturo di prodotti, uno dei pilastri della competitività è la capacità di fornire prodotti diversificati e personalizzati per ogni categoria di acquirente.

Per competere sul mercato le aziende di ogni settore sono portate a rilasciare ogni anno prodotti tecnologicamente più avanzati o creare varianti per soddisfare esigenze sempre più specifiche.

L'aumento dei costi di trasporto sta iniziando in ogni caso ad incidere sull'efficacia della delocalizzazione, la tendenza negli ultimi anni è infatti quella di tornare ad una produzione più vicina al territorio di vendita, gestita in maniera più efficiente.

La chiave per essere competitivi è l'abbattimento dei costi; l'impiego di tecnologie più economiche, l'utilizzo di materiali differenti o la

[1] https://www.centroeuroparicerche.it

delocalizzazione dipendono dagli obiettivi dell'azienda stessa, ma qualsiasi politica produttiva venga impiegata esiste un fattore di risparmio comune: la riduzione dei tempi. Tendenzialmente le realtà produttive vengono delocalizzate puntando sul basso costo della manodopera; con questa idea di base viene meno l'esigenza di accorciare i tempi di produzione e le spese logistiche. Un approccio differente invece consiste nel fabbricare sul territorio prodotti di qualità superiore, con un investimento iniziale certamente più elevato, ma puntando a ridurre i tempi produttivi e di movimentazione e andando così a compensare il maggior costo della manodopera.

Per venire incontro alle recenti logiche produttive ed alla richiesta sempre maggiore di prodotti nuovi da parte dei mercati, il settore della logistica ha subito radicali cambiamenti. A partire dall'approvvigionamento delle scorte fino alla consegna al consumatore, l'informatica e l'automazione sono diventate parte integrante della catena logistica.

In questa tesi verrà focalizzata l'attenzione sui magazzini automatici, apice dei sistemi di stoccaggio moderni. Lo scopo di questa tesi è valutare la possibilità di effettuare simulazioni efficaci sulle performance raggiungibili da questi sistemi in molteplici configurazioni, al fine di fornire alle aziende produttrici un sistema di previsione e preventivazione.

In particolare verranno trattati i sistemi AVS/RS (Autonomous Vehicle Storage and Retrieval System), sistemi di movimentazione introdotti negli ultimi anni come alternativa ai sistemi dotati di trasloelevatore. Sono caratterizzati da un livello di complessità superiore, in quanto non risulta più sufficiente analizzare il comportamento di un oggetto singolo ma dell'interazione tra sistemi multipli. Come caso studio verrà analizzato il sistema ESMARTSHUTTLE prodotto dall'azienda Eurofork[2].

[2] www.eurofork.com

Il problema che incontrano tutti i produttori di questi sistemi è la necessità di sviluppare uno strumento che, date alcune richieste del cliente, consenta di scegliere in modo rapido la soluzione migliore e garantirne l'efficacia.

Data la moltitudine di soluzioni realizzabili, attualmente non esistono strumenti di analisi generali ma solamente strumenti di calcolo sviluppati per la valutazione di casi standard, difficilmente adattabili a casistiche particolari.

Lo scopo di questa tesi è il valutare l'utilizzo di un software di simulazione ad eventi discreti per realizzare un modello di analisi flessibile per magazzini AVS/RS. Questi software studiano l'evoluzione temporale di un sistema secondo risposte definite ad eventi predeterminati.

Effettuando molteplici simulazioni si cercherà di individuare andamenti e variabili che influenzano maggiormente le prestazioni, per ricavare in modo teorico uno strumento predittivo.

Nel primo capitolo è presente un breve riassunto dell'evoluzione tecnologica dei sistemi di immagazzinamento, dai tradizionali carrelli a forche agli attuali AVS/RS, argomento di questa tesi, con il loro funzionamento.

Verrà in seguito presentato lo strumento di simulazione scelto per svolgere questo lavoro, seguito da tutti i passi che sono stati necessari allo sviluppo completo del modello.

Per dimostrare i risultati ottenuti sono successivamente riportati i principali confronti prestazionali effettuabili, con lo scopo di esporre al lettore i vantaggi ottenibili dalla simulazione ad eventi discreti.

Per completezza, vengono in appendice riportati i principali e più complessi script realizzati, utili come spunto per eventuali progetti futuri.

2. Background

Nel corso dei decenni i magazzini hanno subito continue evoluzioni per seguire le esigenze delle industrie ed il progresso tecnologico.

Il magazzino più semplice ed economico nelle realtà industriali è quello più classico, composto da pallet disposti ordinatamente in corridoi, impilati ove possibile, movimentati grazie a carrelli a forche (comunemente detti *muletti*) e transpallet comandati da operatori.

Proprio i carelli sono il mezzo che ha segnato il punto di svolta nella logistica di magazzino; le loro evoluzioni hanno consentito di realizzare magazzini con scaffalature di altezza sempre maggiore e realizzare strutture molto più compatte.

Per ridurre gli spazi di manovra sono state introdotte varianti sia nella struttura dei carrelli che nelle forche, modificando le classiche forche frontali

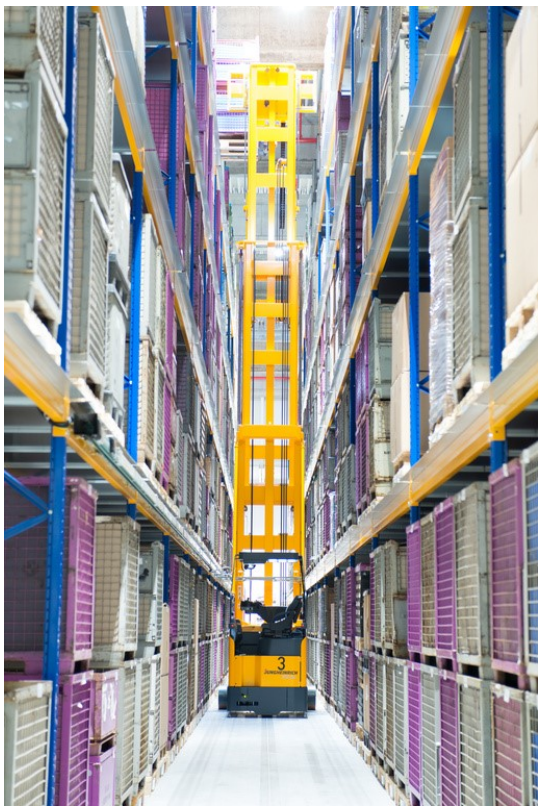


Figura 2.1 – Magazzino con carrello bilaterale

in forcole telescopiche, carrelli laterali, bilaterali e trilaterali, si sono eliminati gli spazi necessari alle manovre tra le scaffalature.

Un'evoluzione parallela è stata determinata dall'introduzione dell'informatica all'interno dei magazzini. Dapprima le UdS (Unità di Stoccaggio) sono state dotate di sistemi di riconoscimento (codici a barre, Q-Code o chip di riconoscimento) consentendo una gestione intelligente dell'inventario. Conoscendo il posizionamento esatto di ogni prodotto, è stato possibile dotare i carelli di sistemi di guida autonoma.

Esistono diverse categorie di carrelli a guida autonoma (AGV, Automatic Guided Vehicle), che si distinguono, oltre che ovviamente per la struttura del carrello, anche per i sistemi di guida del veicolo stesso.

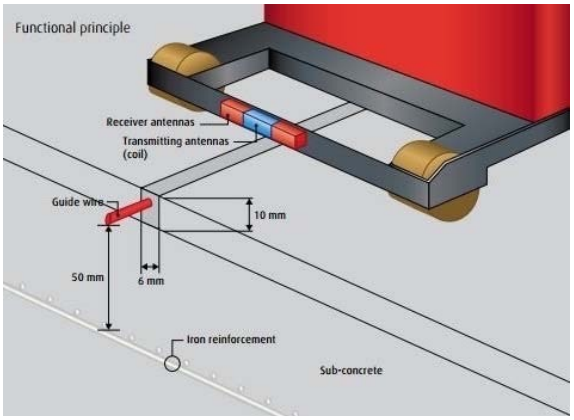


Figura 2.2 – Sistema di guida autonoma induttiva

I primi tentativi di guida automatica sono stati i carrelli a guida induttiva: sfruttando cablaggi posati a filo del piano di movimento si possono tracciare dei percorsi riconoscibili dal veicolo grazie ad appositi sensori che riconoscono il campo magnetico indotto. Utilizzando dei semplici relè è possibile creare dei percorsi di qualsiasi forma, guidando il carrello fino alla postazione desiderata.

Si tratta di un sistema molto semplice ma richiede alcuni accorgimenti, ad esempio ha problemi se posto troppo vicino alle scaffalature in acciaio per ovvi motivi di disturbo del campo magnetico stesso.

Altri sistemi economici sono quelli a guida ottica: il metodo più semplice consiste in una telecamera che segue una linea di un determinato colore sul pavimento della struttura. Utilizzando colori differenti è possibile tracciare percorsi differenti e, con alcune limitazioni, variare i percorsi variando il colore da seguire in presenza di incroci. Rappresenta in ogni caso un sistema molto limitato, utilizzabile solo per trasporto di materiali tra stazioni prefissate in ambienti produttivi, non applicabile a magazzini.

Un sistema ottico più complesso sfrutta invece la guida laser; emettitori generano fasci di luce laser che vengono riflessi da specchi opportunamente posizionati. La triangolazione di questi segnali consente al carrello di identificare con precisione la sua posizione all'interno della struttura.

Rappresenta un sistema più complesso da progettare, implementare e gestire, pertanto trova applicazioni abbastanza limitate nelle realtà industriali.

Per applicazioni di puro immagazzinamento il sistema che ha rivoluzionato maggiormente le realtà industriali è probabilmente il trasloelevatore. Questo sistema ha consentito la realizzazione dei primi magazzini intensivi, ovvero in grado di superare i 12 metri di altezza e con corridoi molto stretti, con un coefficiente di sfruttamento superficiale molto superiore ai magazzini fino a quel momento esistenti.

Un trasloelevatore è costituito da una struttura mobile dotata di un sistema di carico/scarico delle UdC in grado di scorrere lungo una colonna verticale. Questa colonna a sua volta può traslare lungo apposite rotaie posizionate sul pavimento.

Grazie a questo sistema i movimenti longitudinale e verticale sono completamente svincolati, consentendo all'UdC di spostarsi in linea pressoché retta tra la baia di carico e la destinazione prescelta.

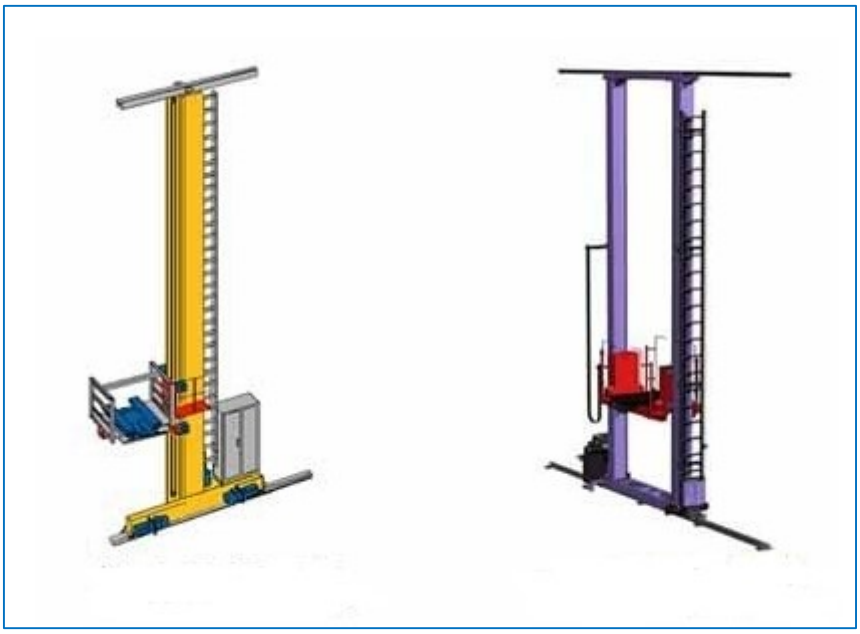


Figura 2.3 – Trasloelevatori mono e bi-colonna

Sono strutturalmente abbastanza semplici da realizzare e possono raggiungere anche i 50 metri di altezza. Nelle configurazioni bicolonna i carichi trasportabili superano anche le 4 tonnellate (potendo trasportare solitamente due UdC in contemporanea).

Nonostante le dimensioni questi sistemi possono raggiungere velocità superiori ai 2 m/s con carichi poco pesanti e sfruttano corridoi poco più larghi dell'UdC trasportata.

Come per i carelli tradizionali, è possibile realizzare sistemi a forche telescopiche a doppia elongazione, consentendo un aumento significativo della capacità di stoccaggio, ricordando però che si tratta di sistemi LIFO (Last In – First Out) in caso di scaffalature a doppia profondità.

Richiedono sempre una o due rotaie a terra ed una sul soffitto con la funzione di contrasto, per evitarne lo sbilanciamento in movimento con carico ad altezze elevate. Dato il peso, il consumo energetico per i motori risulta elevato, ma la loro capaticà di movimentazione compensa la differenza rispetto ad un singolo carrello tradizionale, oltre al risparmio sull'illuminazione stessa del magazzino, da non sottovalutare date le dimensioni di questi impianti.

Con i trasloelevatori a grande altezza sono nati anche i magazzini autoportanti: in queste strutture si sfrutta la scaffalatura stessa come supporto per il fabbricato in cui sono realizzati, aggiungendo una semplice copertura opportunamente fissata sui telai degli scaffali.

Esistono realizzazioni in scala ridotta adattati per operazioni di picking, detti miniload. Questi sistemi non movimentano pallet ma cassette di dimensioni ridotte, e servono direttamente gli operatori per la composizione di ordini misti. Raggiungono velocità più elevate in magazzini di dimensioni ovviamente ridotte.

Il progresso tecnologico successivo consiste nell'introduzione degli shuttle: il trasloelevatore non si occupa più del trasporto e deposito diretto dell'UdC. Si sostituisce alla scaffalatura tradizionale una a canali e lo stoccaggio all'interno dei singoli canali è affidato a delle navette che vengono inserite all'ingresso del canale stesso dal trasloelevatore.

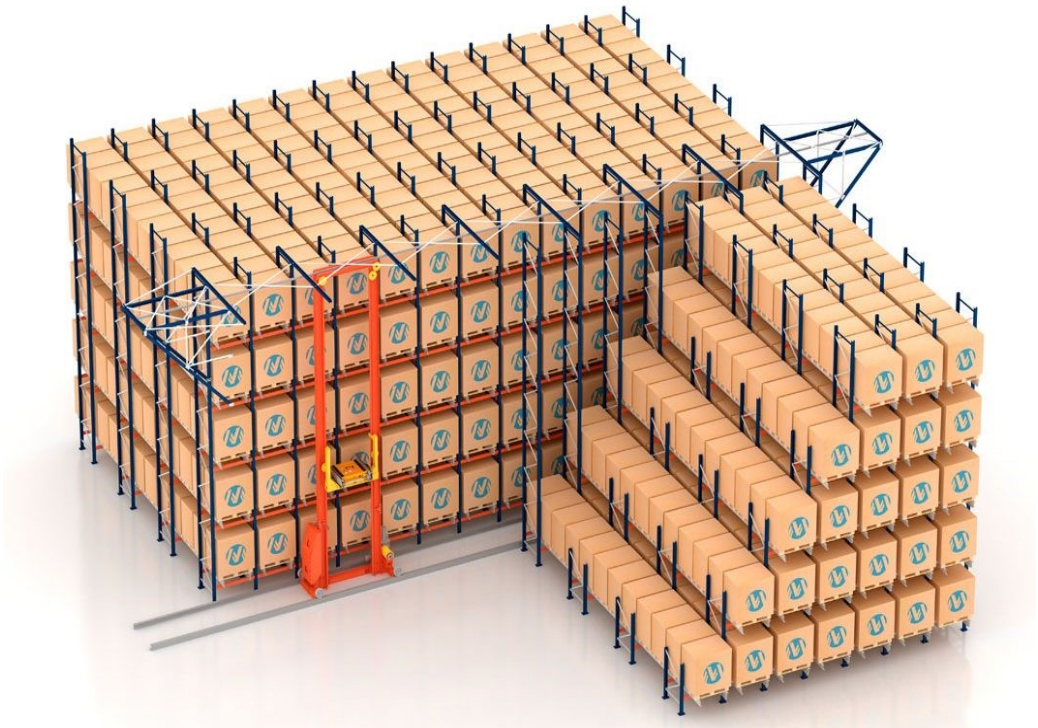


Figura 2.4 - Schema di magazzino servito da trasloelevatore con shuttle

Con questa soluzione è possibile ottenere un magazzino con un coefficiente di sfruttamento volumetrico estremamente elevato, in quanto è richiesto solamente un corridoio della larghezza necessaria al trasloelevatore.

L'utilizzo di una navetta intermedia rende il sistema più lento rispetto ad un trasloelevatore classico, ma risulta economicamente molto vantaggioso in quanto si riduce drasticamente il numero di trasloelevatori da acquistare a parità di UdC gestibili.

Gli shuttle sono tipicamente alimentati a batteria ma sono previste all'interno della scaffalatura delle postazioni di ricarica dedicate. Utilizzandone un numero sufficiente in ogni corridoio è possibile lavorare senza interruzioni alternando utilizzo e ricarica tra le navette disponibili.

Con questa tipologia di scaffalature è conveniente stoccare UdC dello stesso tipo all'interno di uno stesso canale (non è ovviamente possibile prelevare pallet dietro ad altri), risulta talvolta vantaggioso utilizzare un lato come ingresso e l'altro come uscita per mantenere il sistema FIFO (First In – First Out), se si utilizzano più corridoi in parallelo.

Come detto in precedenza, il collo di bottiglia nel funzionamento di questi sistemi è proprio il trasloelevatore: raggiunge velocità elevate, limitate solamente dalla stabilità dei prodotti sull'UdC stessa, ma deve muoversi in magazzini di dimensioni notevoli.

Si è cercato quindi di sostituire il trasloelevatore con un sistema più rapido e meno limitato: a questo scopo sono nati i magazzini automatici gestiti con shuttle e satelliti.

In un sistema AVS/RS (Autonomous Vehicle Storage and Retrieval System) i veicoli autonomi compiono percorsi rettilinei all'interno di uno o più corridoi di stoccaggio presenti nella scaffalatura, mentre gli ascensori, installati in prossimità dell'inizio dei corridoi della scaffalatura, eseguono i movimenti verticali tra i vari livelli.

Generalmente richiedono una scaffalatura provvista di rotaie su ogni livello per consentire lo spostamento delle navette.

Figura 2.6 - Cuby, prodotto da SSI Schafer

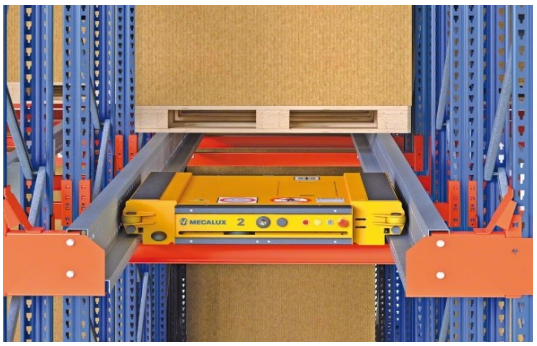


Figura 2.5 - Shuttle di produzione Mecalux

Ne esistono differenti versioni a seconda della tipologia di UdC trasportata, che deve essere in ogni caso di dimensioni standardizzate. Sono in grado di trasportare sia cassette che pallet, ma il posizionamento all'interno della scaffalatura avviene a distanze prefissate, pertanto non sono gestibili prodotti con sporgenze o instabili.

Per la movimentazione di pallet vengono utilizzati in accoppiamento uno shuttle e un satellite: il primo si occupa dei movimenti lungo il corridoio tra canali e lifter, mentre all'interno del canale agisce un satellite.

Il satellite viene caricato dallo shuttle (insieme al pallet, se presente) e movimentato fino al raggiungimento del canale. In seguito il satellite, grazie ad un sistema di sollevamento, carica il pallet e si porta all'interno del corridoio. Una volta uscito dallo shuttle, quest'ultimo è libero di eseguire una nuova missione; il vantaggio di questi sistemi è proprio quello di poter utilizzare nello stesso corridoio più navette in contemporanea, gestendo in simultanea numerose missioni di carico/scarico.

Esistono differenti configurazioni anche per i lifter: in comune hanno tutti in ogni caso una guida di traslazione dello shuttle al suo interno e possono essere provvisti di guide laterali per il carico scarico di satelliti e/o pallet. Ne esistono versioni di dimensione maggiore in grado di trasportare contemporaneamente due shuttle.

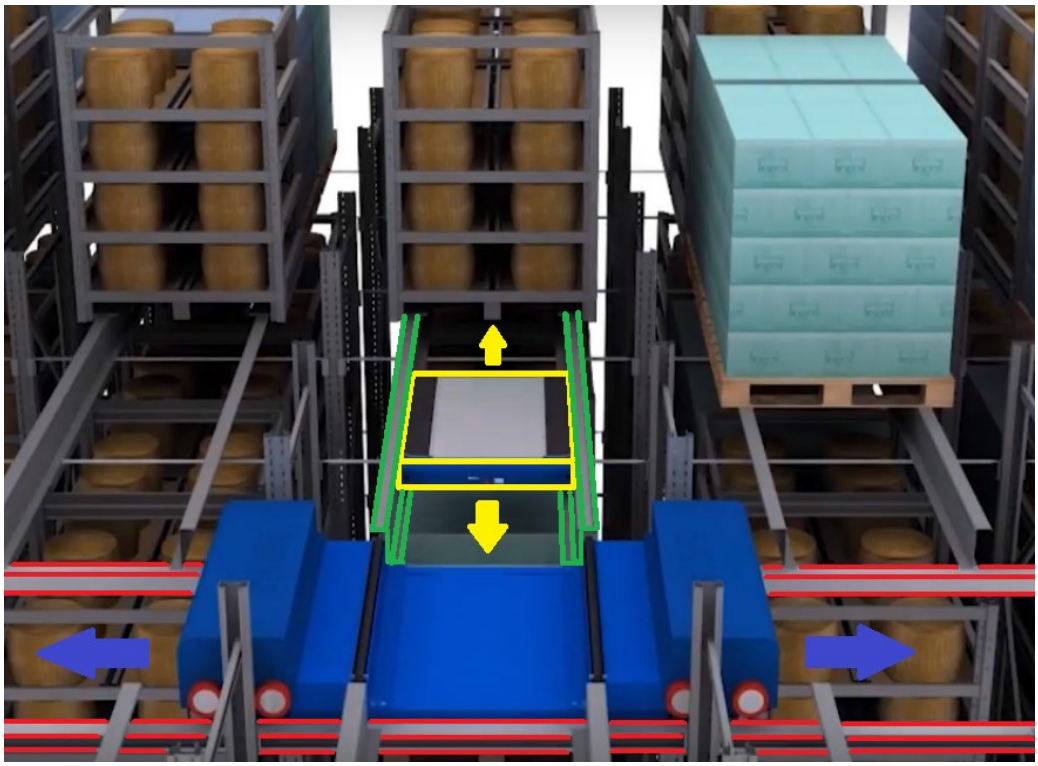


Figura 2.7 - Schema di funzionamento di un sistema shuttle-satellite

Nella figura sopra è rappresentato lo schema di movimentazione di un accoppiamento shuttle e satellite. Il primo si sposta lungo il corridoio grazie alle guide evidenziate in rosso. Il secondo si muove all'interno del canale della

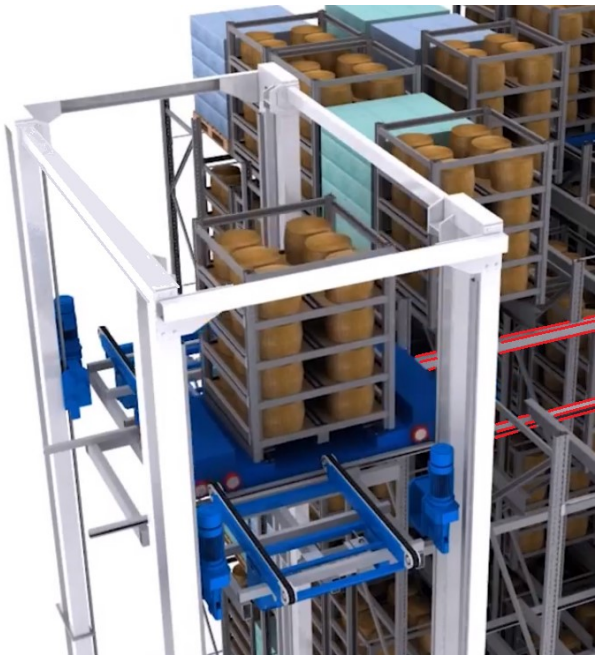


Figura 2.8 - Rappresentazione di un lifter

scaffalatura sulle rotaie evidenziate in verde. La forma dello shuttle segue quella delle rotaie, consentendo al satellite di traslare senza rallentamenti e depositare il pallet durante gli spostamenti per garantire una miglior stabilità dei prodotti.

2.1.Flexsim

La simulazione ad eventi discreti studia l'evoluzione temporale di un sistema composto da elementi mutualmente dipendenti; le interazioni tra questi elementi formano un flusso dinamico. La programmazione di questo flusso consente di simulare un comportamento verosimile di una situazione in funzione di alcune variabili.

L'evoluzione temporale si basa su decisioni logiche e variabili statistiche. In altre parole, è necessario inserire al suo interno tutti gli elementi che concorrono alla simulazione specificandone, oltre alla geometria, anche le relazioni con altri elementi e le decisioni da effettuare. Un esempio può essere l'utilizzo in parallelo di macchinari per lavorazioni anche differenti; le richieste di componenti seguiranno una legge statistica, l'efficienza di ogni singola macchina o operatore un'altra, e il software è in grado di eseguire simulazioni per lunghi periodi di tempo (ad esempio un turno di lavoro) e determinare eventuali colli di bottiglia nella logica utilizzata.

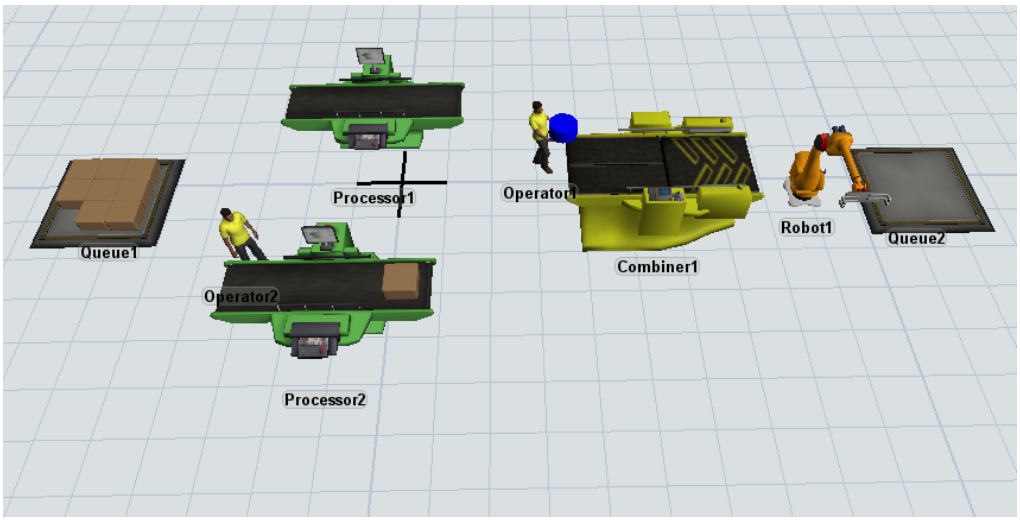


Figura 2.9 – Ambiente di modellazione FlexSim

Vengono utilizzati per testare layout organizzativi differenti e per studiare il flusso delle materie (o persone) all'interno di una struttura.

In quest'ottica, l'idea è stata di sfruttare queste potenzialità con i magazzini automatici, utilizzando uno strumento relativamente semplice per testare sistemi complessi.

Esistono molti software commerciali per la simulazione ad eventi discreti, da generici quali Arena[3] sviluppato da Rockwell Automation a specialistici come Plant Simulation del pacchetto Tecnomatix[4] di Siemens.

In questo lavoro si è scelto di utilizzare Flexsim: questo software riesce a combinare sia la necessità di personalizzazione che la resa grafica per l'utente finale, consentendo anche di far vedere ad un eventuale cliente un rendering del magazzino che si sta vendendo durante la fase di funzionamento, addirittura espandendo il modello su misura con operatori o linee di trasporto automatico.

Questo software prevede al suo interno degli elementi preconfigurati:

- Operatori
- Macchinari di processo
- Carrelli a forche frontali
- Trasportatori a nastro o a rulli
- Scaffalature tradizionali
- Robot antropomorfi
- Trasloelevatori
- Batch di attesa

Oltre a questi, esistono degli elementi detti sorgenti, che consentono di creare ed immettere nella simulazione degli elementi. In questo caso, per esempio, una sorgente crea i pallet da inserire nel magazzino.

[3] https://www.arenasimulation.com

[4] https://www.plm.automation.siemens.com/it/products/tecnomatix/

2.2. Caso d'uso – Il sistema ESS

In questo lavoro si è presa in esame la soluzione ESMARTSHUTTLE (ESS) sviluppata dall'azienda EuroFork[5]. La Eurofork s.r.l. è un'azienda italiana nata e specializzatasi per produzione di forcole telescopiche per i maggiori produttori mondiali e, negli ultimi anni, con la divisione ESMARTSHUTTLE si è aperta allo sviluppo di navette per la gestione di magazzini multiprofondità.

Nei sistemi tradizionali vi è un trasloelevatore che si occupa di posizionare lo shuttle all'ingresso del canale e quest'ultimo si occupa di posizionare il pallet nella postazione prestabilita.

Figura 2.10 – Trasloelevatore tradizionale

Nel sistema ESMARTSHUTTLE invece vi è un semplice lifter in testa alla scaffalatura che colloca lo shuttle al piano corretto; lo shuttle, scorrendo su rotaie nel corridoio, raggiunge il canale desiderato e lì vi è il disaccoppiamento di un satellite. In questo caso il satellite, raggiunto il canale, si prende carico del pallet e lo trasporta all'interno del canale, svincolandosi dallo shuttle che può intraprendere la missione successiva andando a prendere uno degli altri satelliti liberi.

L'investimento per un sistema AVS/RS è superiore rispetto ad un trasloelevatore classico: per la colonna del traslo è infatti sufficiente prevedere una o due rotaie fissate sul pavimento e una controrotaia sul soffitto per la

[5] www.eurofork.com

guida ed il bilanciamento. Per gli shuttle invece è necessario che tutta la scaffalatura sia provvista di apposite rotaie.

Figura 2.11 – Sistema ESMARTSHUTTLE

Anche le navette, seppur singolarmente più economiche, rappresentano una spesa da valutare accuratamente: sovrastimare il numero di shuttle o satelliti necessari comporta un elevato investimento iniziale senza trarne il beneficio desiderato, sottostimarne il numero invece porta a prestazioni inferiori a quelle necessarie. A tal proposito però va precisato che sono sistemi modulari, che non richiedono modifiche per l'aggiunta di navette supplementari.

Grazie all'assenza del trasloelevatore è possibile far lavorare in contemporanea più shuttle su più livelli; si può arrivare al limite di utilizzare una singola navetta per ogni livello ed utilizzare lifter ai lati della testata su cui scaricare direttamente i pallet. L'unico limite sta nel non far incontrare due shuttle lungo un corridoio sullo stesso livello.

Come tutti i sistemi AVS/RS, la soluzione ESS può adattarsi a molteplici configurazioni e personalizzazioni. Per esempio, il trasferimento di un pallet da una baia di carico allo shuttle può essere effettuato sia da un satellite che in modo autonomo, tramite semplici rulli o nastri motorizzati.

La presenza di più navette svincolate tra loro consente di eseguire qualsiasi tipologia di missione in contemporanea alle altre. Con missione si intende:

- Prelievo UdC da baia di carico e deposito nel magazzino;
- Prelievo UdC dal magazzino e consegna alla baia di scarico prescelta;

- Movimentazione interna tra due canali differenti (o riordino dello stesso).

La scaffalatura a canali è di per se un sistema che vincola a logiche di stoccaggio LIFO (Last In – First Out), a meno di non sfruttare un ingresso del canale per il deposito e l'altro per il prelievo, in tal caso è possibile un utilizzo FIFO (First In – First Out).

2.2.1. Lo shuttle

Lo shuttle è l'elemento di base del sistema. Si tratta di una navetta che ha la sola capacità di scorrere lungo un asse (che identificheremo come l'asse **x**).

Esiste sia nella versione alimentato a batteria che in quella con alimentazione tramite bus bar. La prima rende più costosa la navetta e obbliga a pause per

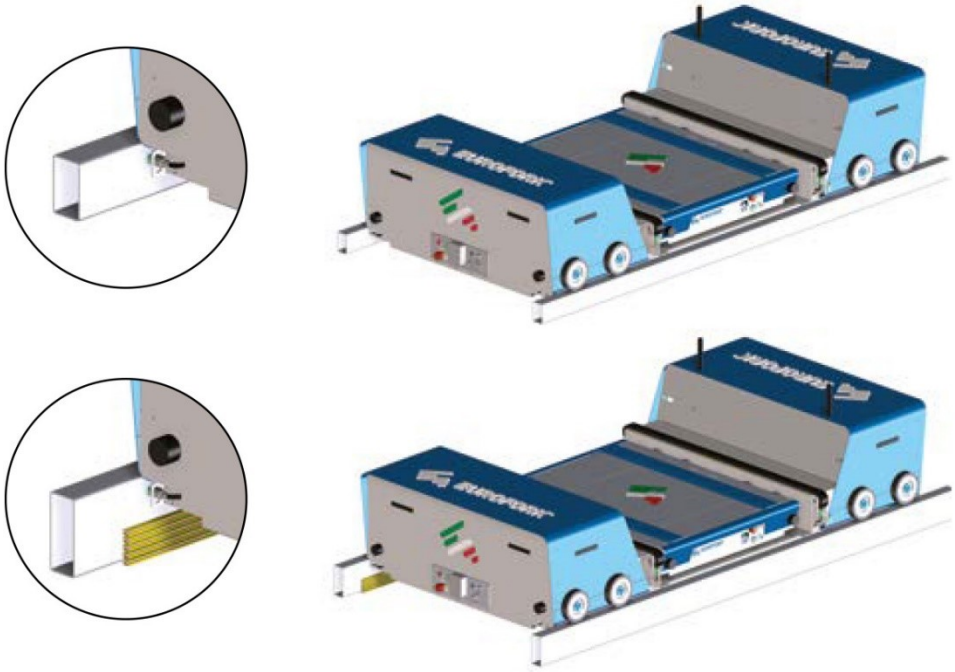


Figura 2.12 - Shuttle con alimentazione a batteria (sopra) e tramite bus bar (sotto)

ricaricare il sistema. Il secondo è più indicato per ambienti a temperatura controllata o qualora i carichi siano elevati, ma aumenta il costo della scaffalatura e ne rende più difficile l'integrazione in strutture già esistenti.

Figura 2.13 – Struttura di uno shuttle prodotto da Eurofork

Lo scopo dello shuttle è quello di trasportare il satellite dal lifter alla baia in cui depositare/prelevare il carico. Durante i tragitti il satellite appoggia il pallet sullo shuttle in modo da limitare lo sforzo ed aumentare la stabilità.

Lo shuttle possiede un sistema autonomo di movimentazione del pallet, consentendo operazioni di prelievo dalle baie di carico senza l'ausilio del satellite: in questo modo si velocizzano eventuali operazioni di carico/scarico in un canale con all'interno già un satellite, risparmiando un viaggio.

2.2.2. Satellite

Il satellite ha il compito di movimentare il pallet all'interno dei canali del magazzino e, qualora non ci fosse un trasportatore automatico fino al lifter, si occupa anche del prelievo del pallet dalle zone di prelievo e consegna.

La sua forma gli consente di spostarsi al di sotto dei pallet scorrendo su rotaie, che sovente sostengono direttamente l'UdC una volta depositata. Il carico e lo scarico avvengono attraverso un sistema di sollevamento della pedana superiore.

Figura 2.14 – Satellite prodotto da Eurofork

Figura 2.15 – Satellite durante una fase di trasporto

3. Modello di simulazione

La descrizione del modello in questa trattazione non approfondirà tutti gli aspetti informatici necessari al suo funzionamento, ma solo la logica utilizzata.

FlexSim non prevede al suo interno elementi costituenti un magazzino automatizzato a shuttle. Sono presenti solamente persone, macchinari di processo e alcuni sistemi AGV (Veicoli a guida automatica). L'unico elemento sfruttabile direttamente è stato l'ascensore, utilizzato come lifter per gli shuttle. La scaffalatura esistente nel software è di tipo tradizionale, ma con qualche forzatura è stato possibile sfruttarla per formare un magazzino a canali.

Il primo passo è stato quindi realizzare una libreria personale con shuttle e satelliti funzionanti, a partire da elementi base messi a disposizione per questo fine. In questi elementi è possibile inserire i parametri di spostamento (velocità, accelerazioni) ma vista la complessità del sistema non si sono potute integrare logiche di movimento pre-programmate, quindi tutti i movimenti vengono specificati da codice nel seguito.

Come detto prima, la scaffalatura esistente nel software è di tipo tradizionale, e ha il grosso limite, ovviamente, di non gestire elementi in movimento al suo interno. Imitare un canale usando una scaffale con una profondità superiore non è stato possibile in quanto non riempie lo scaffale dal fondo, come ovvio che sia, e si perde la dinamicità del sistema, ad esempio l'inserire e prelevare da entrambi i lati del canale.

Per superare questo limite si è utilizzato un blocco di scaffali a singola profondità: mettendo vicini 10 scaffali identici, si ottiene una scaffalatura con canali da dieci posti pallet e si consente di inserire a piacere l'UdC nella locazione desiderata. Si riesce anche ad imitare la gestione a matrice che questi sistemi nella realtà utilizzano (ogni locazione è identificata da coordinate spaziali x, y e z).

Di grande aiuto allo svolgimento di questa tesi è stata la nuova funzione di FlexSim 16 detta “ProcessFlow”, che consente all’utente di gestire il flusso di ogni missione tramite diagrammi di flusso.

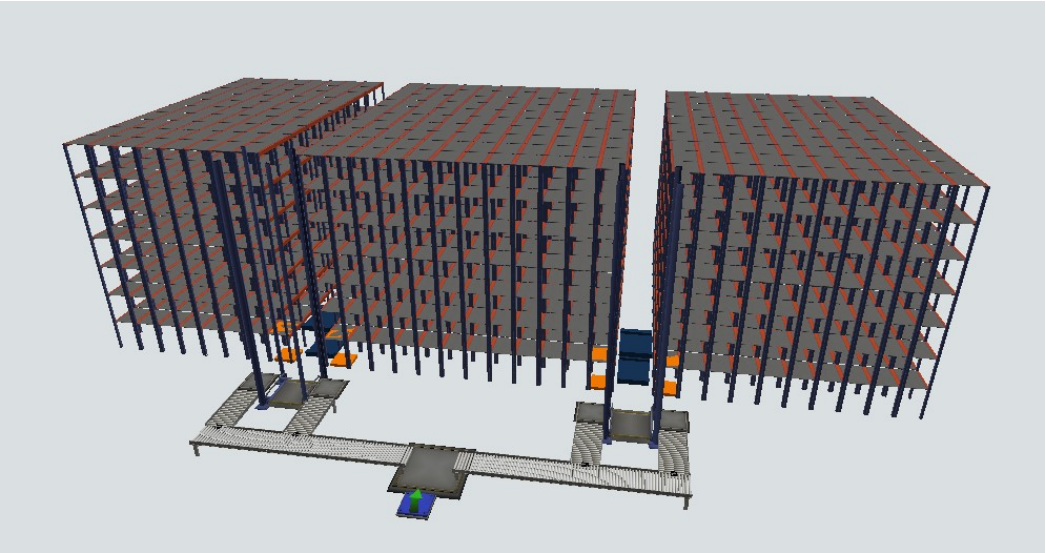


Figura 3.1 – Esempio di magazzino realizzabile

Sono tuttavia necessarie alcune semplificazioni per uniformare e rendere gestibile le simulazioni.

La prima riguarda la geometria: sebbene sia facilmente realizzabile qualsiasi configurazione, il modello generale di simulazione prevede scaffalature uniformi, con stesso numero di canali, posti per canale e numero di livelli. Dovendo effettuare migliaia di simulazioni per popolare il database statistico è stato necessario ridurre la complessità delle strutture.

La seconda riguarda la gestione dei livelli: sebbene nella realtà sia previsto, in questo lavoro è possibile far operare un solo shuttle per livello, in modo da evitare conflitti e collisioni senza introdurre pesanti controlli che rallenterebbero di molto la simulazione.

Una terza è sul sistema di prelievo: questi sistemi sono in gran parte utilizzati per lo stoccaggio di beni ad alto indice di rotazione, pertanto non è troppo restrittivo limitarsi ad un funzionamento di tipo FIFO (First In – First Out).

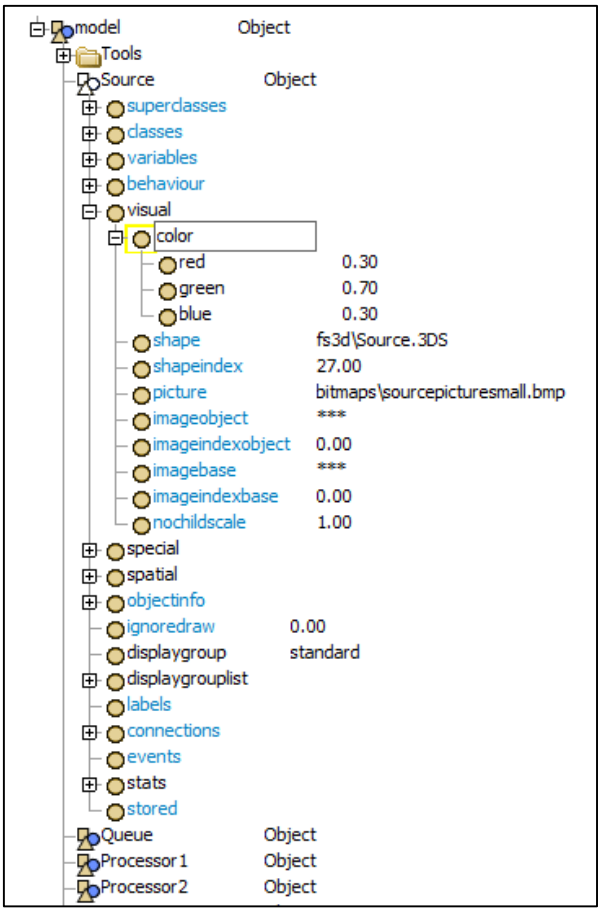


Figura 3.2 – Struttura del Treenode

Tutto il software è basato su una struttura di immagazzinamento delle informazioni ad albero. L'albero è composto da nodi, ognuno dei quali contiene un solo tipo di informazione e può a sua volta contenere al suo interno altri nodi.

Possono essere nodi contenenti semplici dati numerici o nodi riferiti ad oggetti, con all'interno una struttura di variabili necessarie alla definizione completa dell'oggetto (forma, colore, posizione, comportamento, contenuto, classe, statistiche, ect).

Per realizzare e gestire la struttura è stato necessario realizzare codici che modificano direttamente le informazioni in questi nodi. In questo modo si perde purtroppo la facilità di comprensione delle azioni effettuate, ma si rende immediata la costruzione di una qualsiasi geometria del magazzino.

Si distinguono due grosse categorie di nodi:

- Oggetti
- Dati

I nodi oggetto sono, ovviamente, oggetti fisici presenti nel modello simulato. Posseggono al loro interno una serie nodi di proprietà relativi a forma, posizione, statistiche, proprietà di movimento predefinite e variabili che lo caratterizzano. Per esempio, un Task Executer (come può essere un operatore, un carrello, o uno shuttle in questo caso) possiede variabili di velocità, accelerazione, tempi di carico e scarico che vengono richiamate utilizzando le funzioni standard di movimento applicabili a tutti gli oggetti esecutori di missioni.

I nodi di dati invece contengono informazioni, da semplici numeri a script eseguibili richiamando quel nodo. Anch'essi possono possedere al loro interno una struttura ad albero.

Sono alberi di dati anche tabelle, liste ed interfacce grafiche. Comprendere questo concetto apre alla possibilità di editare in automatico tabelle e liste di elementi con istruzioni semplici, sfruttando un linguaggio di programmazione semplificato (FlexScript) o linguaggio C (da cui il linguaggio sviluppato per questo software in ogni caso deriva).

L'edit delle caratteristiche attraverso i treenode rende questo software adatto al genere di personalizzazioni necessarie per lo svolgimento di questo lavoro.

Per la costruzione della libreria ESMARTSHUTTLE gli elementi shuttle e satellite sono stati modellati con un software esterno ed importati in Flexsim passando dall'interfaccia STL. Per non appesantire la simulazione grafica ovviamente il livello di dettaglio è estremamente basso. Forma e dimensioni non influiscono in nessun modo sul risultato delle simulazioni

Entrambi sono stati creati come BasicTE (Basic Task Executer), elemento nativo di Flexsim nato per essere agevolmente configurabile.

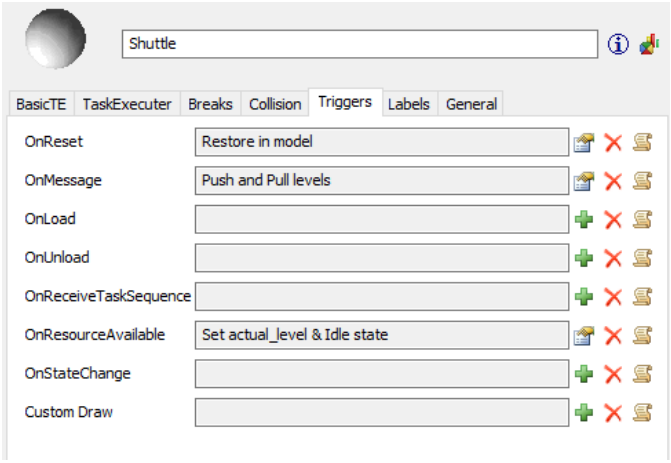


Figura 3.3 – Configurazione di uno shuttle

Tutti gli oggetti in Flexsim possiedono delle caratteristiche comuni:

- Velocità, accelerazione, carico massimo, latenze di risposta;
- Trigger, ovvero azioni (o script) eseguite in determinate situazioni;
- Label, etichette contenenti informazioni sull'oggetto stesso.

Su elementi non pre-configurati quali i BasicTaskExecuter è possibile inserire manualmente tutte le informazioni di accelerazione e tempi d'arresto, velocità massima, ma non il come si muove. Lo spostamento di persone o veicoli è già preimpostato secondo traiettorie, quindi per questi elementi deve essere creato per coordinate.

La descrizione del movimento risulta semplificata in quanto ogni elemento si muove secondo una sola coordinata: il lifter si muove linearmente solo lungo **z**, lo shuttle lungo **x** e i satelliti lungo **y**.

Il funzionamento è affidato interamente al ProcessFlow, tranne per alcuni trigger necessari al suo reset o all'assegnazione dei livelli.

Per la raccolta delle statistiche è stato inoltre necessario creare un set di statistiche alternativo a quello del software.

3.1. Generazione del modello

Flexsim è basato sulla creazione dell'ambiente di simulazione tramite interfaccia grafica, con un comodo sistema drag and drop degli elementi e il collegamento tra loro tramite porte di ingresso/uscita.

In questa tesi si è dovuto scegliere invece un sistema più complesso ma gestibile: gli elementi vengono creati direttamente nel treenode del modello, tramite uno script che imposta automaticamente posizione, dimensione e caratteristiche della scaffalatura, del sistema ESMARTSHUTTLE e tutti gli elementi connessi.

Con semplici cicli interativi si possono creare istantaneamente centinaia di scaffali con la configurazione richiesta.

Tutti i dati necessari (dimensioni, numero di shuttle, etc) sono inseriti tramite un'interfaccia grafica appositamente realizzata (come descritto nel capitolo successivo) e salvati in appositi nodi di dati dell'albero descritto in precedenza.

Un codice principale viene eseguito e, sfruttando quei dati, crea il magazzino desiderato; al termine è possibile apportare modifiche manuali (eliminare o aggiungere navette) senza compromettere il funzionamento del modello.

Per comodità di gestione delle simulazione sono state create varie interfacce grafiche per gestire comodamente tutte le variabili richieste.

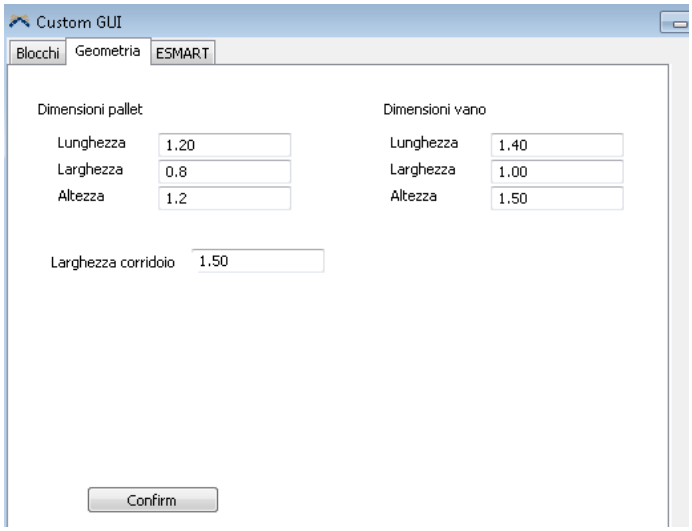


Figura 3.4 - Esempio di GUI

Esiste uno strumento dedicato che consente di creare GUI personalizzate in modo rapido e abbastanza intuitivo.

Ogni interfaccia è collegabile a nodi in cui salvare le informazioni inserite. Questi valori sono utilizzati nel codice del modello.

Nel proseguimento di questa sezione viene spiegato il funzionamento dello script che genera la scaffalatura e tutti gli elementi del sistema AS/RS. Non si entrerà nei dettagli per facilità di lettura ma l'intero script è riportato e commentato per intero in appendice.

L'idea di base è stata di non utilizzare uno scaffale multiprofondità (in cui non è possibile stabilire un posizionamento a coordinate) ma usare tanti scaffali a singola profondità uniti, creando in modo fittizio i canali in cui si muovono i satelliti.

Per far ciò, è necessario creare uno scaffale delle corrette dimensioni e replicarlo tante volte quante sono i posti pallet nel canale. Da qui in poi questo gruppo di scaffali verrà denominato **blocco** per semplicità, in quanto nella realtà si tratta appunto di un elemento unico.

Si creano inizialmente delle variabili per larghezza, lunghezza e altezza del posto pallet, larghezza dei corridoi in cui traslano gli shuttle, numero di blocchi. I valori sono inseriti precedentemente tramite l'interfaccia grafica e salvati nel treenode.

Per ogni blocco si vanno ad individuare il numero di baie, livelli e posti pallet per canale; fatto questo si impostano le dimensioni di uno scaffale di base, si posiziona e si copia tante volte quante sono le postazioni.

Al termine si aggiunge lo spazio per un corridoio e si ricomincia con il blocco di scaffali successivi.

Questo ciclo si ripete tante volte quanti sono i blocchi di scaffali da creare. Al termine di ogni loop vengono inseriti nel corridoio shuttle, satelliti ed il lifter sulla base delle informazioni inserite in precedenza nell'interfaccia grafica.

```
// Imposto le dimensioni della scaffalatura leggendo dal treenode
double baywidth  = getnodenum(node("/Data/Geometry/Location/Length", model));
double rackwidth = getnodenum(node("/Data/Geometry/Location/Width",  model));
double lvlheight = getnodenum(node("/Data/Geometry/Location/Height", model));
double aislesize = getnodenum(node("/Data/Geometry/aislesize", model));

// Genero una variabile che userò in seguito per posizionare i blocchi
double ypointer = y_zero;

// Inizia ora il ciclo di creazione di un blocco
for (int block = 1; block <= numofblocks; block++)
{
        int bays   = getnodenum(node(concat("/Data/Blocks/Block ", numtostring(block), "/2"), model));
        int levels = getnodenum(node(concat("/Data/Blocks/Block ", numtostring(block), "/1"), model));
        int racks  = getnodenum(node(concat("/Data/Blocks/Block ", numtostring(block), "/3"), model));

            // Creo una copia dell'elemento base
            createcopy( node("Rack", model), model);
            setnodename( last(model), "Rack0");
            treenode RACK = last(model);

// IMPOSTO LE DIMENSIONI
    setnodenum(node(">variables/sizetable/Bay1", RACK), baywidth);
    setnodenum(node(">variables/sizetable/Level1", RACK), lvlheight);
    setnodenum(node(">spatial/spatialsx", RACK), bays*baywidth);
    setnodenum(node(">spatial/spatialsy", RACK), rackwidth);
    setnodenum(node(">spatial/spatialsz", RACK), levels*lvlheight);
```

```
// CREO I LIVELLI ALL'INTERNO DEL NODO BAY
        // Devo creare il livello in sizetable, creare il nodo in cui ne verrà salvato
        // il contenuto e impostarne la posizione
        for(int j=1; j<levels; j++)
        {
            createcopy(node("/Rack0>variables/sizetable/Bay1/Level1", model),
                       node("/Rack0>variables/sizetable/Bay1", model));

            setnodename(last(node("/Rack0>variables/sizetable/Bay1", model)),
                       concat("Level", numtostring(j)));

            createcopy(last(last(node("Rack0>variables/contenttable", model))),
                       last(node("Rack0>variables/contenttable", model)),1);

            createcopy(last(last(node("Rack0>variables/locationtable", model))),
                       last(node("Rack0>variables/locationtable", model)),1);
            setnodenum(last(last(node("Rack0>variables/locationtable", model))),
j*baywidth);
        }

//DUPLICO LE BAY
        // Anche in questo caso devo farlo per sizetable, location e content
        for(int k=1; k<bays; k++)
        {
            createcopy(node("/Rack0>variables/sizetable/Bay1", model),
                       node("/Rack0>variables/sizetable", model));

            setnodename(last(node("/Rack0>variables/sizetable", model)),
                       concat("Bay", numtostring(k)));

            createcopy(last(node("Rack0>variables/contenttable", model)),
```

```
                                        node("Rack0>variables/contenttable", model),1);

                createcopy(last(node("Rack0>variables/locationtable", model)),
                                        node("Rack0>variables/locationtable", model),1);

                setnodenum(last(node("Rack0>variables/locationtable",  model)),
k*lvlheight);

        }

        // CREO LE COPIE DI RACK0 PER COSTRIRE IL BLOCCO
        for(int rack=1; rack<=racks; rack++)
        {
                createcopy(node("Rack0", model), model);
                setnodename(last(model), concat("Rack", numtostring(block), "_",
                        numtostring(rack)));

                // Il setloc deve ricordarsi anche del posizionamento dei blocchi
                // precedenti, sfruttando la variabile "ypointer"
                setloc(last(model), x_zero, ypointer+rack*rackwidth, z_zero);

        // Questi comandi sono necessari per il refresh "grafico" della sizetable
        function_s(node("/?Rack", library()), "BasicRefreshTable", last(model),
                                levels, bays, lvlheight, baywidth);
                nodefunction(node("/?Advanced>update", library));
        }
```

```
		// Aggiorno la variabile che tiene memoria della posizione tra un loop ed il seguente
		ypointer = yloc(last(model)) + aislesize/2;



// POSIZIONO LE BAIE DI CARICO
// Per ogni corridoio intermedio sono previste due baie. Vengono impostate forma e posizione, nome ed una
// una label che identifica il corridoio di riferimento. Viene impostato il massimo contenuto a 1.
		if(block<numofblocks)
		{
			createcopy(queue, model);
			setloc(last(model), x_zero-2.5, ypointer-1, z_zero);
			setsize(last(model), 1, 1, 0.1);
			groupaddmember("Queues", last(model));
			setlabel(last(model), "aisle", block);
			setnodenum(node(">variables/maxcontent", last(model)), 1);
			setnodename(last(model), concat("Queue", numtostring(block), "_", numtostring(block)));

			createcopy(queue, model);
			setloc(last(model), x_zero-2.5, ypointer+2, z_zero);
			setsize(last(model), 1, 1, 0.1);
			groupaddmember("Queues", last(model));
			setlabel(last(model), "aisle", block);
			setnodenum(node(">variables/maxcontent", last(model)), 1);
			setnodename(last(model), concat("Queue", numtostring(block), "_", numtostring(block+1)));
		}

// POSIZIONO IL LIFTER
// Per ogni corridoio è previsto un lifter in testa al corridoio.
```

```
//Il lifter è l'unico elemento del sistema AS/RS già esistente all'interno di Flexsim, ma risulta comodo
// utilizzarne una copia già configurata e salvata nella libreria ESMARTSHUTTE

if(block<numofblocks)
	{
		dropuserlibraryobject(lifter);
		setloc(last(model), -3, ypointer+ysize(lifter)/2, 0);
		setnodename(last(model), concat("Lifter", numtostring(block)));
		setlabel(last(model), "aisle", block);
		groupaddmember("Lifters", last(model));
		setresetposition(last(model));
	}



// POSIZIONO GLI SHUTTLE
// Inserisco il numero di shuttle voluti nel corridoio e gli assegno posizione e nome.
// Gli shuttle vengono inseriti su livelli differenti, a partire dal basso.
// Il nome è dato da "Shuttle" + numero del corridoio + numero del livello iniziale

	if(block<numofblocks)
	{
		for(int j=1; j<=shuttles; j++)
		{
			createcopy(shuttle, model);

			string name = concat("Shuttle", numtostring(block),
				numtostring(j));
			setnodename(last(model), name);
			setloc(last(model), 2, ypointer+ysize(shuttle)/2,
```

```
                                z_zero+(j-1)*lvlheight-zsize(shuttle));


                        setresetposition(last(model));
                        setlabel(last(model), "aisle", block);
                        setlabel(last(model), "actual_level", j);
                        groupaddmember("Shuttles", last(model));
                }
        }
```

// POSIZIONO I SATELLITI

// Analogamente posiziono il numero di satelliti voluti nei blocchi. Il nome identifica Corridoio_Blocco_Numero

```
        double x = x_zero+rackwidth/2-xsize(satellite)/2;
        if(block<numofblocks)
        {
                for(int j=1; j<=satellites; j++)
                {
                        createcopy(satellite, model);

                        setnodename(last(model), concat("Sat", numtostring(block),
                                "_", numtostring(block), "_", numtostring(j)));

                        setloc(last(model), x, ypointer-aislesize+ysize(satellite)/2,
                        z_zero+(j-1)*lvlheight-zsize(satellite));

                        setresetposition(last(model));
                        groupaddmember("Satellites", last(model));
                        setlabel(last(model), "block", block);
                        setlabel(last(model), "aisle", block);
                        setlabel(last(model), "actual_level", j);
```

```
                }

                for(int j=1; j<=satellites; j++)
                {
                        createcopy(satellite, model);

                        setnodename(last(model), concat("Sat", numtostring(block),
                                "_", numtostring(block+1), "_", numtostring(j)));

                        setloc(last(model), x, ypointer+aislesize+ysize(satellite)/2,
                        z_zero+(j-1)*lvlheight-zsize(satellite));

                        setresetposition(last(model));
                        groupaddmember("Satellites", last(model));
                        setlabel(last(model), "block", block+1);
                        setlabel(last(model), "aisle", block);
                        setlabel(last(model), "actual_level", j);
                }
        }

        ypointer += aislesize/2;
        destroyobject(RACK);
}
```

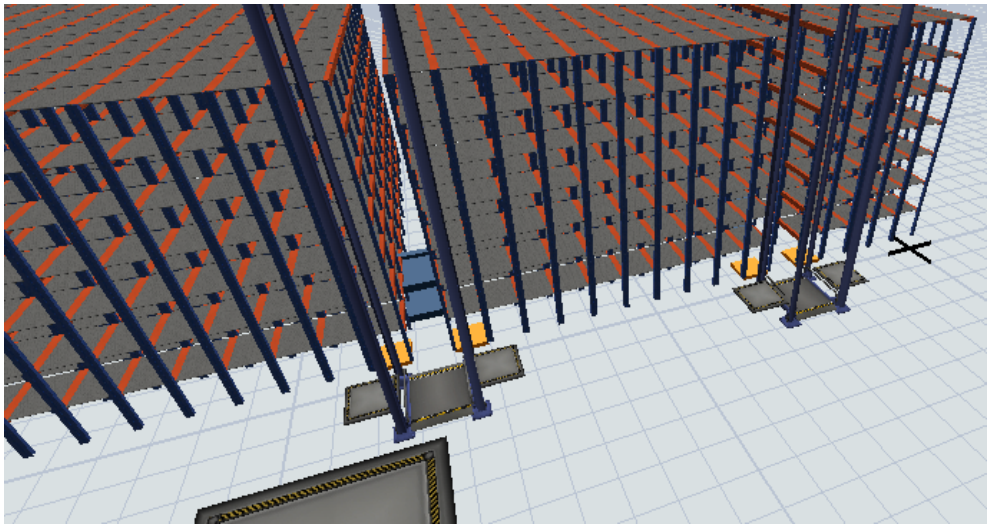


Figura 3.5 – Magazzino ottenuto con l'esecuzione del codice precedente

Al termine dell'esecuzione del codice si ottiene il magazzino completamente generato.

Tutti gli elementi sono stati assegnati a gruppi; tramite questi gruppi è possibile, come si vedrà nel seguito, richiamare ed effettuare ricerche generiche nel gruppo "Shuttles" o "Lifters", per esempio.

4. Logiche di stoccaggio

È necessario chiarire innanzitutto che non esiste una logica di stoccaggio migliore delle altre ma ogni azienda ha necessità differenti.

I principali fattori da esaminare sono la frequenza di prelievo o di stoccaggio e se avvengono in fasi sovrapposte o in momenti differenti della giornata, la variabilità dei prodotti, la giacenza media e l'indice di rotazione della merce.

Se si trattasse, per esempio, di un magazzino per prodotti alimentari finiti presso l'azienda produttrice si può presumere che lo stoccaggio avvenga durante i turni lavorativi giornalieri e che il prelievo invece sia effettuato la sera per caricare i camion diretti ai magazzini di distribuzione. In questo caso la giacenza media dei prodotti deve necessariamente essere molto bassa e l'indice di rotazione elevato vista la deperibilità del bene. Si presuppone una variabilità abbastanza elevata, altrimenti non sarebbe giustificato l'investimento per un magazzino di questo genere. Per questa azienda converrebbe un magazzino capace di riorganizzare internamente la merce nelle ore di inattività per avvicinare alle baie di prelievo la merce in programmazione per la spedizione futura.

Situazione differente invece potrebbe essere quella di un magazzino di distribuzione per una società di e-commerce. In questo caso si prevede un'elevata varietà di prodotti con una giacenza media altamente eterogenea, approvvigionamenti in grossi stock con indici di rotazione molto elevati o piccoli quantitativi con pochi ordini mensili. In questo caso un posizionamento su basi statistiche risulta impraticabile, sarà verosimilmente preferibile un magazzino molto reattivo con canali di profondità variabile, preferibilmente corti, con molti corridoi e livelli per gestire la maggior varietà possibile di prodotti e rifornire efficacemente le scaffalature destinate al picking.

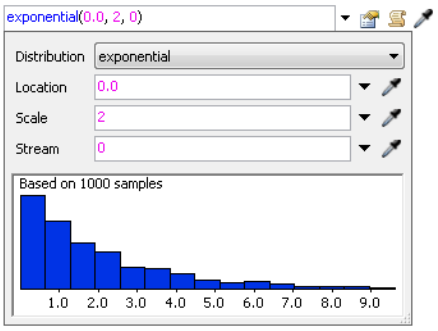


Figura 4.1 – Esempio di distribuzione

Al fine di questa tesi, è stato creato un generatore automatizzato di ordini, personalizzabile in varietà di prodotti, frequenza di consegna e numero di oggetti. Frequenza e differenziazione possono seguire una qualsiasi distribuzione statistica o essere distribuiti normalmente.

Per le simulazioni è stato introdotto un vincolo, sovente utilizzato nei magazzini reali: ogni canale può ospitare una sola tipologia di prodotto. Comunemente questi magazzini utilizzano una logica FIFO per andare incontro alle necessità di rotazione della merce, e con prodotti differenti nello stesso canale non è possibile rispettare facilmente questo principio. Di conseguenza si deve prevedere un numero di canali di molto superiore a quelli strettamente necessario a stoccare la quantità massima prevista, in quanto alcuni conterranno pochi prodotti.

Per quel che riguarda il database di simulazioni, è stato popolato da soli test con distribuzioni uniformi di merce ed immagazzinamento non prioritario, occupando il primo canale libero disponibile, senza posizionamento fisso di un determinato itemtype.

Le informazioni sul contenuto del magazzino ed alcune statistiche relative alle singole missioni sono raccolte in una tabella all'interno del modello stesso. E' strutturata in modo tale che ad ogni riga corrisponda una locazione all'interno della scaffalatura; ogni locazione è identificabile da scaffale, livello, canale e contiene informazioni su contenuto del posto, itemtype predisposto per il canale in cui si trova, tempo di ingresso del pallet, tempo di permanenza, ecc.

Questa tabella (all'interno del software definita Global Table) viene utilizzata per l'assegnazione delle posizioni ai pallet in stoccaggio e per la creazione delle missioni di prelievo tramite query in linguaggio SQL.

Di seguito viene riportato, come esempio, il codice utilizzato per assegnare ad un pallet, sulla base del suo itemtype, una posizione all'interno della scaffalatura, cercando la prima posizione libera disponibile.

```
/**ASSIGN LOCATION IN THE WAREHOUSE*/

int item_type = getlabel(token, "itemtype");

int ID = getlabel(token, "ID");

string name = getname(getlabel(token, "item"));

string channel;

int block;

int aisle;

int position;


// Cerco se ci sono canali non completi con stesso itemtype, se non ci sono prendo il primo libero

query("SELECT Channel, Block, Level, Column, Stor_Aisle FROM Warehouse WHERE Itemtype = $1 AND Item = $2 ORDER BY Priority", item_type, "");


if(getquerymatchcount()!=0)

{

        block   = getqueryvalue(1, 2);

        channel = getqueryvalue(1, 1);

        setlabel(token, "channel", channel);

        setlabel(token, "block",   block);

        setlabel(token, "level",   getqueryvalue(1, 3));

        setlabel(token, "bay",     getqueryvalue(1, 4));

        aisle = getqueryvalue(1, 5);


}
```

```
// Se non ci sono canali dedicati ne inizio ad occupare uno libero
else
{
        query("SELECT Channel, Block, Level, Column, Stor_Aisle FROM Warehouse WHERE Itemtype = 0 AND Item = $1 ORDER BY Priority", "");
        channel = getqueryvalue(1, 1);
        block   = getqueryvalue(1, 2);
        setlabel(token, "channel", channel);
        setlabel(token, "block",   block);
        setlabel(token, "level",   getqueryvalue(1, 3));
        setlabel(token, "bay",     getqueryvalue(1, 4));

        // Eseguo QUERY tra le baie di carico per identificare se una è occupata
        int flag = 0;
        string QUERY;
        QUERY = concat("WHERE pulled.Aisle = ", numtostring(block));
        if(listpull("Storage_Bays", QUERY, 0, 1)!=0)
        {
                flag += 1;
        }
        QUERY = concat("WHERE pulled.Aisle = ", numtostring(block-1));
        if(listpull("Storage_Bays", QUERY, 0, 1)!=0)
        {
                flag += 2;
        }

        // Se sono entrambe libere o entrambe occupate la assegno a caso
        if(flag == 0 || flag == 3)
        aisle = duniform(block-1, block);

        // Se è libera solo la prima
```

```
            if(flag == 1)
            aisle = block;

            // Se  libera solo la seconda
            if(flag == 2)
            aisle = block-1;

            // Se AISLE è uguale a zero, in automatico assegno la missione alla prima AISLE
            if(aisle==0)
            aisle = 1;

            // Se AISLE è uguale al numero di blocchi, assegno la missione all'ultima AISLE
            if(aisle==BLOCKS)
            aisle = BLOCKS-1;

}

            // Imposto l'Itemtype di questo canale e il corridoio di approvvigionamento
            // Per farlo, trovo quante posizioni ha il canale scelto e con un ciclo a partire dal primo
risultato
            // inserisco l'itemtype in tutte le rIghe
            query("SELECT Position FROM Warehouse WHERE Channel = $1", channel);
            int results = getquerymatchcount();
            if(results==0)
            {
                        stop();
            }

            int first_position = getquerymatchtablerow("Warehouse", 1);
            for(int x = first_position; x<first_position+results; x++)
            {
```

```
                settablenum("Warehouse", x, 8, item_type);
                settablenum("Warehouse", x, 9, aisle);
        }


// IDENTIFICO LA LOCATION
if(aisle==block) // Se riempio da sinistra parto dalla posizione 1
{
        query("SELECT Position FROM Warehouse WHERE Channel = $1 AND Item = $2 ORDER BY Position ASC", channel, "");
        position = getqueryvalue(1, 1);
}
else // Se riempio da destra parto dall'ultima posizione
{
        query("SELECT Position FROM Warehouse WHERE Channel = $1 AND Item = $2 ORDER BY Position DESC", channel, "");
        position = getqueryvalue(1, 1);
}

// Salvo le label mancanti
setlabel(token, "aisle", aisle);
setlabel(token, "position", position);


// Salvo in tabella l'item nella cella assegnata
query("SELECT Item FROM Warehouse WHERE Channel = $1 AND Position = $2", channel, position);
int row = getquerymatchtablerow("Warehouse", 1);
settablestr("Warehouse", row, 7, name);


// Assegno lo scaffale al token
```

```
treenode rack = node(concat("/Rack", numtostring(block), "_", numtostring(position)), model);

setlabel(token, "rack", rack);
```

5. Logiche di funzionamento

La selezione della miglior combinazione tra shuttle e satelliti da utilizzare per la missione rappresenta il punto cruciale per la valutazione di un magazzino automatico con sistema ESMARTSHUTTLE.

Per lo sviluppo della logica implementata in questa tesi è utile procedere per gradi di complessità

5.1. Uno shuttle e un satellite, stoccaggio

In questo primo esempio è facile prevedere quali saranno gli step eseguiti nella missione.

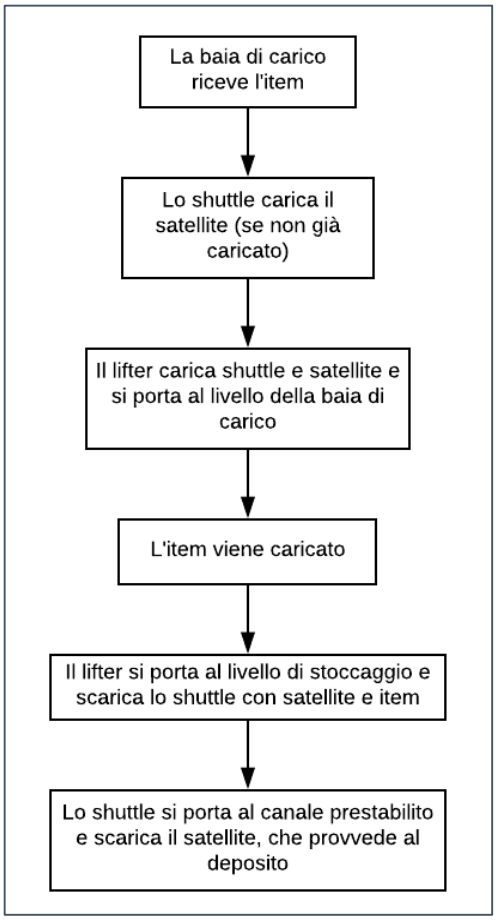


Figura 5.1 – Esempio di flusso logico

Con questa configurazione non vi sono scelte da effettuare. Lo svolgimento di una missione è lineare, shuttle e satellite si troveranno sempre sullo stesso livello.

5.2. Uno shuttle ed un satellite, ciclo combinato

Già con questa configurazione si aprono più possibilità:

- Esiste una logica prioritaria tra stoccaggio o prelievo?
- È possibile prelevare un lotto di item definito in ordine casuale tra loro?

Supponiamo che si preferisca eseguire in alternanza una missione di prelievo ed una di deposito (che con una distribuzione casuale degli ordini rappresenta forse la scelta migliore). Assumiamo verosimilmente che l'operatore addetto al deposito non sia in grado di modificare gli ordini ricevuti ma li consegni in maniera sequenziale. La coda di prelievo invece sarà probabilmente nota a lotti (ad esempio per il carico di un camion).

In questa situazione l'unità di controllo del magazzino sarà sicuramente in grado di selezionare, ove possibile, prodotti da prelevare che si trovino allo stesso livello dell'item appena stoccato, eliminando un cambio di livello.

5.3. Uno shuttle e più satelliti, ciclo combinato

In questa situazione le prestazioni migliorano tanto più nota è la coda di missioni da effettuare.

Con più satelliti viene introdotta la necessità di stimare il tempo di completamente di eventuali missioni in corso. Scegliere un satellite disponibile non necessariamente rappresenta l'alternativa migliore; infatti se lo shuttle si trova già sul livello del satellite che sta terminando una missione precedente e il prodotto da prelevare anche, conviene ovviamente aspettare che lo stesso satellite finisca l'attuale incarico e si liberi, piuttosto che effettuare un cambio di livello per caricare il satellite subito libero e tornare sul livello iniziale.

5.4.Più shuttle e più satelliti

La vera potenzialità di un magazzino con sistema ESMARTSHUTTLE ovviamente è data dall'utilizzo di molte navette, in grado di svolgere varie missioni in contemporanea. In questo caso (quello realizzato nella pratica) diventa essenziale la capacità di previsione del tempo richiesto a tutti i sistemi coinvolti, al fine di scegliere accuratamente le navette da utilizzare. Questa valutazione può essere effettuata in tempo reale, ma non può essere effettuata dal singolo shuttle.

Se ogni shuttle scegliesse direttamente quale satellite gli consentirebbe il tempo di esecuzione della missione minore, potrebbe andare a rallentare il lavoro degli altri shuttle presenti.

Nel modello sviluppato oltretutto si potrebbe creare un conflitto di acquisizione dei livelli di lavoro (un'assunzione fatta inizialmente consiste infatti nel limitare i livelli a contenere un solo shuttle).

Un buon sistema di gestione deve essere in grado di effettuare iterativamente questo genere di valutazioni. Solitamente si tratta di algoritmi a cascata multipla, in cui si valutano possibili schedulazioni di ordini e per ogni missione vengono stimati i tempi di risposta di tutte le possibili combinazioni di shuttle e satelliti. La simultaneità delle missioni limita tuttavia le combinazioni effettuabili, in quanto non tutte le navette saranno disponibili.

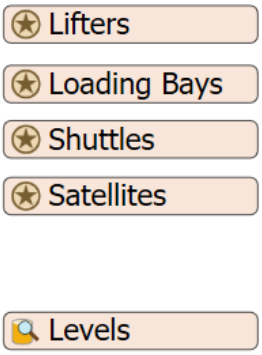
5.5.Implementazione in Flexsim

Per gestire l'intera simulazione si è sfruttato l'ambiente ProcessFlow di Flexsim. Si tratta di un ambiente di creazione di flussi decisionali strutturato a diagrammi di flusso, collegabile con un modello 3D o utilizzabile per gestire solo informazioni e dati.

Il diagramma di flusso viene seguito da un entità detta **token**. In questa tesi il token non è associato al pallet, bensì alla missione stessa. Il suo flusso determina lo stato di avanzamento dell'operazione. La missione ha inizio

all'arrivo in baia di un pallet da depositare o alla richiesta di prelievo; questo genera un nuovo token, tramite blocchi detti sorgenti.

Nella terminologia di FlexSim gli oggetti che eseguono azioni sono detti **Risorse**.



In particolare, in questo caso si identificano come risorse i lifter, le baie di carico, gli shuttle e i satelliti. Son identificate grazie ai gruppi a cui sono state assegnate in fase di creazione del modello stesso.

Le baie di carico in testa alla scaffalatura non eseguono vere e proprie azioni ma vengono comunque impegnate e disimpegnate, e rappresentano a tutti gli effetti una risorse utilizzata dal pallet (detto **flowitem**) durante il suo tragitto.

Una soluzione comoda usata per evitare conflitti sullo stesso piano tra shuttle è quella di utilizzare una lista per i livelli: la lista è divisa in sezioni (a rappresentare i corridoi) e ogni sezione contiene i livelli. In ProcessFlow è infatti possibile acquisire una voce della lista ed in seguito restituirla, esattamente come fatto con le risorse.

In questo modo uno shuttle acquisisce un piano e lo rende occupato, e solo al termine dell'utilizzo lo rilascia e ne consente l'acquisizione ad un altro shuttle.

Possono esistere più istanze di ProcessFlow attive nello stesso modello. Per esempio risulta comodo gestire in parallelo il flusso decisionale per operazioni di stoccaggio e di prelievo; entrambe le istanze sono legate allo stesso modello 3D, ne condividono le risorse e sono in grado di passarsi informazioni in tempo reale. Questo porta ad un miglioramento della facilità d'interpretazione degli schemi.

Tutte le logiche implementabili sono state costruite per step di complessità crescente; per aiutare a capire meglio il funzionamento di Process Flow e fornire una spiegazione sui principali elementi che lo compongono viene

riportato un ciclo di stoccaggio singolo, che non prevede interazione con ciclo parallelo di prelievo. In questo modo si evitano tutti i controlli di priorità tra missioni e si limitano le situazioni in cui è possibile trovare navette disponibili, eliminando il caso di navetta con satellite disponibile sul lifter dopo aver rilasciato un pallet alla baia.

Ogni ciclo di ProcessFlow inizia quando alla baia iniziale arriva un **item**. Ogni item ha un codice identificativo o informazioni utili per determinarne il punto di stoccaggio.

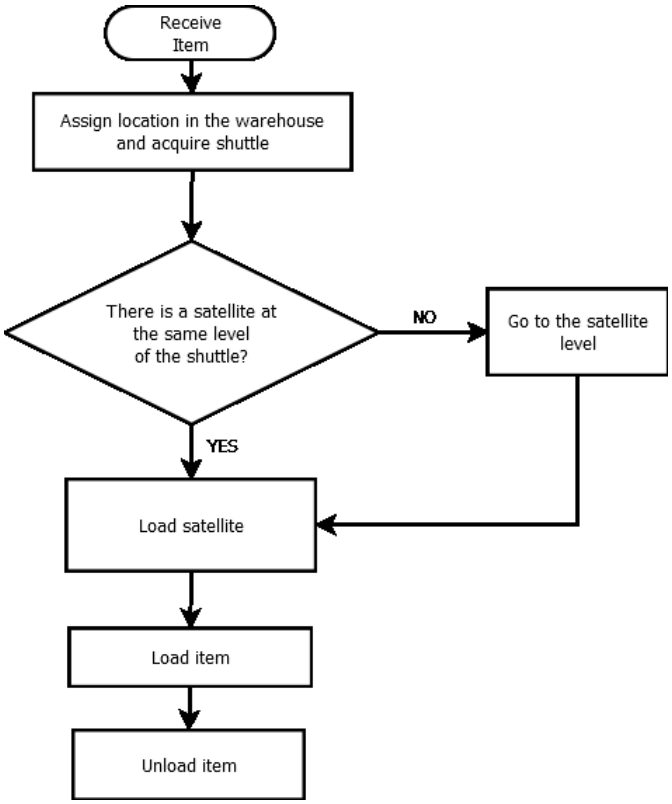


Figura 5.2 – Logica minima di stoccaggio

Sulla base della logica di stoccaggio scelta (tipicamente in mancanza di informazioni si scelgono le postazioni più vicine alla baia di carico stessa)

Determinata la posizione si acquisisce lo shuttle, si identifica il satellite migliore ed inizia così la missione.

Ovviamente, in un caso pratico, per effettuare tutti i controlli è necessario uno schema più complesso.

5.5.1. Go to satellite's level

Se lo shuttle non si trova allo stesso livello del satellite, si avvia questa sub-flow.

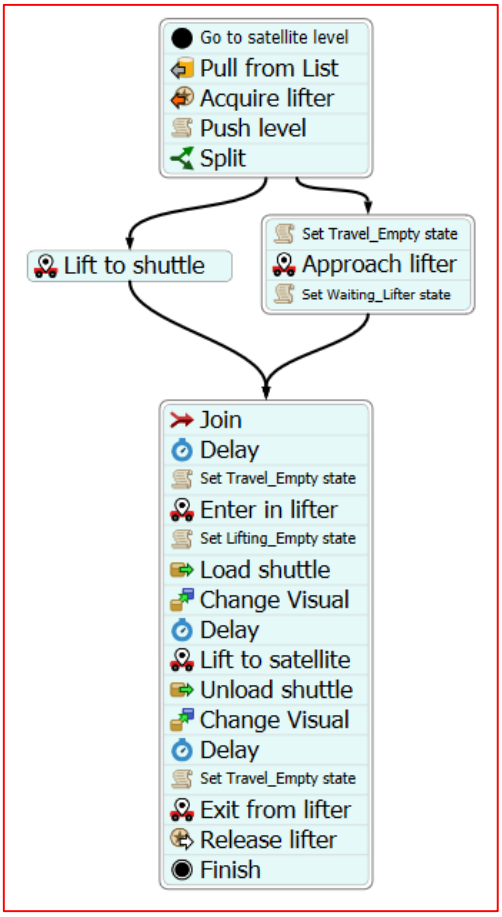


Figura 5.3 – Go to satellite's level

Lo shuttle innanzitutto acquisisce il livello in cui si trova il satellite (o resta in attesa finché non si libera) e acquisisce il lifter. Acquisito il lifter può liberare il livello in cui si trova, in quanto non è possibile che in questa transizione un altro shuttle si inserisca nel livello (il lifter è già occupato).

L'avvicinamento dello shuttle al lifter e lo spostamento di quest'ultimo avvengono in contemporanea per ottimizzare i tempi.

Avvengono poi in sequenza l'ingresso nell'ascensore, lo spostamento al livello del satellite e la sua uscita, con conseguente rilascio del lifter.

Tra le operazioni sono inseriti dei cambi di stato dello shuttle secondo un set di statistiche alternativo a quello standard (essendo lo shuttle un elemento derivato da uno tradizionale, sono necessarie statistiche aggiuntive non presenti di default nel software).

I cambi di visuale servono a centrare graficamente lo shuttle nel lifter, non hanno impatto sulla simulazione.

Sono inseriti dei delay che simulano i tempi di scambio dati tra i vari sensori e PLC del sistema.

5.5.2. Load satellite

In questa fase avviene solamente l'avvicinamento reciproco di shuttle e satellite e la fase di carico di quest'ultimo sullo shuttle per passare alla fase successiva.

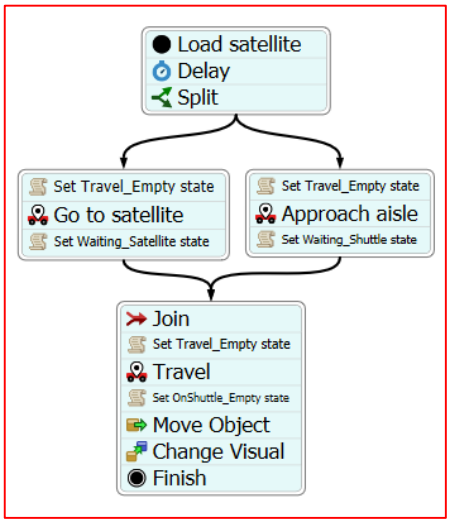


Figura 5.4 - Load satellite

Anche in questo ciclo vengono impostati manualmente gli stati che consentono di valutare le statistiche delle navette.

5.5.3. Load item

Una volta che il satellite è stato recuperato dallo shuttle viene acquisito il lifter.

Avviene ancora in contemporanea l'avvicinamento reciproco e il classico settaggio delle statistiche.

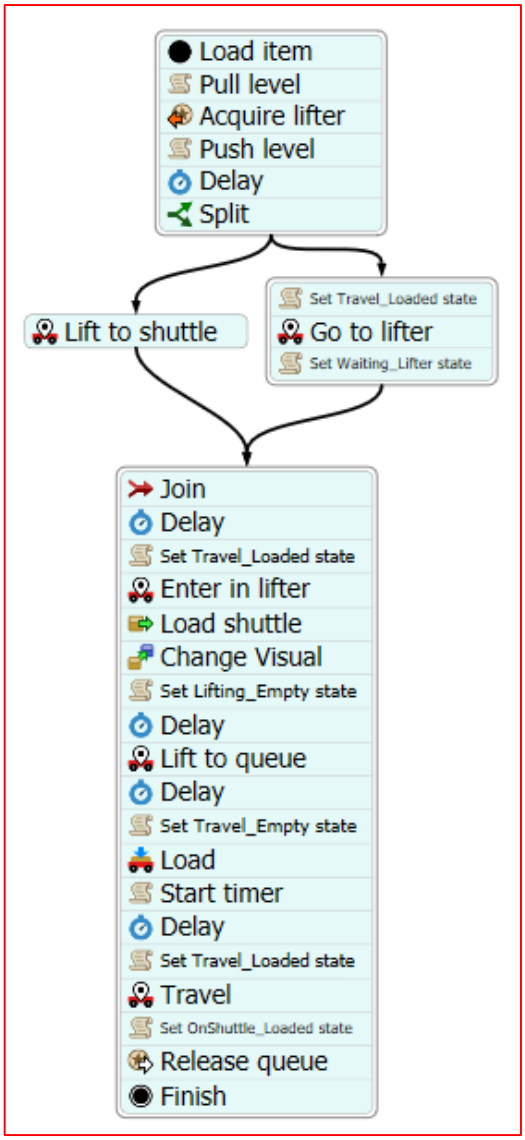


Figura 5.5 - Load item

Lo shuttle viene caricato sul lifter per scendere fino a raggiungere le baie di carico al livello zero del magazzino.

L'operazionedi carico avviene sfruttando il satellite.

Esistono versioni che consentono di caricare in autonomia il pallet sullo shuttle tramite rulli ma utilizzare il satellite risulta essere un sistema comune e rende più completo il modello.

Anche in questo ciclo sono presenti cambiamenti di stato e delay per simulare i tempi di risposta del sistema.

Al termine, la baia di carico viene rilasciata e resa usufruibile dalle missioni successive.

5.5.4. Unload item

Dopo aver completato l'operazione di carico e liberato la baia, il lifter porta le navette al livello di stoccaggio.

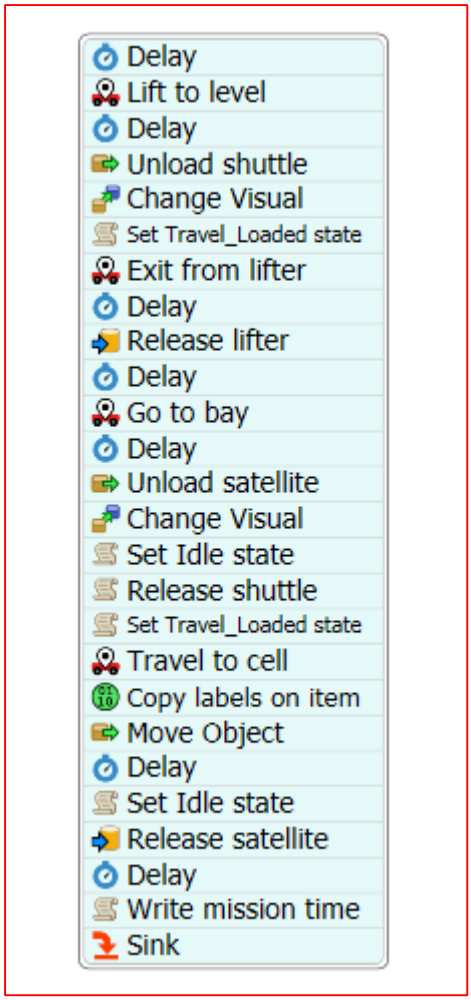


Figura 5.6 - Unload item

Lo shuttle trasla fino al canale selezionato e si arresta per permettere la discesa della navetta che trasporterà il pallet nel canale fino a raggiungere la posizione desiderata. Sia il lifter che lo shuttle, una volta completato il loro compito, vengono rilasciati dal token, ovvero vengono resi disponibili per l'utilizzo in altre missioni.

Al termine vengono salvate tutte le statistiche raccolte all'interno della global table "Warehouse" che contiene tutte le informazioni sul magazzino

5.5.5. Process Flow completi

Quanto riportato nelle sezioni precedenti funziona ma i risultati non sono ovviamente utilizzabili per valutare le prestazioni di un magazzino.

Sotto vengono riportati come esempi due modelli, rispettivamente per stoccaggio e prelievo, che sono in grado di operare in parallelo (con opportune limitazioni alle variabili del sistema) con risultati accettabili.

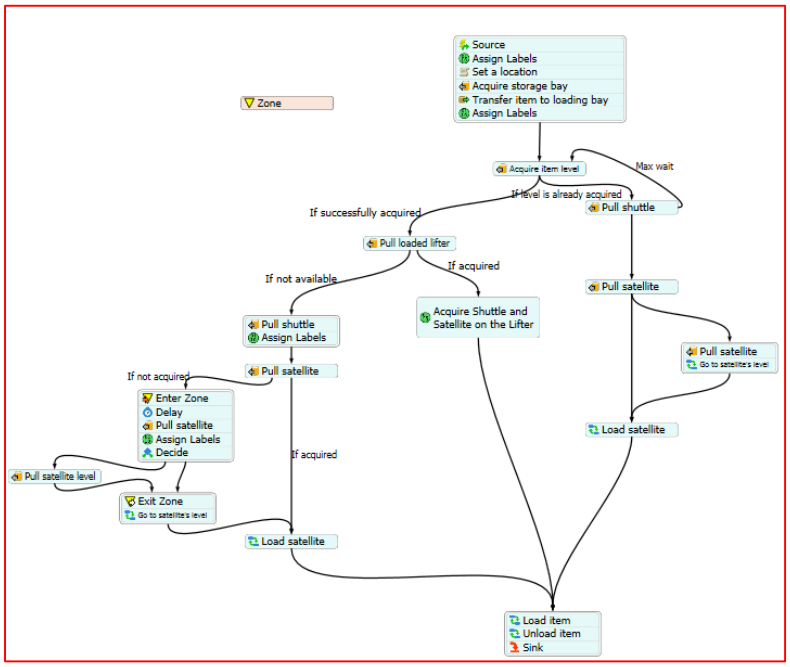


Figura 5.7 - ProcessFlow semplificato

Tutti i ProcessFlow possono essere composti da SubFlow, ovvero all'interno di un diagramma di flusso è possibile inserire rimandi ad altri diagrammi, in modo da semplificare la lettura.

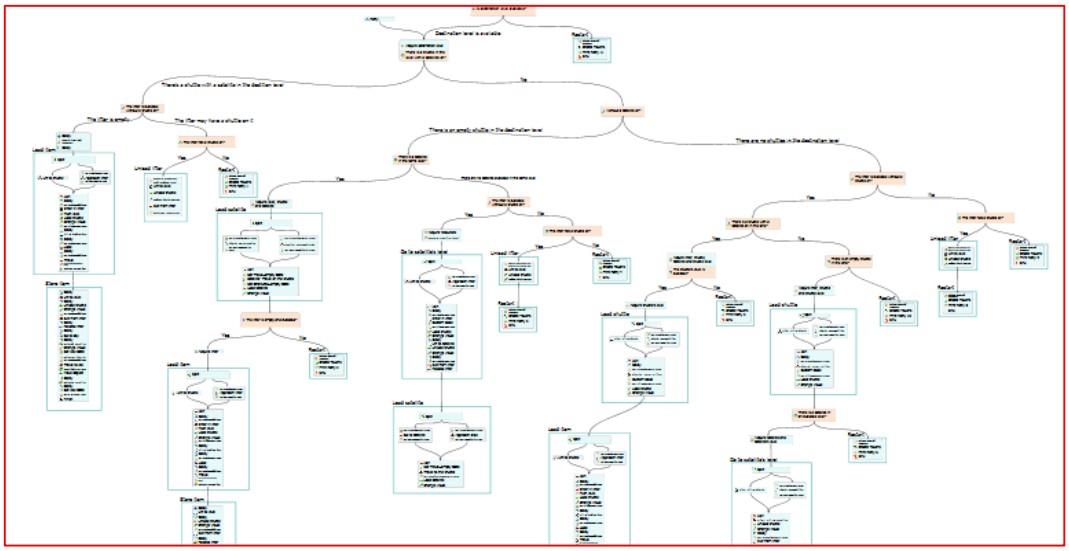


Figura 5.1 - Esempio di ProcessFlow integrale

6. Piano sperimentale

Mettere in piedi un piano di sperimentazione è necessario per capire quali fattori sono più significativi e qual è il loro impatto sui risultati della simulazione. Attraverso delle prove preliminari è stato possibile osservare i range oltre i quali l'incremento di una variabile non risulta più significativo e quali altri fattori invece non hanno un impatto osservabile sui dati raccolti.

Profondità dei canali: la soluzione ESMARTSHUTTLE svincola l'utilizzo di shuttle e satelliti. Come introdotto nelle logiche di funzionamento nel capitolo precedente, lo shuttle, dopo aver depositato un satellite, inizia subito la missione successiva. La profondità dei canali non influisce pertanto in modo significativo sulle prestazioni del magazzino, aumenta solo linearmente di pochi secondi la durata della singola missione. Pertanto, al di fuori di situazioni mirate, il piano di simulazioni prevedrà canali di dimensioni fisse (8 posti per canale).

Numero di livelli: variare il numero di livelli è essenziale per valutare il carico di lavoro del lifter. Il numero di posti pallet (e quindi di oggetti gestiti) aumenta in modo esponenziale il carico di lavoro gestito dalla CPU del computer utilizzato, pertanto ci si limiterà a magazzini di massimo 12 livelli (circa 22 metri di altezza).

Numero di corridoi: per lo stesso motivo di cui sopra, ci si limita a magazzini di 3-4 corridoi, in quanto è osservabile un aumento abbastanza lineare delle prestazioni al crescere del numero di corridoi.

Numero di canali per livello: analogamente al numero di corridoi, aumentare il numero di canali oltre certi limiti risulta poco significativo ed appesantisce inutilmente la simulazione.

Numero di shuttle: questo parametro non incide particolarmente sulla pesantezza delle simulazioni ma, limitandosi a situazioni verosimili, verranno testati fino a 4 shuttle per corridoio.

Numero di satelliti: tendenzialmente si installano due satelliti per ogni shuttle, quindi questo rapporto viene mantenuto fisso per tutte le simulazioni.

Numero di itemtype: questo fattore influisce in modo significativo solamente in configurazioni di piccole dimensioni, pertanto verrà fissato a 50 per ogni simulazione.

Il piano che ne deriva prevede quindi:

Variabile	Valori					
Corridoi	1	2	3			
Livelli	6	8	10	12		
Baie	5	10	15	20	25	30
Shuttle	1	2	3	4		

Per un totale di 288 casi base. Oltre a queste simulazioni ne sono state effettuate anche altre, con variabilità ridotta, variando parametri quali il numero di itemtype o la velocità del lifter. Questa simulazioni comprendono sia quelle effettuate come indagine preliminare sulla capacità di risposta del modello, sia per confronti su situazioni specifiche quali la variazione di velocità del lifter.

Per ogni simulazione vengono raccolte statistiche relative al tempo totale di simulazione, al throughput orario (ovvero il numero di missioni effettuate mediamente in un'ora), la percentuale di utilizzo medio dei lifter, degli shuttle e dei satelliti. Sulla base dei dati raccolti è possibile anche estrapolare l'energia consumata dal magazzino.

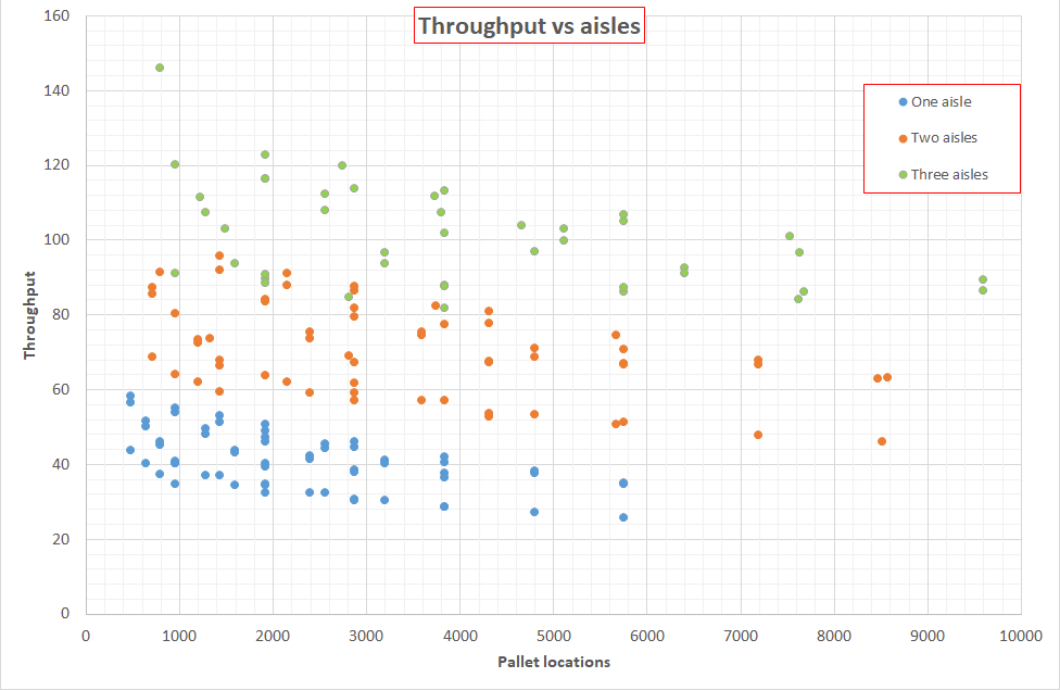
7. Analisi dei risultati

Seguiranno ora dei confronti sull'incidenza delle principali variabili sulle prestazioni del magazzino

7.1. Massimo throughput ottenibile

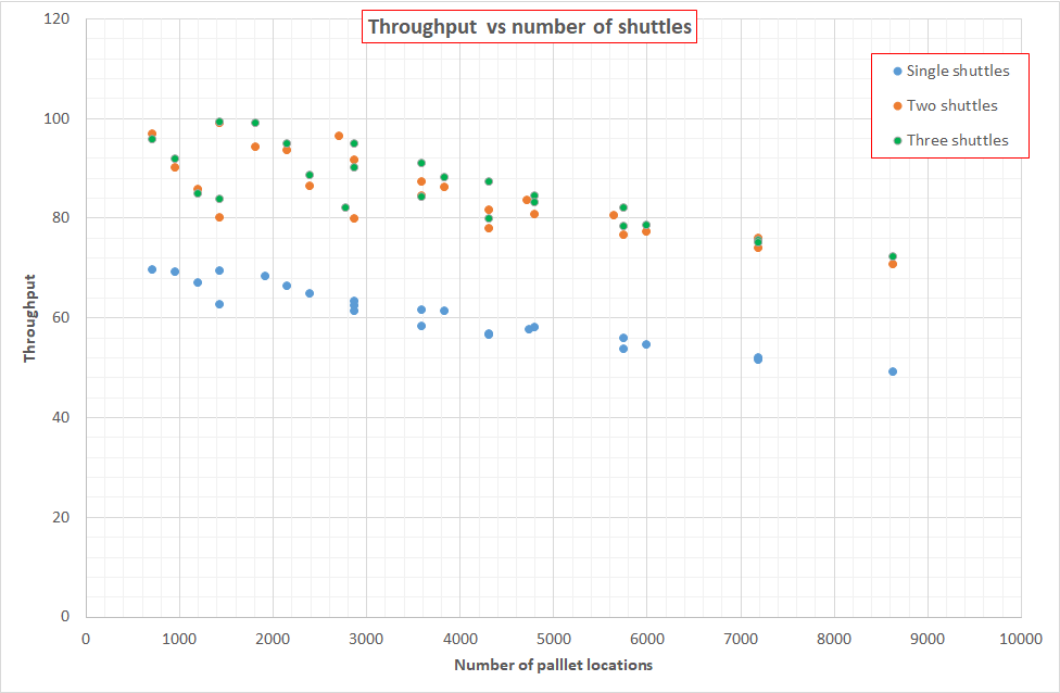
L'indice di movimentazione oraria è indubbiamente il fattore più interessante da valutare.

Come facilmente immaginabile, suddividendo i risultati ottenuti in base al numero di corridoi si ottiene una netta divisione tra le prestazioni. A parità di capacità di immagazzinamento, utilizzare due lifter raddoppia le prestazioni rispetto al singolo corridoio, in merito anche al fatto che, mantenendo costante il numero di posti pallet per ogni canale, si utilizza una struttura con meno livelli e con corridoi più corti.

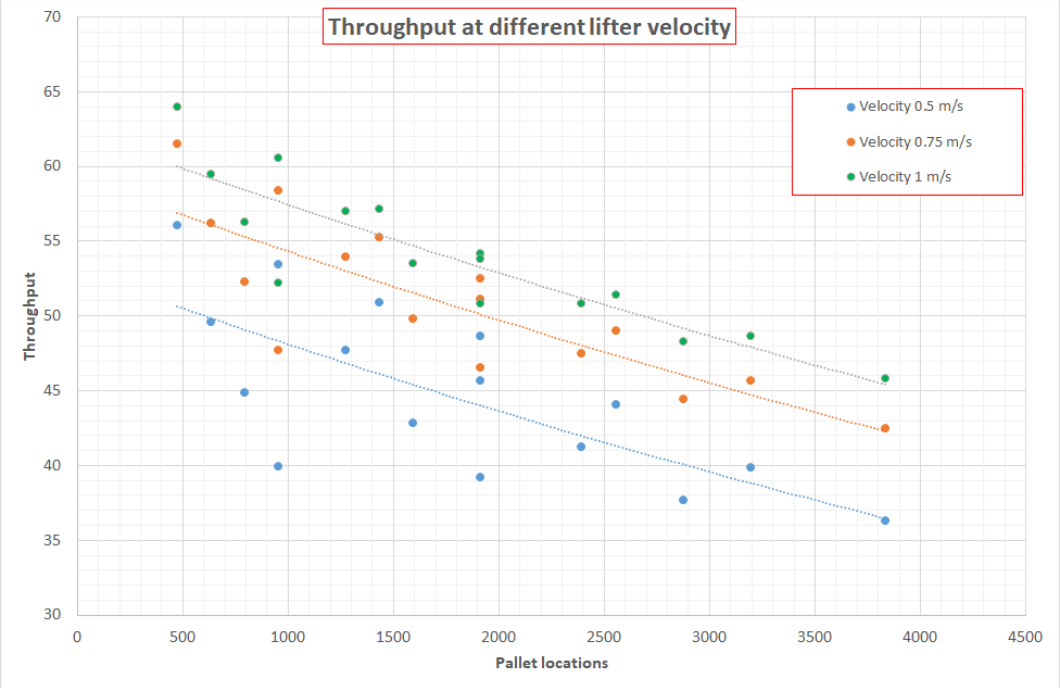


Valutando invece la dipendenza dal numero di shuttle utilizzati, limitando per facilità di lettura il numero di corridoi a due, si può facilmente osservare che l'utilizzo di due shuttle aumenta in modo considerevole le prestazioni del

magazzino a parità di capacità di immagazzinamento. L'utilizzo di più shuttle invece non porta a miglioramento, segno che il lifter con due shuttle si trova già in condizione di massima saturazione.



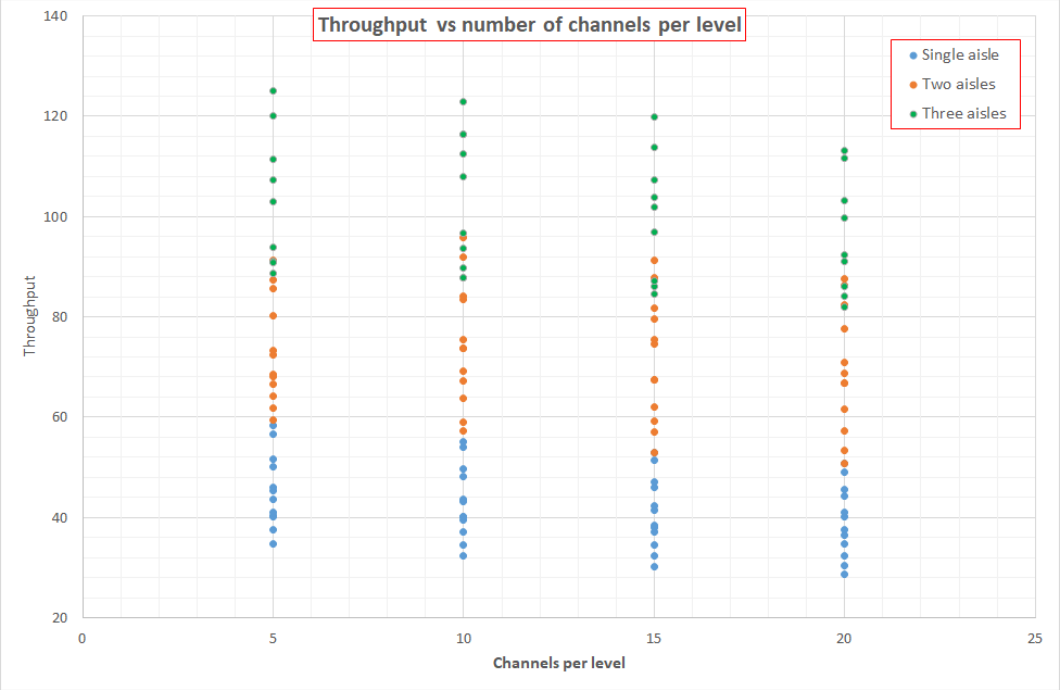
In questo caso si è utilizzato un lifter con una velocità massima di 0.75 m/s, con accelerazioni e decelerazioni massime di 0.5 m/s^2.



Il grafico precedente riporta, per un singolo corridoio, l'influenza della velocità del lifter sulla capacità di movimentazione. E' facile osservare che esso rappresenta il collo di bottiglia per questi sistemi. La spesa per l'aumento del numero di shuttle utilizzati in un magazzino automatico deve essere quindi supportata anche dall'acquisto di un dispositivo di sollevamento adeguato.

7.2. Influenza della lunghezza dei corridoi

Classificando i risultati in funzione del numero di canali per livello si ottiene la situazione sotto riportata.



Questo ci porta innanzitutto a confermare l'assunzione iniziale che, all'aumentare del numero di corridoi, l'indice di immagazzinamento orario aumenti in maniera pressoché lineare; infatti il passaggio da un singolo corridoio a due porta mediamente ad un raddoppio del throughput.

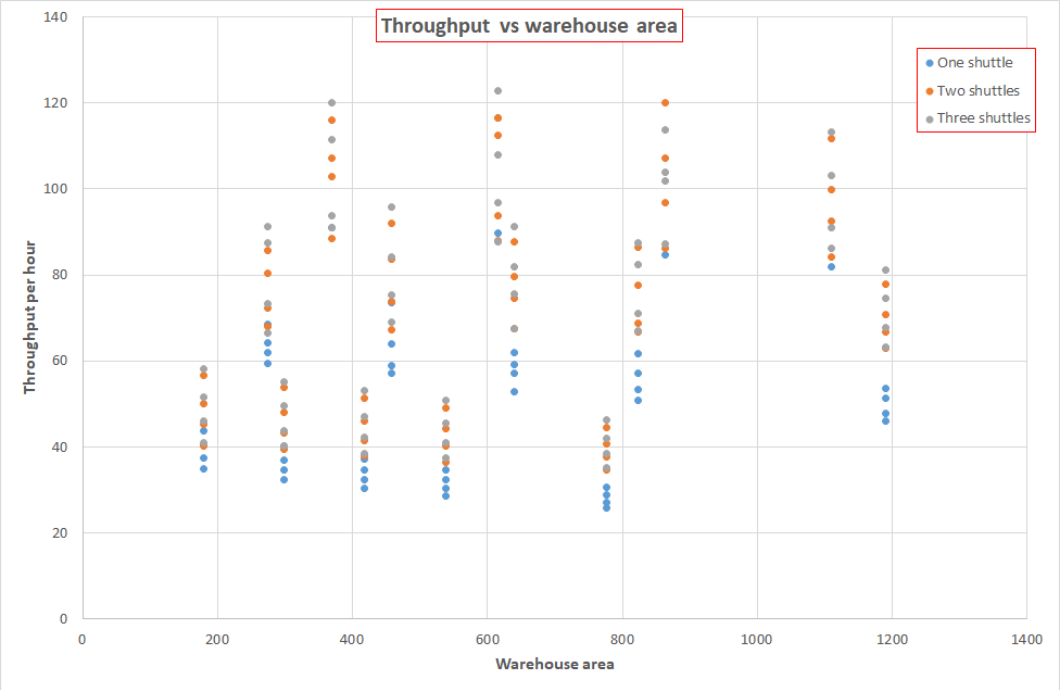
La seconda considerazione è invece sul valore medio a parità di numero di corridoi: al crescere del numero di canali non si nota un grosso peggioramento delle prestazioni, indice che la velocità di spostamento degli shuttle lungo i

corridoi non rappresenta un collo di bottiglia per il sistema, ma piuttosto che il tempo medio delle missioni dipende principalmente dai tempi carico/scarico tra navette, baie di carico e lifter.

7.3. Classificazione per dimensioni

Un'altra tipologia di classificazione utile è rispetto alla superficie disponibile.

Conoscendo quali possono essere le dimensioni massime della struttura è possibile in questo modo conoscere qualitativamente quali potranno essere le capacità di stoccaggio orario ottenibili.



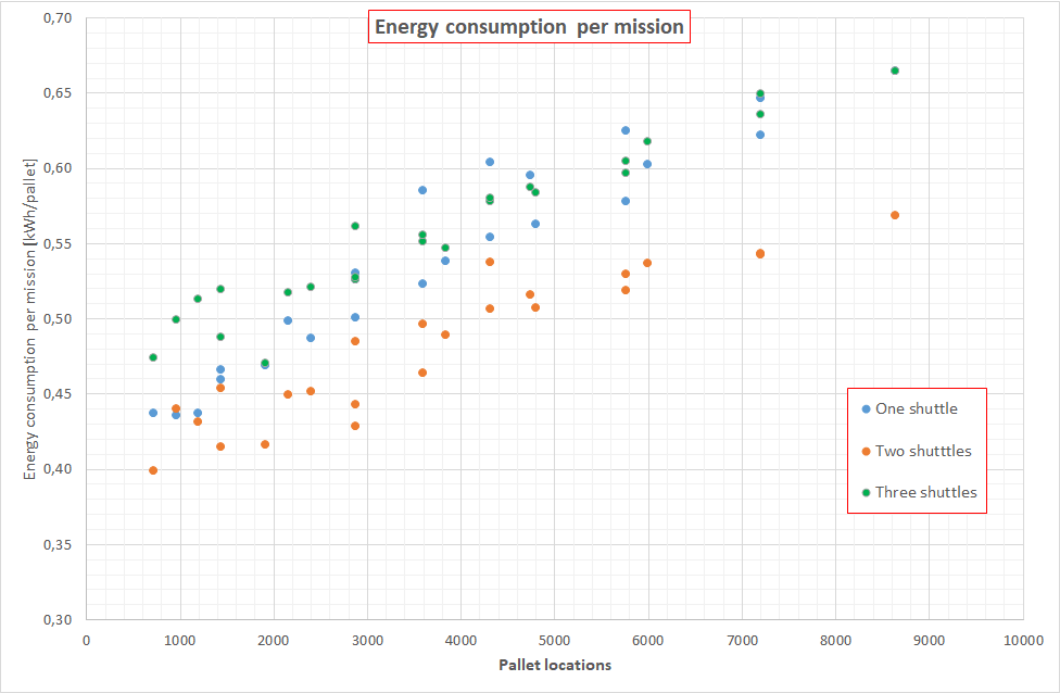
7.4. Consumo energetico

Risulta interessante osservare anche il comportamento energetico del magazzino. Raccogliendo informazioni sull'indice di utilizzo medio delle navette e del lifter è possibile stimare infatti il consumo energetico richiesto per effettuare una missione nelle differenti configurazioni.

Per queste analisi si assumono delle potenze assorbile pari a:

- 1.5 kW per gli shuttle
- 0.67 kW per i satelliti
- 17 kW per i lifter

Riprendendo l'esempio precedente, si può distinguere nettamente il risparmio energetico ottenuto utilizzando due shuttle rispetto ad un solo elemento.



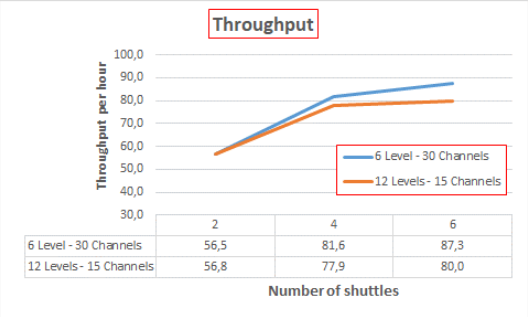
Questo fenomeno è giustificato dall'aver mantenuto il rapporto di due satelliti per ogni shuttle; in questo modo si ritrovano satelliti disponibili su più livelli, riducendo le corse necessarie al raggiungimento di una navetta disponibile.

7.5. Ricerca di una configurazione ottimale

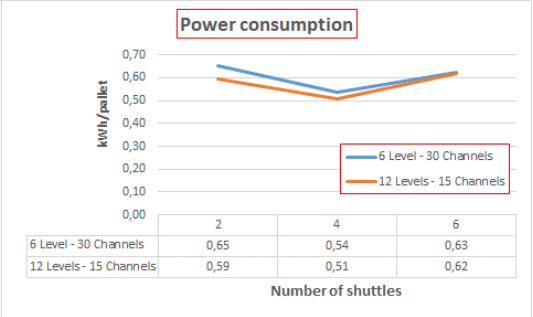
Oltre che ad osservare comportamenti su ampia scala, lo scopo di questa tesi è soprattutto fornire uno strumento di risposta rapida per casi specifici. Per fare questo è possibile limitare il range dei dati osservati per rispecchiare esigenze specifiche.

7.5.1. Esempio 1 - Capienza minima del magazzino

Un primo caso può essere rappresentato dal dover realizzare un magazzino da almeno 4000 posti pallet, con una elevata differenziazione tra i prodotti da gestire, senza indicazioni sulle dimensioni massime da considerare. Per rispondere a questa richiesta si ricercano tra le simulazioni effettuate le casistiche più vicine alle esigenze e si confrontano le performance ottenute



	2	4	6
6 Level - 30 Channels	56,5	81,6	87,3
12 Levels - 15 Channels	56,8	77,9	80,0



	2	4	6
6 Level - 30 Channels	0,65	0,54	0,63
12 Levels - 15 Channels	0,59	0,51	0,62

In questo caso d'esempio si confrontano due configurazioni da circa 4300 posti pallet divisi in 540 canali da 8 posti l'uno. Le dimensioni sono leggermente superiori al richiesto per consentire di non mescolare i prodotti nei canali qualora il magazzino si avvicini alla saturazione.

Valutando due configurazioni opposte tra loro (2 corridoi, 6 livelli da 30 canali o 12 livelli da 15 canali) si racchiudono verosimilmente le possibili configurazioni intermedie.

Si può osservare che la velocità di stoccaggio è superiore utilizzando meno livelli ma, per contro, il consumo energetico risulterà leggermente più elevato, a parità di numero di navette impiegate. In entrambi i casi le dimensioni

suggeriscono che, salvo necessità di prestazioni molto elevate, l'utilizzo di due shuttle per corridoio è la scelta migliore in termini di consumo energetico.

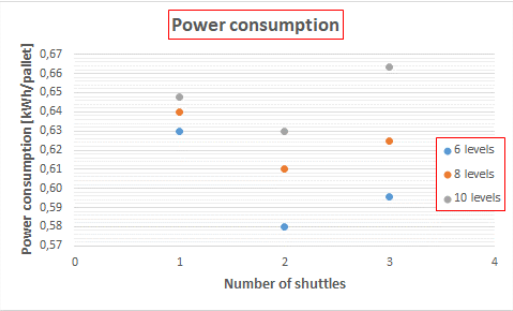
Questo confronto permette di scegliere rapidamente le dimensioni del magazzino e sapere fin da subito che una missione di stoccaggio/prelievo durerà mediamente 45 secondi, con un consumo medio di 0.55 kWh che, con un costo medio dell'energia elettrica di 0.15€/kWh significa meno di 0.10€ per ogni trasporto (8€ l'ora per l'intero magazzino).

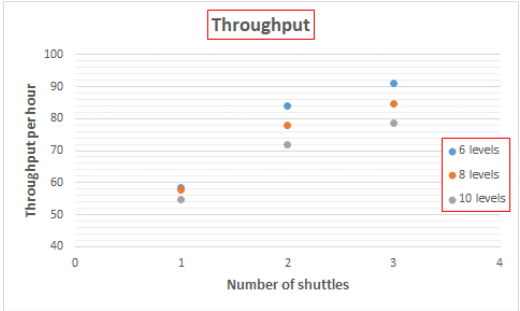
Anche prendendo in considerazione i costi di supervisione e manutenzione è facile osservare che la spesa annua per l'utilizzo di un sistema AS/RS è molto contenuta in rapporto alla sua efficienza, pertanto l'investimento iniziale può essere facilmente recuperato se ben dimensionato per le reali esigenze del cliente.

7.5.2. Esempio 2 – Riqualificazione magazzino esistente

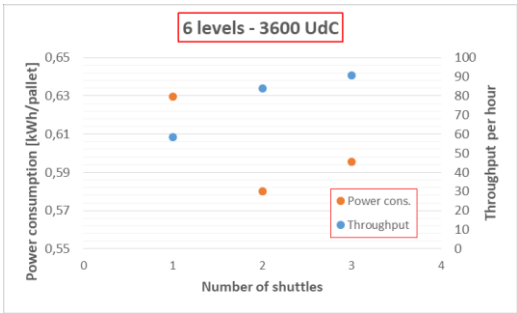
Un secondo caso può essere quello di un cliente che disponga già di un'area destinata a magazzino ma interessato ad aumentarne l'efficienza.

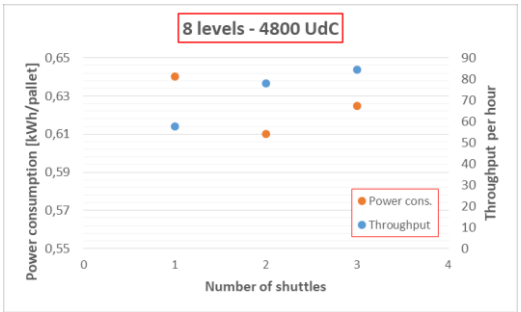
Prendiamo come esempio una struttura industriale in cui esisteva già un magazzino a media altezza, circa 40x30 metri, con altezza sotto filo catena di 15 metri.

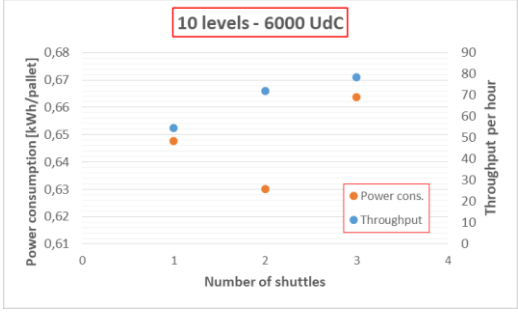




Si può ricavare subito che le prestazioni medie potranno variare tra le 50-100 movimentazioni orarie, con un consumo variabile tra 0.58 e 0.67 kWh/missione.







8. Conclusioni e sviluppi futuri

Il problema principale per qualsiasi azienda fornitrice di magazzini ad elevata automazione consiste nel riuscire a fornire un preventivo affidabile alle richieste di molti clienti.

Prima di passare all'ottimizzazione di una soluzione infatti è necessario fornire una risposta al cliente in tempi brevi; ad una richiesta generica di informazioni, in cui vengono forniti indicativamente il numero di pallet da immagazzinare e la movimentazione media giornaliera, una società quale EuroFork deve poter presentare una serie di soluzioni con indicazione medie di investimento iniziale, dimensione e spese di gestione.

Come anticipato, questa attività può essere svolta preventivamente attraverso la costruzione di un piano di simulazioni che copra le principali configurazioni realizzabili.

Blocchi	Livelli	Canali per livello	Posti per canale	Shuttle per corridoio	Satelliti ogni corridoio/	Inter-arrival time	Item stored	Number of itemtype	Totale canali	Totale posti	Totale shuttle	Totale satelliti	Totale lifter	Simulation time	Average mission time	Throughput per hour	% Idle	% Waiting	% Loaded	% Empty
2	6	10	10	2	2	30	1000	20	120	1200	2	2	1	63383	30179	56,8	46%	34%	12%	8%
2	6	10	10	2	2	30	1000	50	120	1200	2	2	1	62052	29563	58,02	45%	34%	12%	8%
2	6	10	10	2	2	60	1000	20	120	1200	2	2	1	64372	31181	55,92	47%	33%	12%	8%
2	6	20	10	2	2	30	2000	20	240	2400	2	2	1	141741	64942	50,8	44%	33%	13%	10%
2	6	20	10	2	2	30	2000	50	240	2400	2	2	1	139441	64497	51,63	43%	33%	14%	10%
2	6	20	10	2	2	60	2000	20	240	2400	2	2	1	144291	67299	49,9	45%	32%	13%	10%
2	6	20	10	2	2	60	2000	50	240	2400	2	2	1	141514	66322	50,88	44%	33%	13%	10%
2	6	20	15	2	2	30	2000	20	240	3600	2	2	1	140041	65291	51,41	43%	33%	14%	10%
2	6	20	15	2	2	30	2000	50	240	3600	2	2	1	139696	65071	51,54	43%	33%	14%	10%
2	6	20	15	2	2	60	2000	20	240	3600	2	2	1	141240	66284	50,98	44%	33%	13%	9%
2	6	20	20	2	2	30	2000	50	240	4800	2	2	1	137393	66755	52,4	46%	34%	11%	8%
2	6	30	10	2	2	30	2000	50	360	3600	2	2	1	140301	65103	51,32	44%	33%	14%	10%
2	8	10	15	2	2	30	2000	50	160	2400	2	2	1	132936	64314	54,16	45%	35%	11%	8%
2	8	10	15	2	2	60	2000	50	160	2400	2	2	1	133375	64463	53,98	45%	35%	11%	8%
2	8	20	20	2	2	30	2000	50	320	6400	2	2	1	136617	67026	52,7	47%	35%	11%	8%
2	8	30	15	2	2	30	5000	50	480	7200	2	2	1	389096	179305	46,26	43%	33%	14%	10%
2	10	10	15	2	2	30	2000	50	200	3000	2	2	1	132564	64105	54,31	45%	36%	11%	8%
2	10	20	15	2	2	30	2000	50	400	6000	2	2	1	133488	64413	53,94	46%	35%	11%	8%
2	10	20	15	2	2	30	5000	50	400	6000	2	2	1	379915	177406	47,38	43%	34%	13%	10%
2	10	20	15	2	2	30	5000	20	400	6000	2	2	1	378029	174293	47,62	43%	34%	13%	10%
2	8	30	15	2	2	30	2000	20	480	7200	2	2	1	134313	64701	53,61	46%	35%	11%	8%

Per le aziende con l'intenzione di riconvertire magazzini esistenti in strutture automatizzate con un'efficienza superiore è importante valutare se l'investimento iniziale è recuperabile in termini di risparmio nel tempo. La possibilità di inserire il sistema ESMARTSHUTTLE all'interno di strutture già esistenti rende questo prodotto una soluzione da valutare accuratamente.

Al termine di questo lavoro, il primo punto di riflessione è certamente sull'utilità di servirsi di software ad eventi discreti per la simulazione di magazzini ad elevata automazione.

Attualmente non esistono alternative per predire con affidabilità le prestazioni ottenibili con questi sistemi, solitamente ci si affida ad esperienza e soluzioni realizzate in precedenza. Con lo sviluppo di un modello di simulazione ben calibrato l'unico limite all'accuratezza del risultato è la variabilità dei dati di input.

Tramite il confronto con soluzioni esistenti, le prestazioni ed i tempi di risposta delle navette sono estremamente accurati; è possibile introdurre latenze dovute ad inerzie, tempi di risposta dei PLC e addirittura soste per problemi tecnici.

L'aspetto probabilmente più utile è la possibilità di valutare l'efficacia delle logiche di controllo, senza intervenire direttamente programmando e testando un magazzino di testing appositamente realizzato, con un importante risparmio di tempo e risorse.

Un aspetto negativo è invece legato all'utilizzo di un software di simulazione generico quale Flexsim senza l'ausilio di un team di sviluppo specializzato: sebbene l'utilizzo di questi software sia abbastanza orientato all'utilizzo da parte di un utente poco esperto, questi sistemi soffrono comunque delle mancanza di elementi e funzioni non realizzate appositamente per magazzini serviti da navette. Appurati i vantaggi introdotti dalla possibilità di simulare una soluzione, risulta conveniente dotarsi di software più evoluti e personalizzati per le proprie esigenze.

Bisogna tener conto in ogni caso che la simulazione ad eventi discreti ha già portato ad enormi vantaggi nella progettazione di siti produttivi, essendo in grado di gestire elementi molto più complessi da descrivere matematicamente, primi tra tutti gli operatori.

Per quanto concerne il modello sviluppato, esso è compatibile con le nuove versioni rilasciate di Flexsim, nonostante l'introduzione di interessanti nuove

funzionalità (specialmente nel codice Flexscript, derivazione del C++ utilizzato nel software).

I principali miglioramenti introducibili riguardano l'utilizzo del tool Experimenter, che consente di programmare decine di scenari ed effettuarne la simulazione al di fuori dell'ambiente grafico, riducendo drasticamente i tempi di elaborazione.

Per il futuro, spero che qualcuno possa sfruttare questa tesi come punto di partenza per creare un modello di larga scala, più intuitivo e stabile di quello attuale, sfruttando le novità introdotte con le nuove release di Flexsim attualmente in fase di rilascio.

BIBLIOGRAFIA

[1]. Armando Monte (2009): "Elementi di impianti industriali. Vol 1". Cortina ed

[2]. Howard Zollinger, Zollinger Associates Inc (2001): "AS/RS application, benefits and justification in comparison to other storage methods: a white paper". Disponibile su http://www.mhi.org

[3]. Antonio Forte (2017): "La congiuntura internazionale", Centro Europa Ricerche. Disponibile su https://www.centroeuroparicerche.it

[4]. Banu Yetkin Ekren (2011): "Performance evaluation of AVS/RS under various design scenarios: a case study". The International Journal of Advanced Manufacturing Technology, August 2011, Volume 55, Issue 9–12, pp 1253–1261

[5]. Eurofork, 2018: Catalogo Esmartshuttle®, disponibile su http://www.eurofork.com/sites/default/files/download/eurofork-esmartshuttle-catalogue-2018.pdf

[6]. Mecalux, Pallet Shuttle. Brochure disponibile su https://www.mecalux.it/scaffalature-metalliche/scaffalature-per-pallet/pallet-shuttle

[7]. Jungheinrich, Under Pallet Carrier UPC. Brochure disponibile su https://www.jungheinrich.it/prodotti/mezzi-di-movimentazione/sistemi-shuttle-pallet-carrier/under-pallet-carrier-upc/

[8]. SSI Schaefer, CUBY: single-level shuttle solution for maximum storage density. Brochure disponibile su https://www.ssi-schaefer.com/en-us/products/storage/small-load-carriers/storage-shuttle-systems-bins-cartons-trays/cuby--53500

APPENDICE I. Create_Warehouse

Di seguito è riportato l'intero codice che consente di generare la struttura del magazzino e predispone gruppi e tabelle necessarie alla simulazione.

```
/** ****************** CREATE WAREHOUSE ***************************** **/

// La creazione del magazzino avviene per blocchi di scaffali. Esiste un elemento Rack di base che
// viene configurato con il numero di livelli e baie per livello desiderato, ne vengono create poi
// delle copie parallele, in modo da realizzare la profondità dei canali.

/** Clear model **/

// Elimina treenode fino a trovare gli elementi di base da mantenere
// identificati da Rack, posto con il rank più basso

while (getname(last(model)) != "Rack")
{
	destroyobject(last(model));
}

/** Set Variables **/

// Offset iniziali
double x_zero = getnodenum(node("/Data/x_zero", model));
double y_zero = getnodenum(node("/Data/y_zero", model));
double z_zero = getnodenum(node("/Data/z_zero", model));

// Genero una variabile che userò in seguito per posizionare i blocchi
```

```
double ypointer = y_zero;


// Dimensioni della scaffaltura
double baywidth  = getnodenum(node("/Data/Geometry/Location/Length", model));
double rackwidth = getnodenum(node("/Data/Geometry/Location/Width",  model));
double lvlheight = getnodenum(node("/Data/Geometry/Location/Height", model));
double aislesize = getnodenum(node("/Data/Geometry/aislesize", model));


int numofblocks = getnodenum(node("/Data/Blocks", model));
int numofaisles = numofblocks - 1;


int shuttles   = getnodenum(node("/Data/ESMART/numofshuttle",   model));
int satellites = getnodenum(node("/Data/ESMART/numofsatellite", model));


int total_channel = 0;
int total_cell    = 0;


treenode queue = node("fixedresources/Queue", library);
treenode lifter = node("../userlibrary/ESMARTSHUTTLE/Lifter", library);
treenode shuttle = node("../userlibrary/ESMARTSHUTTLE/Shuttle", model);
treenode satellite = node("../userlibrary/ESMARTSHUTTLE/Satellite", model);



/** Build **/
// Ciclo di creazione dei blocchi di scaffali


for(int block = 1; block <= numofblocks; block++)
{
        int bays  = getnodenum(node("/Data/Blocks/Bays", model));
        int levels          = getnodenum(node("/Data/Blocks/Levels", model));
```

```
total_channel+=bays*levels;
setnodenum(node("/Data/Blocks/Channels", model), total_channel);

int racks = getnodenum(node("/Data/Blocks/Location", model));

total_cell+=bays*levels*racks;
setnodenum(node("/Data/Blocks/Cells", model), total_cell);

// Creo una copia dell'elemento base
createcopy(node("Rack", model), model);
setnodename(last(model), "Rack0");
treenode RACK = last(model);

// IMPOSTO LE DIMENSIONI
setnodenum(node(">variables/sizetable/Bay1", RACK), baywidth);
setnodenum(node(">variables/sizetable/Level1", RACK), lvlheight);
setnodenum(node(">spatial/spatialsx", RACK), bays*baywidth);
setnodenum(node(">spatial/spatialsy", RACK), rackwidth);
setnodenum(node(">spatial/spatialsz", RACK), levels*lvlheight);

// CREO I LIVELLI ALL'INTERNO DEL NODO BAY
// Devo creare il livello in sizetable, creare il nodo in cui ne verrà salvato
// il contenuto e impostarne la locazione
for(int j=1; j<levels; j++)
{
        createcopy(node("/Rack0>variables/sizetable/Bay1/Level1", model),
                        node("/Rack0>variables/sizetable/Bay1", model));

        setnodename(last(node("/Rack0>variables/sizetable/Bay1", model)),
                        concat("Level", numtostring(j)));
```

```
            createcopy(last(last(node("Rack0>variables/contenttable", model))),
                        last(node("Rack0>variables/contenttable", model)),1);

            createcopy(last(last(node("Rack0>variables/locationtable", model))),
                        last(node("Rack0>variables/locationtable", model)),1);

            setnodenum(last(last(node("Rack0>variables/locationtable", model))),
                        j*baywidth);
        }

        // DUPLICO LE BAY
        // Anche in questo caso devo farlo per sizetable, location e content
        for(int k=1; k<bays; k++)
        {
            createcopy(node("/Rack0>variables/sizetable/Bay1", model),
                        node("/Rack0>variables/sizetable", model));

            setnodename(last(node("/Rack0>variables/sizetable", model)),
                        concat("Bay", numtostring(k)));

            createcopy(last(node("Rack0>variables/contenttable", model)),
                        node("Rack0>variables/contenttable", model),1);

            createcopy(last(node("Rack0>variables/locationtable", model)),
                        node("Rack0>variables/locationtable", model),1);

            setnodenum(last(node("Rack0>variables/locationtable", model)),
                        k*lvlheight);
```

```
}

// CREO LE COPIE DI RACK0 PER COSTRIRE IL BLOCCO
for(int rack=1; rack<=racks; rack++)
{
        createcopy(node("Rack0", model), model);
        setnodename(last(model), concat("Rack", numtostring(block), "_",
                numtostring(rack)));

        // Il setloc deve ricordarsi anche del posizionamento dei blocchi
        // precedenti, sfruttando la variabile "ypointer"
        setloc(last(model), x_zero, ypointer+rack*rackwidth, z_zero);

        // Questi comandi sono necessari per il refresh "grafico" della sizetable
        function_s(node("/?Rack", library()), "BasicRefreshTable", last(model),
                        levels, bays, lvlheight, baywidth);

        nodefunction(node("/?Advanced>update", library));
}

ypointer = yloc(last(model)) + aislesize/2;

// POSIZIONO LE QUEUE
// Esse sono le baie di carico. Per ogni corridoio intermedio sono previste due baie,
// Viene impostata forma e posizione, il nome e una label che identifica il corridoio di
// riferimento. Viene impostato il massimo contenuto a 1.
if(block<numofblocks)
{
        createcopy(queue, model);
        setloc(last(model), x_zero-2.5, ypointer-1, z_zero);
        setsize(last(model), 1, 1, 0.1);
```

```
            groupaddmember("Storage_Bays", last(model));
            setlabel(last(model), "aisle", block);
            setnodenum(node(">variables/maxcontent", last(model)), 1);
            setnodename(last(model), concat("Storage Bay ", numtostring(block)));

            createcopy(queue, model);
            setloc(last(model), x_zero-2.5, ypointer+2, z_zero);
            setsize(last(model), 1, 1, 0.1);
            groupaddmember("Retrieval_Bays", last(model));
            setlabel(last(model), "aisle", block);
            setnodenum(node(">variables/maxcontent", last(model)), 1);
            setnodename(last(model), concat("Retrieval Bay ", numtostring(block)));
}


// POSIZIONO IL LIFTER
// Per ogni corridoio è previsto un lifter
if(block<numofblocks)
{
            dropuserlibraryobject(lifter);
            setloc(last(model), -3, ypointer+ysize(lifter)/2, 0);
            setnodename(last(model), concat("Lifter", numtostring(block)));
            setlabel(last(model), "aisle", block);
            groupaddmember("Lifters", last(model));
            setresetposition(last(model));
}

// POSIZIONO GLI SHUTTLE
// Inserisco il numero di shuttle voluti nel corridoio e gli assegno posizione e nome

if(block<numofblocks)
```

```
{
        for(int j=1; j<=shuttles; j++)
        {
                createcopy(shuttle, model);

                string name = concat("Shuttle", numtostring(block),
                        numtostring(j));

                setnodename(last(model), name);

                setloc(last(model), 2, ypointer+ysize(shuttle)/2,
                        z_zero+(j-1)*lvlheight-zsize(shuttle));

                setresetposition(last(model));
                setlabel(last(model), "aisle", block);
                setlabel(last(model), "actual_level", j);
                groupaddmember("Shuttles", last(model));
        }
}

// POSIZIONO I SATELLITI
// Analogamente posiziono il numero di satelliti voluti nei blocchi
// Il nome identifica Corridoio_Blocco_Numero
double x = x_zero+rackwidth/2-xsize(satellite)/2;

if(block<numofblocks)
{
        for(int j=1; j<=satellites; j++)
        {
                createcopy(satellite, model);
```

```
                setnodename(last(model), concat("Sat", numtostring(block),
                        "_", numtostring(block), "_", numtostring(j)));

                setloc(last(model), x, ypointer-aislesize+ysize(satellite)/2,
                z_zero+(j-1)*lvlheight-zsize(satellite));

                setresetposition(last(model));
                groupaddmember("Satellites", last(model));
                setlabel(last(model), "block", block);
                setlabel(last(model), "aisle", block);
                setlabel(last(model), "actual_level", j);
        }

        for(int j=1; j<=satellites; j++)
        {
                createcopy(satellite, model);

                setnodename(last(model), concat("Sat", numtostring(block),
                        "_", numtostring(block+1), "_", numtostring(j)));

                setloc(last(model), x, ypointer+aislesize+ysize(satellite)/2,
                z_zero+(j-1)*lvlheight-zsize(satellite));

                setresetposition(last(model));
                groupaddmember("Satellites", last(model));
                setlabel(last(model), "block", block+1);
                setlabel(last(model), "aisle", block);
                setlabel(last(model), "actual_level", j);
        }
}
```

```
            ypointer += aislesize/2;
            destroyobject(RACK);
}

/** Initialize structure table **/

settablesize("Warehouse", total_cell, 20);

for(int index=1; index<=total_cell; index++)
{
            for(int index2=7; index2<=20; index2++)
            {
                        settablenum("Warehouse", index, index2, 0);
            }
}



settablesize("Warehouse", total_cell, 20);

// Imposto gli header definito con GlobalMacro
settableheader("Warehouse", 2, 1,  HEAD_1);
settableheader("Warehouse", 2, 2,  HEAD_2);
settableheader("Warehouse", 2, 3,  HEAD_3);
settableheader("Warehouse", 2, 4,  HEAD_4);
settableheader("Warehouse", 2, 5,  HEAD_5);
settableheader("Warehouse", 2, 6,  HEAD_6);
settableheader("Warehouse", 2, 7,  HEAD_7);
settableheader("Warehouse", 2, 8,  HEAD_8);
settableheader("Warehouse", 2, 9,  HEAD_9);
settableheader("Warehouse", 2, 10, HEAD_10);
settableheader("Warehouse", 2, 11, HEAD_11);
```

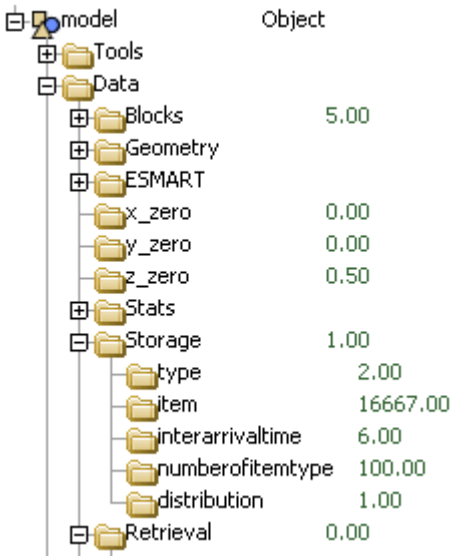
```
settableheader("Warehouse", 2, 12, HEAD_12);
settableheader("Warehouse", 2, 13, HEAD_13);
settableheader("Warehouse", 2, 14, HEAD_14);
settableheader("Warehouse", 2, 15, HEAD_15);
settableheader("Warehouse", 2, 16, HEAD_16);



int counter = 1;
for(int block = 1; block <= numofblocks; block++)
{
        int bays  = getnodenum(node("/Data/Blocks/Bays", model));
        int levels          = getnodenum(node("/Data/Blocks/Levels", model));
        int locs   = getnodenum(node("/Data/Blocks/Location", model));

        for(int level = 1; level<=levels; level++)
        {
                for(int bay = 1; bay<=bays; bay++)
                {
                        for( int loc = 1; loc<=locs; loc++)
                        {
                                string ID;

                                if(block<10)
                                ID = concat("0", numtostring(block));
                                else
                                ID = numtostring(block);

                                if(level<10)
                                ID = concat(ID, "0", numtostring(level));
                                else
```

```
                                        ID = concat(ID, numtostring(level));

                                        if(bay<10)
                                        ID = concat(ID, "0", numtostring(bay));
                                        else
                                        ID = concat(ID, numtostring(bay));


                                        settablenum("Warehouse", counter, 1, block);
                                        settablenum("Warehouse", counter, 2, level);
                                        settablenum("Warehouse", counter, 3, bay);
                                        settablestr("Warehouse", counter, 4, ID);
                                        settablenum("Warehouse", counter, 5, 0.2*level+bay-1);
                                        settablenum("Warehouse", counter, 6, loc);

                                        counter++;
                                }
                        }
                }
        }
```

APPENDICE II. Update_Storage_Source

Il codice seguente è utilizzato per aggiornare la sorgente delle missioni di stoccaggio.

E' basato sui dati contenuti nel treenode Data/Storage, da cui viene selezionato il numero di missioni, il numero di itemtype, l'intervallo di arrivo dei prodotti e la distribuzione statistica di quest'ultimi.




```
/**Update_Storage_Source*/

treenode source = node("Storage Source", model);

treenode storage = node("/Data/Storage", model);


// Identfico innanzitutto la tipologia di funzionamento

int mode = getnodenum(node("/Data/Storage/type", model));

setnodenum(node(">variables/arrivalmode", source), mode);


int numberofitemtype = getnodenum(node("/numberofitemtype", storage));
```

```
int distribution =    getnodenum(node("/distribution", storage));


// Configuro Storage Source


if(mode==2)
{
        treenode schedule = node(">variables/schedule", source);
        int settime = getnodenum(node("/interarrivaltime", storage));


        // Elimino la vecchia schedulazione
        for(int x=content(schedule); content(schedule)>0; x--)
        {
                destroyobject(last(schedule));
        }


        // Creo ora il numero di missioni desiderato
        int n = getnodenum(node("/item", storage));
        int t = 0; // Temporizzatore
        for(int y=1; y<=n; y++)
        {
                t+= settime;
                nodeinsertinto(schedule);
                setnodename(last(schedule), concat("Arrival", numtostring(y)));


                nodeinsertinto(last(schedule)); // Tempo di arrivo
                nodeadddata(last(last(schedule)), DATATYPE_NUMBER);
                setnodenum(last(last(schedule)), t);


                nodeinsertinto(last(schedule)); // Nome item
                nodeadddata(last(last(schedule)), DATATYPE_STRING);
                setnodestr(last(last(schedule)), "Product");
```

```
                    nodeinsertinto(last(schedule)); // Itemtype
                    nodeadddata(last(last(schedule)), DATATYPE_NUMBER);
                    if(distribution==1) // D Uniform
                    {
                        setnodenum(last(last(schedule)), duniform(1,
numberofitemtype));
                    }
                    if(distribution==2)
                    {
                        setnodenum(last(last(schedule)), pareto(1, 2));
                    }
                    nodeinsertinto(last(schedule)); //
                    nodeadddata(last(last(schedule)), DATATYPE_NUMBER);
                    setnodenum(last(last(schedule)), 1);
            }
}
```

APPENDICE III. Write_Stats

Di seguito è riportato il codice necessario a compilare la tabella di export contenente le statistiche delle simulazioni effettuate.

Ogni volta che viene lanciato questo script viene aggiunta una riga alla tabella "Results"; in questa tabella vengono riportati:

- Blocchi
- Livelli
- Canali per livelli
- Posti per canale
- Numero di shuttle
- Numero di satelliti
- Numero di itemtype
- Totale posti pallet
- Totale canali
- Item immagazzinati nella simulazione
- Tempo di simulazione
- Numero medio di missioni eseguite in un'ora
- Statistiche di utilizzo per lifter, shuttle e satelliti
- Distanza percorsa da tutte le navette

Questa tabella è utilizzata direttamente per popolare il database (per comodità è stato utilizzato Microsoft Excel).

```
/** Write Stats */


int rows = gettablerows("Results");
int x;
```

```
if(rows==1)
{
	x = 2;
}
else
{
	x = rows+1;
}
addtablerow("Results");

/**	DATI DI INPUT	*/

// Dati geometrici
settablenum("Results", x, 1, BLOCKS);
settablenum("Results", x, 2, LEVELS);
settablenum("Results", x, 3, BAYS);
settablenum("Results", x, 4, LOCATIONS);

settablenum("Results", x, 10, BLOCKS*LEVELS*BAYS);
settablenum("Results", x, 11, BLOCKS*LEVELS*BAYS*LOCATIONS);
settablenum("Results", x, 12, (BLOCKS-1)*getnodenum(node("Data/ESMART/numofshuttle",
model)));
settablenum("Results", x, 13, (BLOCKS-1)*getnodenum(node("Data/ESMART/numofsatellite",
model)));
settablenum("Results", x, 14, BLOCKS-1);

// Dati ESMART
settablenum("Results", x, 5, getnodenum(node("Data/ESMART/numofshuttle", model)));
settablenum("Results", x, 6, getnodenum(node("Data/ESMART/numofsatellite", model)));

// Dati Source
```

```
settablenum("Results", x, 7, getnodenum(node("Data/Storage/interarrivaltime", model)));
settablenum("Results", x, 8, getnodenum(node("Data/Storage/item", model)));
settablenum("Results", x, 9, getnodenum(node("Data/Storage/numberofitemtype", model)));



/**    RISULTATI SIMULAZIONE    */


// Tempo totale
settablenum("Results", x, 15, time());


// Tempo medio missione
rows = gettablerows("Warehouse");
int missions = 0;
int totalmissionstime = 0;
for(int y=1; y<=rows; y++)
{
        int missiontime = gettablenum("Warehouse", y, 13);
        if(missiontime>0)
        {
                missions++;
                totalmissionstime+=missiontime;
        }
}
settablenum("Results", x, 16, totalmissionstime/missions);
settablenum("Results", x, 17, getoutput(node("Storage Bay", model))*3600/time());



double idle;
double travel_empty;
double travel_loaded;
double waiting;
```

```
// Shuttle stats
int numofshuttles = groupnummembers("Shuttles");

idle = 0;
travel_empty = 0;
travel_loaded = 0;
waiting = 0;
for(int y=1; y<=numofshuttles; y++)
{
	idle += gettablenum(node(">stats/state_profiles/Shuttle States/profile", groupmember("Shuttles", y)), 5, 2);
	travel_empty += gettablenum(node(">stats/state_profiles/Shuttle States/profile", groupmember("Shuttles", y)), 2, 2);
	travel_loaded += gettablenum(node(">stats/state_profiles/Shuttle States/profile", groupmember("Shuttles", y)), 3, 2);
	waiting += gettablenum(node(">stats/state_profiles/Shuttle States/profile", groupmember("Shuttles", y)), 4, 2);
}
settablenum("Results", x, 18, idle/time()/numofshuttles*1000);
settablenum("Results", x, 19, waiting/time()/numofshuttles*1000);
settablenum("Results", x, 20, travel_loaded/time()/numofshuttles*1000);
settablenum("Results", x, 21, travel_empty/time()/numofshuttles*1000);

// Satellite stats
int numofsatellites = groupnummembers("Satellites");

idle = 0;
travel_empty = 0;
travel_loaded = 0;
waiting = 0;
```

```
for(int y=1; y<=numofsatellites; y++)
{
        idle += gettablenum(node(">stats/state_profiles/Satellite States/profile", groupmember("Satellites", y)), 5, 2);
        travel_empty += gettablenum(node(">stats/state_profiles/Satellite States/profile", groupmember("Satellites", y)), 2, 2);
        travel_loaded += gettablenum(node(">stats/state_profiles/Satellite States/profile", groupmember("Satellites", y)), 3, 2);
        waiting += gettablenum(node(">stats/state_profiles/Satellite States/profile", groupmember("Satellites", y)), 4, 2);
}
settablenum("Results", x, 23, idle/time()/numofsatellites*1000);
settablenum("Results", x, 24, waiting/time()/numofsatellites*1000);
settablenum("Results", x, 25, travel_loaded/time()/numofsatellites*1000);
settablenum("Results", x, 26, travel_empty/time()/numofsatellites*1000);
```

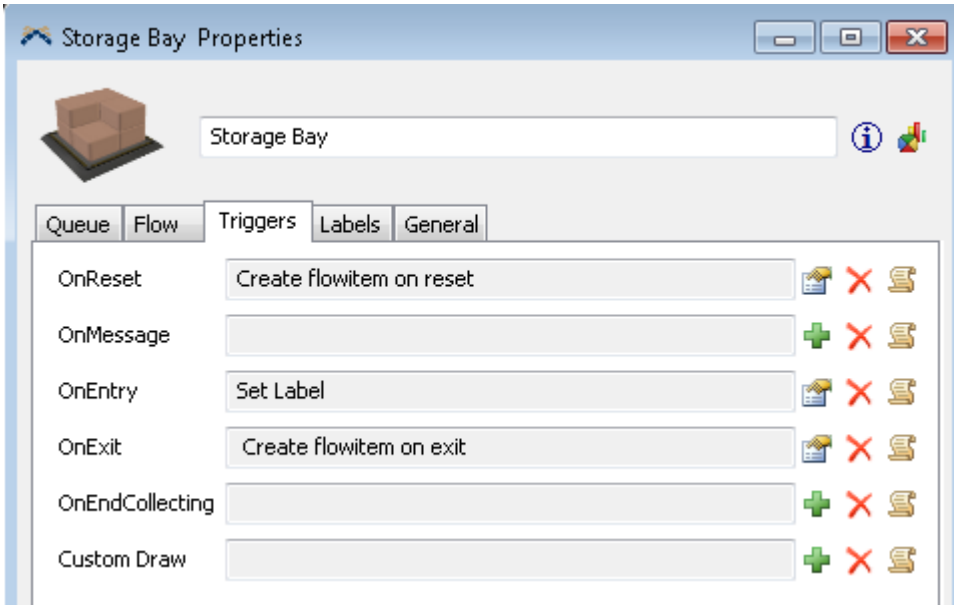
Per la scrittura delle percentuali di utilizzo è stato necessario moltiplicare per 1000 i risultati in quanto il comando settablenum() vincola la scrittura di un numero intero in tabella.

APPENDICE IV. Create_Flowitem on exit

Lo script seguente viene utilizzato nelle simulazioni in cui si vuol testare la massima velocità del magazzino creato di immagazzinare prodotti.

Si trova nel trigger OnExit all'interno della Storage Bay, ovvero l'elemento sorgente che alimenta il magazzino.

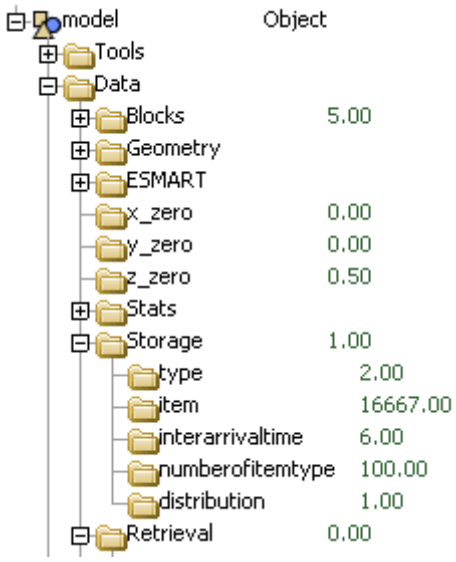
Il suo funzionamento è legato all'uscita di un item dalla sorgente stessa, pertanto deve essere combinato ad uno script che, al reset del modello, crea un primo flowitem da inserire nel magazzino



All'item, nel momento della sua creazione, vengono aggiunte una serie di informazioni che determinano itemtype e un nome identificativo univoco per permettere di tracciare la sua storia.

Questo nome viene riportato anche, al momento dello stoccaggio, all'interno della tabella "Warehouse" per consentire, con uno sviluppo ulteriore di questo modello, di gestire prelievi selettivi di prodotti.

Il numero di flowitem creati viene limitato sulla base di quanto riportato nel treenode, nello stesso nodo usato per definire il numero di item da stoccare in caso di simulazione tradizionale.

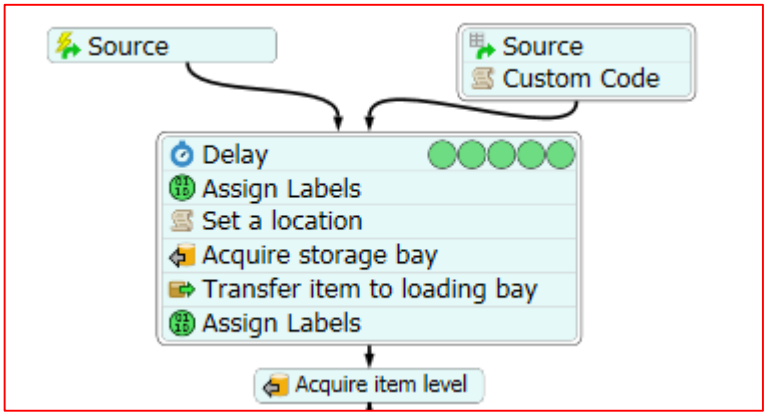



```
/** Create flowitem on exit*/

// Questo trigger viene attivato all'uscita di un item dalla sorgente di missioni di stoccaggio

treenode item = param(1);
int port = param(2);

treenode current = ownerobject(c);

// Il pallet utilizzato graficamente nelle simulazioni è il flowitem numero 11
```

```
int flowitemrank = 11;

treenode template = first(rank(node("/Tools/FlowItemBin",model()),flowitemrank));


// Limito la creazione di item al numero massimo stabilito nella GUI

if(getoutput(current)<getnodenum(node("/Data/Storage/item", model())))

{

        treenode newitem = insertcopy(template, current);

        setitemtype(newitem,                                                    duniform(1,
getnodenum(node("/Data/Storage/numberofitemtype", model()))));

}
```

APPENDICE V. Set_Location

Rappresenta l'inizio ed il cuore dei Process Flow. All'arrivo di un item viene avviata una missione di deposito; il primo passo da effettuare è appunto la scelta del dove collocare il prodotto.




```
/**Set a location*/

treenode current = param(1);
treenode activity = param(2);
treenode token = param(3);
treenode processFlow = ownerobject(activity);

int item_type = getlabel(token, "itemtype");
int ID = getlabel(token, "ID");
string name = getname(getlabel(token, "item"));
string channel;
int block;
int aisle;
```

```
int position;


// Cerco se ci sono canali non completi con stesso itemtype, se non ci sono prendo il primo libero
query("SELECT Channel, Block, Level, Column, Stor_Aisle FROM Warehouse WHERE Itemtype = $1 AND Item = $2 ORDER BY Priority", item_type, "");


if(getquerymatchcount()!=0)
{
        block   = getqueryvalue(1, 2);
        channel = getqueryvalue(1, 1);
        setlabel(token, "channel", channel);
        setlabel(token, "block",   block);
        setlabel(token, "level",   getqueryvalue(1, 3));
        setlabel(token, "bay",     getqueryvalue(1, 4));
        aisle = getqueryvalue(1, 5);

}

// Se non ci sono canali dedicati
else
{
        query("SELECT Channel, Block, Level, Column, Stor_Aisle FROM Warehouse WHERE Itemtype = 0 AND Item = $1 ORDER BY Priority", "");
        channel = getqueryvalue(1, 1);
        block   = getqueryvalue(1, 2);
        setlabel(token, "channel", channel);
        setlabel(token, "block",   block);
        setlabel(token, "level",   getqueryvalue(1, 3));
        setlabel(token, "bay",     getqueryvalue(1, 4));

        // Eseguo QUERY tra le baie di carico per identificare se una è occupata
```

```
int flag = 0;
string QUERY;
QUERY = concat("WHERE pulled.Aisle = ", numtostring(block));
if(listpull("Storage_Bays", QUERY, 0, 1)!=0)
{
        flag += 1;
}
QUERY = concat("WHERE pulled.Aisle = ", numtostring(block-1));
if(listpull("Storage_Bays", QUERY, 0, 1)!=0)
{
        flag += 2;
}

// Se sono entrambe libere o entrambe occupate la assegno a caso
if(flag == 0 || flag == 3)
aisle = duniform(block-1, block);

// Se è libera solo la prima
if(flag == 1)
aisle = block;

// Se  libera solo la seconda
if(flag == 2)
aisle = block-1;

// Se AISLE è uguale a zero, in automatico assegno la missione alla prima AISLE
if(aisle==0)
aisle = 1;

// Se AISLE è uguale al numero di blocchi, assegno la missione all'ultima AISLE
if(aisle==BLOCKS)
```

```
		aisle = BLOCKS-1;


}


		// Imposto l'Itemtype di questo canale e il corridoio di approvvigionamento
		// Per farlo, trovo quante posizioni ha il canale scelto e con un ciclo a partire dal primo risultato
		// inserisco l'itemtype in tutte le reghe
		query("SELECT Position FROM Warehouse WHERE Channel = $1", channel);
		int results = getquerymatchcount();
		if(results==0)
		{
			stop();
		}

		int first_position = getquerymatchtablerow("Warehouse", 1);
		for(int x = first_position; x<first_position+results; x++)
		{
			settablenum("Warehouse", x, 8, item_type);
			settablenum("Warehouse", x, 9, aisle);
		}



// IDENTIFICO LA LOCATION
if(aisle==block) // Se riempo da sinistra parto dalla posizione 1
{
		query("SELECT Position FROM Warehouse WHERE Channel = $1 AND Item = $2 ORDER BY Position ASC", channel, "");
		position = getqueryvalue(1, 1);
}
else // Se riempo da destra parto dall'ultima posizione
```

```
{
        query("SELECT Position FROM Warehouse WHERE Channel = $1 AND Item = $2 ORDER BY Position DESC", channel, "");
        position = getqueryvalue(1, 1);
}

// Salvo le label mancanti
setlabel(token, "aisle", aisle);
setlabel(token, "position", position);

// Salvo ora in tabella l'item nella cella assegnata
query("SELECT Item FROM Warehouse WHERE Channel = $1 AND Position = $2", channel, position);
int row = getquerymatchtablerow("Warehouse", 1);
settablestr("Warehouse", row, 7, name);

// Assegno lo scaffale al token
treenode rack = node(concat("/Rack", numtostring(block), "_", numtostring(position)), model);
setlabel(token, "rack", rack);
```