UNIVERSITÀ DEGLI STUDI DI PADOVA

UNIVERSITÁ DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria Industriale DII

Dipartimento di Tecnica e Gestione dei Sistemi Industriali DTG

Corso di Laurea Magistrale in Ingegneria Meccanica

SIMULATION-BASED EVALUATION OF TRADITIONAL AND MOBILE RACKS PICKING SYSTEMS

Relatrice Ch.ma Prof. Battini Daria

Correlatore Ch.mo Prof. Sgarbossa Fabio

Correlatore Ch.mo Dr. Grosse Eric

Lucchini Giacomo 1156523

Anno Accademico 2018/2019

To my mum,

for having been close to me at all times

and for having made me become the man I am today.

A mia mamma,

per essermi rimasta vicino in ogni momento

e per avermi fatto diventare l'uomo che sono oggi.

Ackowledgements

Within this whole piece of research, this is the page where I get express my personal gratitude to all those people who had been an important part of my life and this work, wouldn't have been possible without their support and constant motivation.

I would like to thank my Italian and German supervisors, Prof. Battini Daria, Prof. Sgarbossa Fabio and Dr. Grosse Eric, for their advice and support during the work. The gave my freedom to pursue my ideas, work at my own pace and look for solution to the problem through my approach. I enjoyed spending time learning and gaining experience under them. A special thanks to Prof. Sgarbossa Fabio who gave me the possibility to work for six months in Darmstadt, where I learned how to work harder and where I had a great time. An other special thanks to Dr. Grosse Eric who has closely followed this work. I am highly indebted to him for clarifying my doubts and for providing suggestions and criticisms on my work.

Thanks to all online plantsimulation forum and in particular to Steffen who taught me to use the program with which I performed my work.

A big thanks goes to my parents and my girlfriend, who have always supported me in good and bad times, both with advice and finance. They have always believed in me and they have never made me feel alone.

Thank to my friends Cristian, Paolo, Giorgio, Federica, Giulio, Andrea and all the others. I would like to thank my batchmates Soumya, Martin, Min, Ceren, Abdel, Mark, Sumit and Parastoo from my language course. I have learnt a lot about different nations and their cultures through them. Learning in this course wouldn't have been so fun without them. Although our friendship was just a brief time, I shared some greatest moments of my life with them.

To finish it up, I would like to thank myself. Because of all the sacrifices I have made, for every weekend of four years to work to be independent. Nobody sees all the work behind a result, but the result is always worth it.

Abstract

In the operation procedure of distribution center, picking is considered as the primary field to improve operation efficiency of warehousing. The growth in online retailing has fuelled the order fulfilment business, where thousands of orders are now picked in warehouse largely by human pickers. This has inevitably changed the way in which the orders processing is performed by suppliers, who are face the need to respond very rapidly and flexibly. This trend strongly affects the configuration of picking warehouse and of the activity that have to be carried out within them, with larger pick volumes that have to be satisfied within shorter time windows.

The order picking is the most labour-intensive, time and cost consuming activity in warehouse, it includes 55% of the whole warehouse operating costs. Obviously, the travel and search times of pickers are an unproductive parts of the order picking process, then to improve performance and at same time reduce costs, these times must be reduced. We need to use strategy to obtain some benefits in this sense. A paperless picking system is constituted of a set of device designed and adopted to facilitate the work of the operators, mostly in terms of getting information on the product to be picked and finding the corresponding storage location. Another method to increase productivity, at the expense of reducing the times involved in picking activity, is to adopt a picking system different from the traditional one (picker-to-parts) and precisely a parts-to-picker method. The basic aim of these system is to automatically move the stock keeping units to the pickers, so that they can concentrate on the productive part of their employment. A new frontier of parts-to-picker system is the mobile racks method in robotic environment, also called robotic mobile fulfilment system (RMFS). This is a more recent AGVs application method.

Then, to reduce the search time can be used support systems to drive and control the pickers during their work or to reduce both the search and displacement time have been changed the type of order picking system from picker-to-parts to parts-to-picker. Moreover, to reduce the travel time, keeping the same picking system, can be modified the number of orders picked simultaneously, using the batching policy linked to a time window, or can be changed the routing method of pickers.

The aim of this work is compare, through simulations, the performance and economic evaluation of three different picking systems: the traditional one, the traditional one with the help of the paperless support system, precisely with pick-to-light support, and the mobile racks system, precisely the most famous called Kiva system.

Table of contents

List of figures

List of tables and charts

Introduction

In the operation procedure of distribution center, picking is considered as the primary field to improve operation efficiency of warehousing, this has become more and more true with the beginning of the era of e-commerce and the evolution of customer orders. The growth in online retailing has fuelled the order fulfilment business, where thousands of orders are now picked in warehouse largely by human pickers (Bozer and Aldarondo, 2017). This has inevitably changed the way in which the orders processing is performed by suppliers, who are face the need to respond very rapidly and flexibly. This trend strongly affects the configuration of picking warehouse and of the activity that have to be carried out within them, with larger pick volumes that have to be satisfied within shorter time windows (De Koster et al., 2007; Bartholdi and Hackman, 2011).

The order picking is the most labour-intensive, time and cost consuming activity in warehouse, it includes 55% of the whole warehouse operating costs (Tompkins et al., 2010). This cost can be divided further: travelling takes the 55% of the time, searching takes the 15%, picking the 10% and paperwork and other activities the 20%. Searching items and travel may account for 70% of the time required to fill orders (Franzelle, 2002). Obviously, the travel and search times of pickers are an unproductive parts of the order picking process, then to improve performance and at same time reduce costs, these times must be reduced.

According to De Koster et al. (1998) paperless order picking systems can be a useful strategy to obtain some benefits in this sense. A paperless picking system is constituted of a set of device designed and adopted to facilitate the work of the operators, mostly in terms of getting information on the product to be picked and finding the corresponding storage location (Battini et al. 2014). A new frontier of paperless picking is represented by the use of important devices that have been developed to speed up picking activities and to avoid picking errors, such as LED display or digital screens, voice-activated devices (voice picking), wireless appliances and lighting systems (pick-to-light). Picker and warehouse staff are connected online with the warehouse information system, enabling updated stock information, immediate reactions to particular situations and the real-time monitoring of operational status, leading to an overall productivity increase.

Another method to increase productivity, at the expense of reducing the times involved in picking activity, is to adopt a picking system different from the traditional one (picker-to-parts) and precisely a parts-to-picker method (Boysen et al., 2016). The basic aim of these system is to automatically move the stock keeping units to the pickers, so that they can concentrate on the productive part of their employment. A new frontier of parts-to-picker system is the mobile racks method in robotic environment, also called robotic mobile fulfilment system (RMFS). This is a more recent AGVs application method (Lamballais et al., 2017).

Then, to reduce the search time can be used support systems to drive and control the pickers during their work or to reduce both the search and displacement time have been changed the type of order picking system from picker-to-parts to parts-to-picker. Moreover, to reduce the travel time, keeping the same picking system, can be modified the number of orders picked simultaneously, using the batching policy linked to a time window, or can be changed the routing method of pickers (De Koster et al., 2006).

The aim of this work is compare, through simulations, the performance and economic evaluation of three different picking systems: the traditional one, the traditional one with the help of the paperless support system, precisely with pick-to-light support, and the mobile racks system, precisely the most famous called Kiva system. The results that determine the performance of the work systems were found with a 6-month simulation created on PlantSimulation14, a Siemens program. The picking systems have been implemented in a warehouse with dimensions suitable for being able to use also a parts-to-picker method. In order to have as realistic as possible simulation, these dimensions and other parameters, such as the number of SKUs and daily orders, were searched in the literature (Lamballais et al., 2015; Liernet et al., 2017; Bozer et al., 2018; Zou et al., 2017; Merschformann et al. 2017; Merschformann et al. 2018; Bahrami et al., 2017; Petersen and Aase, 2003; De Koster et al., 2011; Horvat, 2012) and then assumed the average values.

An important characteristic of this work is the field of application of picking system, the randomly created orders, equal for each system, are meant to be online food retailing orders. Then with a large number of order lines per order.

The performances calculated at the end of the simulation for each configuration of the picking systems are:

- Elapsed time to complete daily order
- Picker throughput calculated in order line per hour
- Average time for order
- Number of shelves visited to complete an order
- Times and percentages for each activity of the picker (getting information, searching, picking, confirm and travel)
- Picker utilization
- Times and percentages for each activity of the Kiva robot (waiting and working)
- Kiva robot utilization

The economic evaluation has been calculated following the procedure proposed in the article “A comparative analysis of different paperless picking system” performed by Battini et al.,2014. The result of the economic evaluation is an hourly cost in function of number pick for hour, calculated also with the variation of some parameters.

1. Warehouse picking design

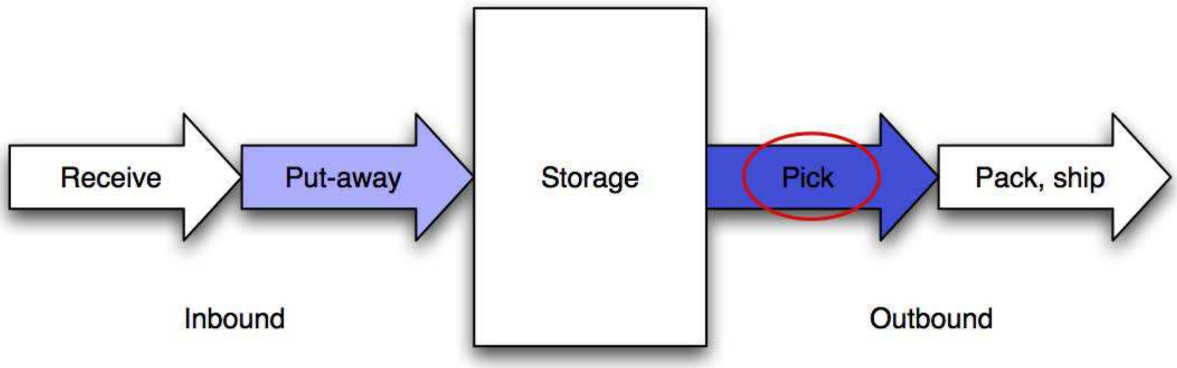
The aim of this chapter is to give the reader the basic knowledge on warehouse. Here, the steps that the items must pass before being shipped to the customer will be listed and explained, focusing on the phase that requires the greatest effort and cost. This phase is the picking activity. After that, will be defined the different policies that could be used to determine order picking system, layout design, storage assignment, batching orders and routing methods.

1.1. Warehouse flow

In general, warehouse reorganize and repackages products. The products typically arrive packaged and leave the warehouse packaged, but in a smaller scale. The reason why the products arrive in a warehouse in lot is that it is faster and simpler to handle lot than each. A golden rule, suggested by Bartholdi and Hackman (2017) is "the smaller is the handling unit, the greater is the handling cost".

More or less in every warehouse there is a common flow of materials: receive bulk shipments and store them for quick retrieval; then, in response to a costumer's order, products are picked automatically or by an operator and they are shipped to the customer as soon as possible. The flow of material in a warehouse can be summarised in two parts: inbound and outbound processes. In inbound processes the two main activities are receiving and put-away. In outbound processes the main activities are order-picking, checking, packaging and shipping. Between inbound and outbound processes there is storage, where products are stocked. This material flow is summarized in figure 1.1.

Fig. 1.1. - Warehouse flow. Circled in red its most demanding part, that requires most of the time and cost



A process must flow as fast as possible and without interruptions, because each time a product is put down, it means it has to be picked up again later: double-handling is a loss of time, energy and money. The effect of double-handling is wider if we think that we have to handle thousand of SKUs[1] per hour. Then if it is possible to avoid double handling, it is better to do it to save money and therefore gain more.

From here on, will be explain widely four part of flow mentioned before. After that, will be focus more on picking, which is the most labour-intensive activity in warehouse.

1.1.1. Receiving

Material is received after an order has been done. Receiving begins with a list, which shows the schedule of arrivals; this list lets the warehouse to know exactly when the trucks are arriving and in which order. As soon as a product arrives, it is registered in the database, it is checked and stocked. Products are usually shipped in pallets: it means they are held together on a platform 800x1200 (European pallet) or 1016x1219 (American pallet) or 1165x1165 (Australian pallet); the main advantage to use pallet is that loads and unloads of

[1] A stock keeping unit, or SKU, is the smallest physical unit of a production that is tracked by an organization.

trucks are faster. Along the flow these pallets will be disassembled in smaller groups of products.

The cost of receiving is around the 10% of the whole cost.

1.1.2. Put away

Put away is a very important issue in warehouse. Before doing it, is very important to decide the location to stock it. The place where they are stocked determines how quickly products can be reached and the later cost of product handling. The location of products is essential to write the picking list, which shows to the pickers where retrieving the product. As soon as a product is put away, it has to be registered on a software, which creates the picking list.

Put away typically accounts the 15% of the warehouse costs, but this cost can be reduced if the locations to stock pallets are chosen well.

1.1.3. Checking and packing

In general, after all the products of an order have been picked, every order has to be checked to control if it is complete and accurate. Order accuracy is one of the most useful indicator to measure the level of service given to a customer. Inaccurate orders lead to problems: the customers can be annoyed and they could send the products back, generating a return, which is very expensive to handle (up to 10 times the costs of normal shipping). After the control, is always better to pack all the parts of an order together. The customer often requires it, because he can shorten the time of shipping, unloading and handling.

1.1.4. Shipping

When the products are ready and packed together, they can be shipping. In general, shipping works with larger units than picking, because all the items are consolidated in few containers. Depending on the type of pallet and the type of truck, a different number of pallet can be shipped. As soon as the truck leaves the factory, the departure is register and customer is warned.

1.1.5. Order-picking

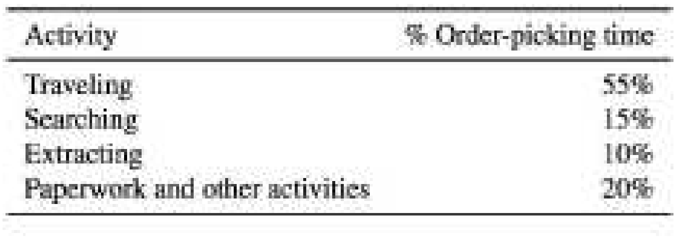
Order picking is the most labour-intensive activity in warehouse. It also determines the service seen by the customers. It must be flawless and fast. It can be done by a person or by a machine. Once a customer orders some products, it is checked if these products are available in the warehouse; if they are, the order can be accepted. As soon as the order is accepted, a software called warehouse management system (WMS) creates a picking list to guide order-pickers. The software produces all the shipping documentation and the shipping schedule and coordinates all the different activities in the warehouse.

The action of picking can be divided in three phases:

- Travel to the storage location: the operator, thanks to the picking list, has to reach the right storage location.
- Local research: once the operator has reached the right location, he has to find the exact position of the product. The smaller is the product, the more difficult is the operation. That is the part of picking in which is simpler to make mistake, so the operator has to pay a lot of attention, not to pick the wrong SKU.
- Reach, grab and put: it is when the operator takes and put in the container the products requested by the customer. It is only part of picking which is value-adding.

Order picking includes 55% of the whole warehouse operating costs. This cost can be divided further: travelling takes the 55% of time, searching take the 15%, picking the 10% and paperwork and other activities the 20%. The division of the cost is show in Fig. 1.2.

Fig. 1.2. - Division of costs in order-picking. (from Bartholdi and Hackman, 2017)



Activity	% Order-picking time
Traveling	55%
Searching	15%
Extracting	10%
Paperwork and other activities	20%

The object that catches the eye immediately is the cost of travelling, which is then major cost. This means that, to reduce dramatically the cost of a warehouse, the first thing to do is to reduce the travelling cost, because it is the biggest one. To reduce the travelling cost it is important to optimize the layout of the warehouse and to have an efficient picking list. All the actions that are not adding value to the product can be considered waste and they have to be reduced a minimum or eliminated.

With the picking list, pickers know the number of products they have to pick, where to go and in which order pick products.

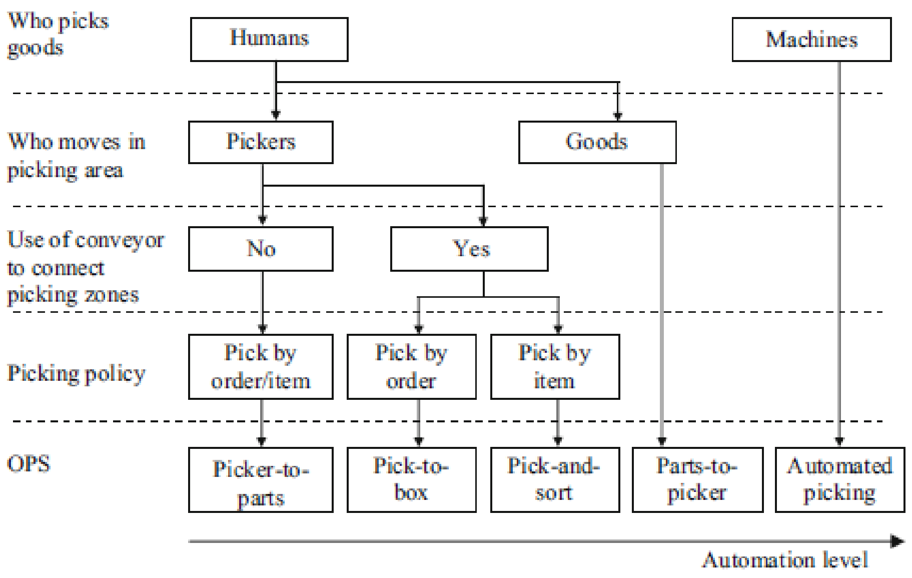
1.2. Order picking system classification

Within a warehouse, order picking is the process of clustering and scheduling the customer orders, and of consequently picking the articles from the various storage locations to fulfil such customer orders. An order picking system can be different types, each type having peculiar characteristics which make it more or less suitable for different fields of application. To better identify the application field for each order picking system, is proposed a classification that focuses chiefly on the operational policy rather than on the specific equipment type adopted (Dallari et al., 2007). Classification of order picking system (fig. 1.3.):

- "Picker-to-parts" system
- "Pick-to-box" system
- "Pick-and-sort" system
- "Parts-to-picker" system

- "Completely automated picking" system

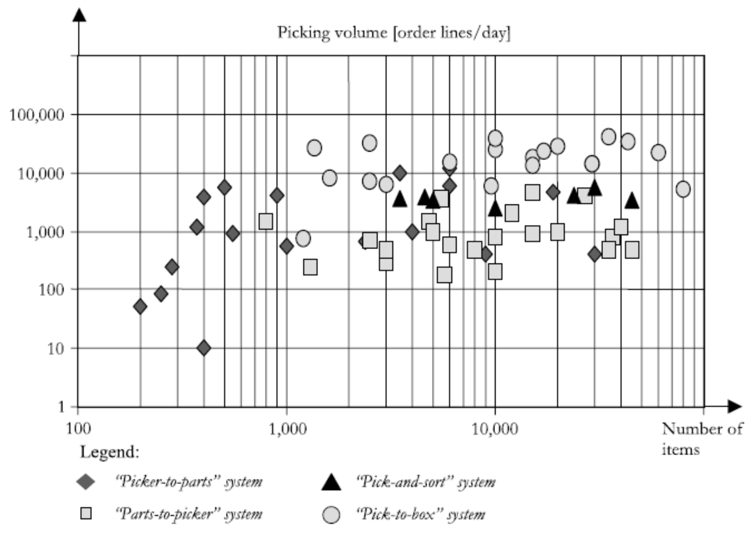
Fig. 1.3. - Classification of order picking system



Order picking system are classified accordingly with four main decisions: who pick goods (humans/machines), who moves in picking area (pickers/goods), if conveyors are used to connect each picking zone and which picking policy is employed (pick by order or by item). Automation level increases, ranging from the "picker-to-parts" system to the "completely automated picking" one.

From the research of Dallari (Dallari et al., 2007) can be found a field of use for every order picking system, considering the number of items stored and the number of daily picking of a warehouse (fig.1.4.). Collected data represent average values and not distributions, as it has been difficult to obtain data with higher detail level.

Fig.1.4. - Empirical analysis of investigated order picking system (Dallari et al., 20017)



Each order picking system will be briefly described on the basis of both the equipment component and the resource requirements, the "completely automated picking" system will not be considered because it is employed in very limited contexts.

1.2.1. "Picker-to-parts" system

"Picker-to-parts" system represents the very large majority of picking systems in warehouse. It can be considered as the basic system for the picking activity. In such system, pickers walk or drive along the aisles to pick items, completing a single order or a batch of multiple orders, depending on the order picking policy. In the batch picking policy, the picked items are immediately sorted by the picker. We can distinguish two types of picker-to-parts systems: low- and high-level picking. In low-level customer orders are small, urgent and different from each other. Therefore, it is normal that items are picked from picking location while the picker walks along the aisles. In high-level customer orders are typically large and of similar products. Each picker makes a lot of picks in a short distance, then picking location are visited by pickers on board of an order-pick truck.

In the "picker-to-parts" system, further optimisation can be carried out by means of routing algorithms, items allocation policies and paperless operations using support systems to drive and control the pickers during their work, like:

- Barcodes handheld
- RFID tags handheld
- Voice picking
- Pick to light
- RFID pick to light

Some of these support systems will be used within this work and explained in the next chapter.

1.2.2. "Pick-to-box" system

"Pick-to-box" system, also known as "pick-and-pass" system, divides the picking area in zones, each of them assigned to one or more pickers. All the picking zones are connected by a conveyor on which boxes filled up with picked items are placed, each of them corresponding (partially or completely) to a customer order. Customer orders are sequentially picked zone by zone. Therefore, a line-end sorting per each order is not necessary, as the orders have already been prepared in boxes. The resulting advantages of separating the forward area in multiple picking zones mainly lay in the reduction of the overall picker travel time. The costs and complexity are related to workload balancing among the multiple picking zones.

1.2.3. "Pick-and-sort" system

Operators in the picking area retrieve the amount of each single item resulting from the batching or multiple orders and put it on a takeaway conveyor connecting the forward area with sorting area. A computerised system then determines the destination bay for each item, each destination bay refers to an individual customer order. This system, typically, works with pick waves, where all of the orders in a pick wave are completely sorted before releasing the following pick wave. The batch size is consistently high, at least 20 customer orders per pick wave.

1.2.4. "Parts-to-picker" system

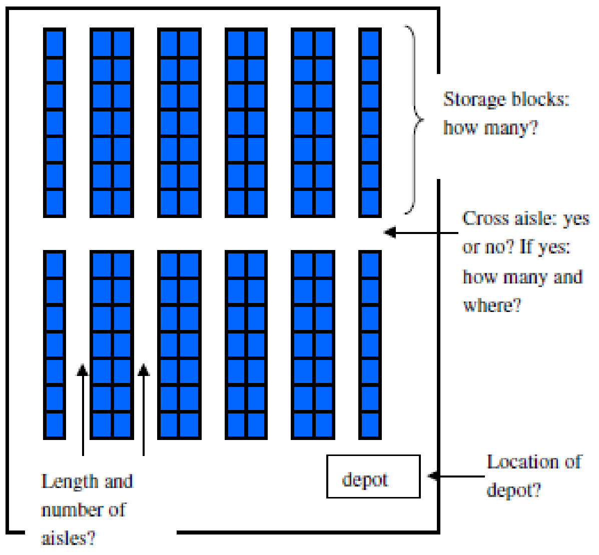
In "parts-to-picker" system, an automatic device brings unit loads from the storage area to the picking station, where the pickers select the required amount of each item. Afterward, the unit loads, if not empty, are conveyed back to the storage area. The basic aim of these systems is to automatically move the stock keeping units (SKUs) to the pickers, so that they can concentrate on the productive part of their employment. Unproductive travel times reduce productivity per picker, so that picker-to-parts system require a larger workforce for realizing the same output compared to parts-to-picker system. Potential equipment types for this storage area are:

- Carousels
- Modular vertical lift modules
- Miniloads
- AS/RS
- Mobile racks in robot environment (this will be used within this work and explained in the next chapter)

1.3. Layout design

In the context of order picking, the layout design concerns two sub-problems: the layout of the facility containing the order-picking system and the layout within the order-picking system. The first problem concerns the decision of where to locate various department (receiving, picking, storage, sorting, etc.). The common objective is minimising the handling cost, which in many cases is represented by linear function of travel distance. the second problem is usually called internal layout design (fig. 1.5.) or aisle configuration problem. It concerns the determination of the number of blocks, and the number, length and width of aisles in each block of picking area. The common goal, again, is the travel distance.

Fig. 1.5. - Typical layout decision in order picking system design


Storage blocks: how many?
Cross aisle: yes or no? If yes: how many and where?
Length and number of aisles?
depot
Location of depot?

1.4. Storage assignment

Products need to be put into storage locations before they can be picked to fulfil customer orders. A storage assignment method is a set of rules which can be used to assign products to storage locations. Before such an assignment can be made, however, a decision must be made which pick activities will take place in which storage system.

1.4.1. Forward-reverse allocation

In order to speed up the pick process, it is in many cases efficient to separate the bulk stock (reserve area) from the pick stock (forward area). The size of the forward area is restricted: the smaller the area, the lower the average travel times of the order pickers will be. It is important to decide how much of each SKU is placed in the forward area and where in the area it has to be located. Dividing a SKUs inventory over multiple areas implies regular internal replenishments from the reserve to the forward area. One of the trade-offs to be made is then to balance additional replenishment efforts over extra pick effort savings. It may even be advantageous to store some of the SKUs only in the reserve area, for example if demand quantities are high or if demand frequencies are low. Furthermore, replenishments are often restricted to times at which there is no order picking activity, which gives additional constraints.

A concept closely related to the forward-reverse problem is dynamic storage. It aims at making the pick area very small in order to reduce travel time and bringing the SKUs to the storage dynamically, just in time for the pick (like "part-to-picker" system). The number of locations available in the forward are is usually smaller than the total number of SKUs. As these systems are capable of achieving very high picker productivity, they are becoming more and more popular.

1.4.2. Storage assignment policies

There are numerous ways to assign products to storage locations within the forward storage areas. Five, frequently, used types of storage assignment are:

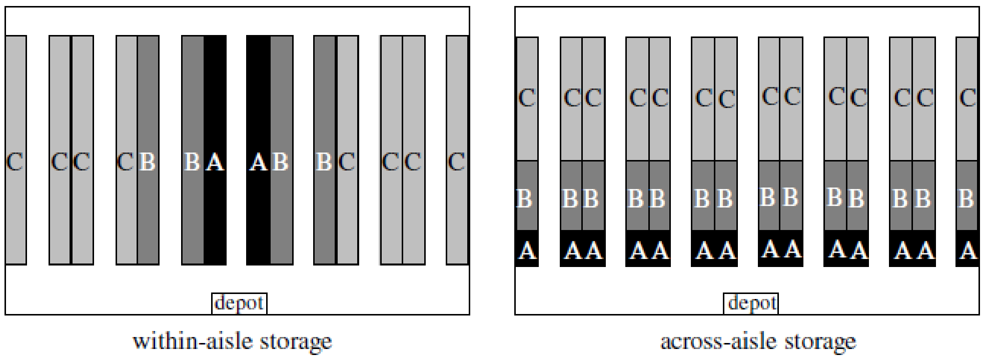
- random storage: every incoming pallet is assigned a location in the warehouse that is selected randomly from all eligible empty locations with equal probability;
- closest open location storage: the first empty location that is encountered by the employee will be used to store the products;
- dedicated storage: each product has a fixed location;
- full turnover storage: this policy distributes products over the storage area according to their turnover, the products with the highest sales rates are located at the easiest accessible locations and near the depot;
- class-based storage (policy used in this work, it will be explained in the next paragraph)

All storage assignment policies discussed so far have not entailed possible relations between products. For example, customers may tend to order a certain product together with another product. In this case, it may be interesting to locate these two products close each other. An example of this is called family-grouping, where similar products are located in the same region of the storage area. Clearly, grouping of products can be combined with some of the previously mentioned storage policies.

1.4.3. Class-based storage

The concept of class-based storage combines some of the methods mentioned so far. In inventory control, a classical way for dividing items into classes based on popularity is Pareto's method. The idea is to group products into classes in such a way that the fastest moving class contains few percentage of products stored but contributes to high percentage of the turnover. Each class is then assigned to a dedicated area of the warehouse. Storage within an area is random. Classes are determined by some measure of demand frequency of the products. Fast moving items are generally called A-items. The next fastest moving category of products is called B-items, and so on. Often the number of classes is restricted to three (fig 1.6.).

Fig. 1.6. - Illustration of two ways to implement class-based storage


C
C C
C B
B A
A B
B C
C C
C
depot
within-aisle storage
C C C C C C C C C C C C C C C
B B B B B B B B B B B B B B B
A A A A A A A A A A A A A A A
depot
across-aisle storage

1.5. Batching

When orders are fairly large, each order can be picked individually (one order picking tour). This way of picking is often referred as the single order picking policy (or discrete picking or pick-by-order). However, when orders are small, there is a potential for reducing travel times by picking a set of orders in a single picking tour. Order batching is the method of grouping a set of orders into a number of sub-sets, each of which can then be retrieved by single picking tour. There are basically two criteria for batching: the proximity of pick locations and time windows. Proximity batching assigns each order to a batch based on proximity of its storage location to those of other orders.

Under time windows batching, the orders arriving during the same time interval, called time window, are grouped as a batch. These orders are then processed simultaneously.

1.6. Routing methods

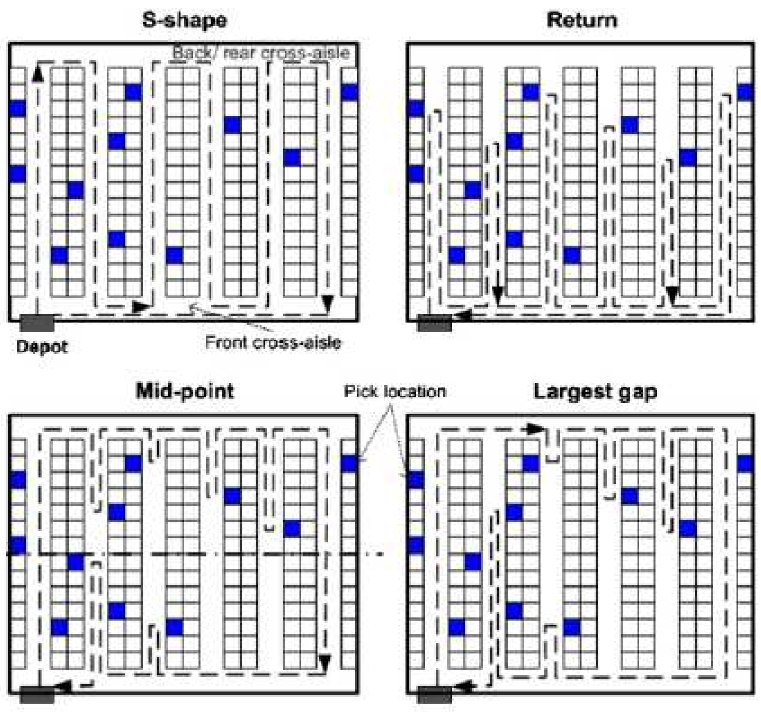
The objective of routing policies is to sequence the items on the pick list to ensure a good route through the warehouse. The problem of routing order pickers in a warehouse is actually a special case of travelling salesman problem. A salesman, starting in his home city, has to visit a number of cities exactly once and return home. He knows the distance between each pair of cities and wants to determine the order in which he has to visit the cities such that the total travelled distance is as small as possible. Clearly, the situation of the travelling salesman has many similarities with that of an order picker in warehouse. The order picker starts at the depot, where he receives a pick list, has to visit all pick locations and finally has to return to the depot.

In practice, the problem of routing order pickers in a warehouse is mainly solved by using heuristics. Some heuristic routing order are (fig. 1.7.):

- S-shape or traversal method: routing order pickers by using the S-shape method means that any aisle containing at least one pick is traversed entirely. Aisles without picks are not entered. From the last visited aisle, the order picker returns to the depot;
- Return method: where an order picker enters and leaves each aisle from the same end. Only aisles with picks are visited;
- Midpoint method: essentially divides the warehouse into two areas, picks in the front half are accessed from the front cross aisle and picks in the back half are accessed from the back cross aisle;

- Largest gap strategy: is similar to the midpoint method except that an order picker enters an aisle as far as the largest gap within an aisle. The gap represents the separation between any two adjacent picks, between the first pick and the front aisle, or between the last pick and the back aisle

Fig. 1.7. - Example of heuristic routing methods



1.7. Warehouse Management System (WMS)

A warehouse management system or WMS primarily aims to control the movement and storage of materials within a warehouse and process the associated transaction, including shipping, receiving, put-away and picking. A warehouse management system is a database driven computer application, which is used by logistics personnel to improve the efficiency of the warehouse by directing cutaways and to maintain accurate inventory by recording warehouse transactions. The system also direct and optimize stock based on real-time information about the status of bin utilization. It often utilizes auto ID data capture technology, such as barcode scanners, mobile computers, wireless LANs and radio-frequency identification (RFID) to efficiently monitor the flow of products. Once data has been collected, there is either batch synchronization or a real-time wireless transmission, with the central database. The database can then provide useful reports about the status of goods in the warehouse.

The primary function of a warehouse control system is to receive information from the upper level host system, most often being the warehouse management system, and translate it for the daily operations. Warehouse control system is usually the interface that is used to manage processes, people and equipment on the operational level.

Based on warehouse control system, literature distinguishes three types of warehouse management system (Ramaa et al., 2012):

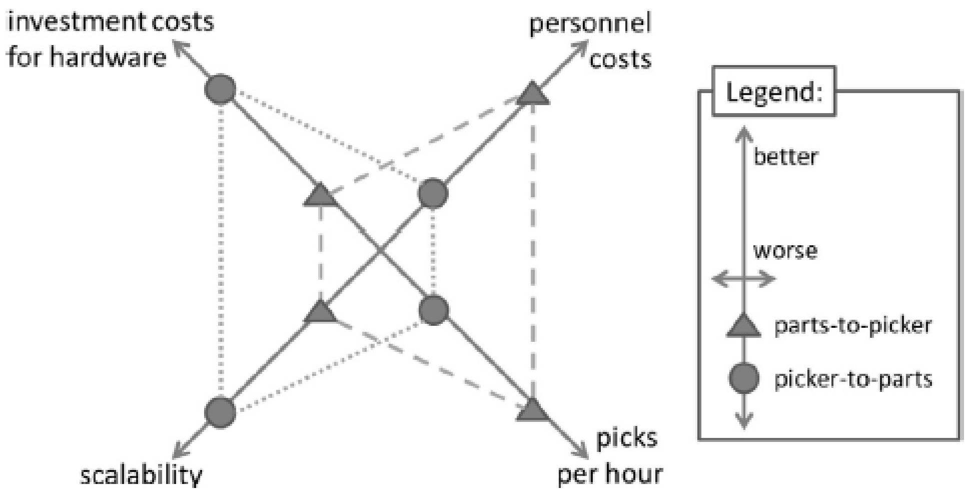
- Basic WMS: this system is apt to support stock and location control only. It is mainly used to register information. Storing and picking instruction may be generated by the system and possibly displayed on terminals. The warehouse management information is simple and focuses on throughput mainly.
- Advanced WMS: above the functionality offered by a basic WMS, an advanced WMS is able to plan resources and activities to synchronize the flow of goods in the warehouse. The WMS focuses on throughput, stock and capacity analysis.
- Complex WMS: with a complex WMS the warehouse can be optimized. Information is available about each product in terms of where it is located (tracking and tracing), what is its destination and why (planning, execution and control). Further, a transportation, dock door and value added logistics planning which help to optimize the warehouse operations as a whole.

Warehouse management system can be stand alone systems or modules of an ERP (Enterprise resource planning) system or supply chain execution suite.

2. Systems description

In this chapter are described the picking systems that will be used in this work. They are, for the picker-to-parts category, barcode handheld system and pick-to-light system, while for the parts-to-picker category is mobile racks system in robot environment. The following figure 2.1. represents the main pros and cons for picker-to-parts and parts-to-picker order picking.

Fig. 2.1. - Pros and cons for picker-to-parts and parts-to-picker



2.1. State of art of the systems supporting picking activities

As warehouse manual picking is considered one of the most critical warehouse activities, many support systems have been developed, able to drive and control pickers during the work. One of the first devices adopted to facilitate picking process and one of the most widespread, too, is the handheld barcode scanner. All the stock keeping units are tagged with barcode, that are scanned by the operator during the picking of the particular SKU. In this way, the picking information are immediately communicated to the warehouse information system. Handhelds are often able to emit acoustic signal, too. This feature generally helps to understand whether the scanner has correctly read the barcode, but it could be used also to notify that the product scanned is exactly what the picker has to take. Such system can be combined with paper picking list. Once an item has been picked, the screen of the handheld shows the following product to take. Recently, Handhelds RFID scanners are also available. The operating principle is similar to what just presented, except that the SKUs are tagged with RFID passive tags instead of barcodes. The working frequency is LF (low frequency) or HF (high frequency), with small reading distances of the handhelds.

RFID (radio frequency identification) system is a wireless communication system in which the radio link between the base station and the transponders are provided by the modulated back-scattered waves. A basic RFID system consist of an antenna or coil, a transceiver and a transponder tag. Such tags consist of an antenna and a chip, electronically programmed with unique information, that are often attached to object in order to allow their identification. In fact, they can store data related to the products but also, more simply, a unique serial number that creates the connection to actual data in database. According to their application, the transponders can be of two types: active or passive. The first ones have an own power supply that enables them to transmit at higher power levels, hence to be read and written at greater distances (also over 100 m). They are typically larger and more expensive. On the contrary, passive tags obtain their energy from electromagnetic field of the reading device, so they are very small and economic. In warehouse and manufacturing passive tags are the most widespread. A RFID system can be different in terms of the frequency large in which it operates, too. In particular, there are three worldwide established frequencies: low frequency (LF), high frequency (HF) and ultrahigh frequency (UHF). Every working frequency is more suitable for some applications than for others. Low frequency system is well-suited to industrial use, above all when working near metals and water is needed. High frequency system are characterised by greater ranges and higher reading speeds, the simultaneous reading of multiple tags is possible. UHF system are more suitable for warehousing and good tracking.

The most widespread techniques are pick-to-voice and pick-to-light. A pick-to-voice system is a voice-directed system that uses speech recognition to allow warehouse operators to communicate with warehouse management system.

Pickers are equipped with a headset and microphone to receive instruction of picking by voice, and to verbally confirm their actions back to the system. The warehouse operator reads back the last digits of the item he has picked so that the system can check whether the correct item has been selected, then it can give the following instruction. On the other hand, in a pick-to-light- system operators are guided by lights that are installed on the warehouse shelving. Each stock location has one light that turns on if the operator has to pick the corresponding product. In order to complete every single pick, the picker has to press the button of the interested stock location. If the simultaneous work of more than one piker in the same warehouse are is needed, such system has to be integrated with paper picking list, so that every picker can understand which are the lights turned on for his order. An example of pick-to-light using RFID has been presented in 2011. In the reported test case RFID readers are installed in some points beneath the conveyor belt, while RFID passive tags are attached to the plastic buckets in which workers place the products required to fulfil the orders. When the bucket reaches a RFID reader point, this sends the signals of turning on of the lights of the required products, so that the operator can easily and quickly identify them. Another frontier for picking is represented by special glasses worn by the operator reporting on the lenses all the information he needs.

2.1.1. Barcode handheld system

The barcode handheld system is one of the first devices adopted to facilitate the picking process and also one of the most widespread. All the stock keeping units or also just the stock locations are tagged with a barcode that is scanned by the operator during the picking of the SKU corresponding to the items on the picking list. In this way, the picking information is immediately communicated to the warehouse information system. Handhelds are often able to emit acoustic signals, too. This feature generally helps the user to understand whether the scanner has correctly read the barcode, but it can also be used to provide notification that the product scanned is exactly what the picker was expected to take. Such a system can be combined with paper picking lists, but picking lists can also be integrated directly into the handheld. Once an item has been picked the screen of the handheld shows the next product to be taken. This method requires low investment costs therefore easily applicable, but the research time, picking confirmation time and time to obtain information are among the highest compared to the other picking systems.

2.1.2. Pick-to-light system

In a pick-to-light system operators are guided by lights that are installed on the warehouse shelving. Each stock location has one light that turns on if the operator has to pick a corresponding product from that location and a display that shows the quantity of parts to be taken. In order to complete every single pick, the picker has to press the button of the relevant stock location. If more than one picker in the same warehouse area needs to work simultaneously, such system has to be integrated with paper picking list so that every picker can understand which lights are turned on for his or her order. The visual identification of the storage location and the display for the quantity of parts lead to a reduction of dead-times and therefore to a higher picking performance. Disadvantages are the high investment costs, the need for a superior management system and a high organizational effort.

2.2. Mobile racks system in robot environment

The specific parts-to-picker system treated in this work is based on quite simple mobile robots, which are able to lift a rack and transport them directly to a stationary picker. Alternative descriptions of this system, which is known as the Kiva system.

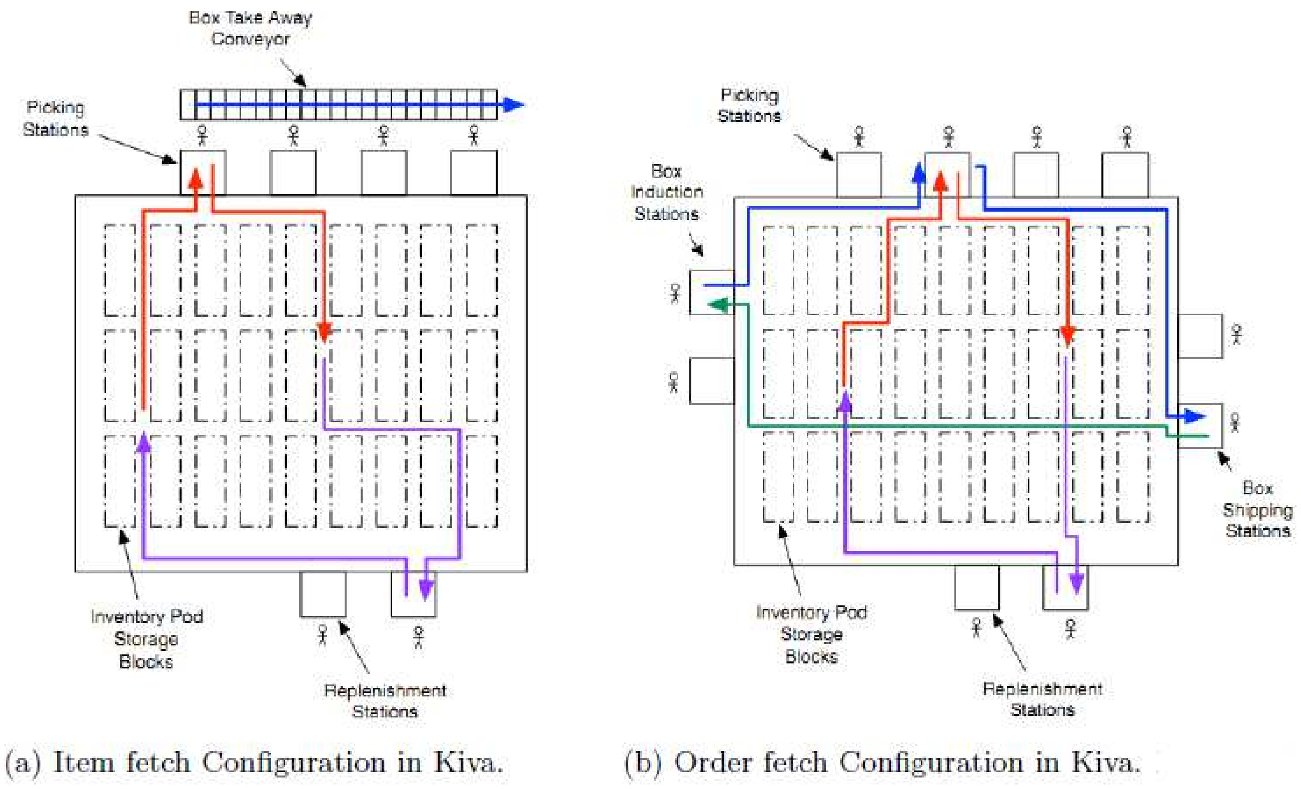
2.2.1. Kiva system configuration

Kiva system can be configured in two different ways as shown in fig. 2.2.:

- Item fetch configuration: in this configuration Kiva robots are responsible for carrying inventory pods from storage area to the picking station. The packed order is carried by conveyor for the shipment. Robots also carry the pod to the replenishment station.

- Order fetch configuration: in this configuration robots have additional responsibility to carry the order pods for the shipment. The human operator picks the order from inventory shelves and put it to another order pod pointed by laser put to light. After all the order in a pod are fulfilled the robot transports the pod for the shipment.

Fig. 2.2. - Two different configuration for Kiva system. Red line indicates robot carrying inventory pod from storage area t the picking station and back to the picking area. Violet indicates robot carrying pod for replenishment. Blue line indicates order pod being carried from induction station to picking station and finally to the shipping station. Green line indicates robot again carries empty order pod from shipping station to induction stations.



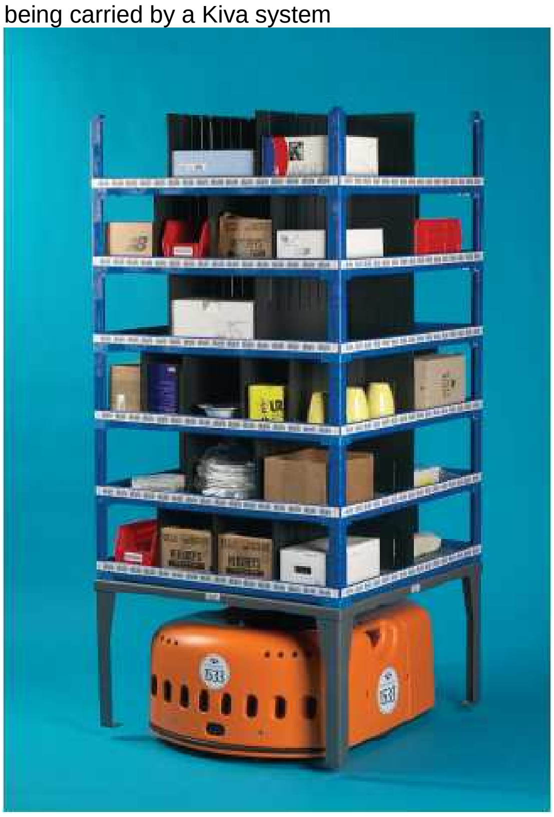
(a) Item fetch Configuration in Kiva.   (b) Order fetch Configuration in Kiva.

2.2.2. Kiva system resources

Kiva system consist of several resources which are useful to accomplish the goal of order picking.

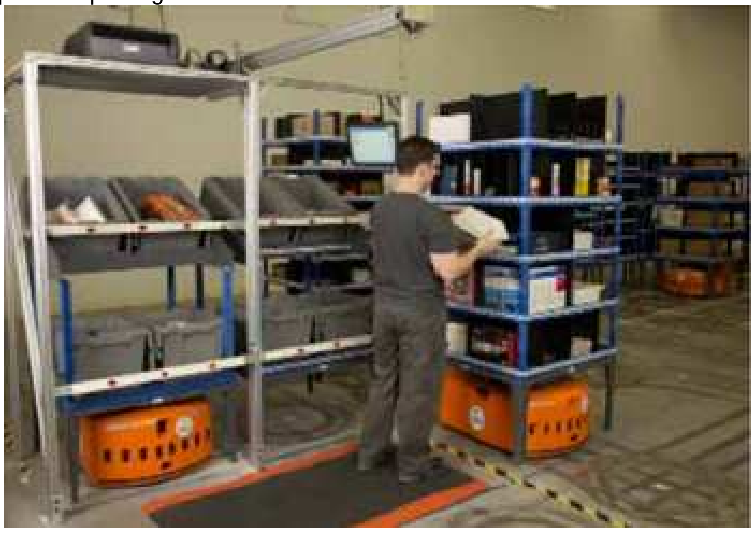
- Inventory: inventories are the products available in the warehouse. Each product is described by its dimensions, packing quantity and its velocity. All the items are assigned unique barcode.
- Pods (Fig. 2.3.): pods are the logically storage locations for the inventories in the warehouse. One robot can carry a pod at a time but it can be scheduled to visit many picking stations. Pods can consist of hundreds of bins, that are filled by human at replenishment station and emptied at the pick station. In between these two process they are stored in the parking areas of the warehouse. A bin does not have to be filled with the same inventory that it previously contained. Pods come in two standard sizes with the most commonly deployed unit size at 1x1 and 1,8 of height. The larger pod is 1,2x1,2x1,8. In general smaller pods are used in application where the weight requirement is up 500 kilograms and the larger pod used in applications where requirements are up 1500 kilograms. The pods are designed to prevent goods from falling off of the shelf levels in transit.

Fig. 2.3. - An inventory pod being carried by a Kiva system



- Parking spaces: there are the spaces where the inventory pods are kept. Every pod is assigned a space park. Once a pod is removed from a space, that space is available for any other pod to be stored in it. Then the space is not fixed and the position of the pods could change every time, typically if the pod location changes the pod remains in the same area of its frequency class (A, B or C). If the frequency, in which an article is requested, does not change then the pod may also maintain its position.
- Stations (Fig. 2.4.): The picking and the replenishment station are placed in the perimeter of the warehouse. Each inventory has unique bar code and the pods are facilitated with pick to light and put to light system. When item is picked, the operator scans the item's barcode, then the place for the item to place for the fulfilment of the order is again indicated by the laser light. This innovation station-based pick-to-light and scanning increases the accuracy of the system. The robots buffer in a queue in the station for the picking. Operators interacts with the pods in the station. The time of interaction depend upon the number and type of the item to be picked. Work station are the ergonomically designed. Stations can be strictly used for inbound restocking or for outbound picking. For flexibility reasons, the stations can be configured to support both.

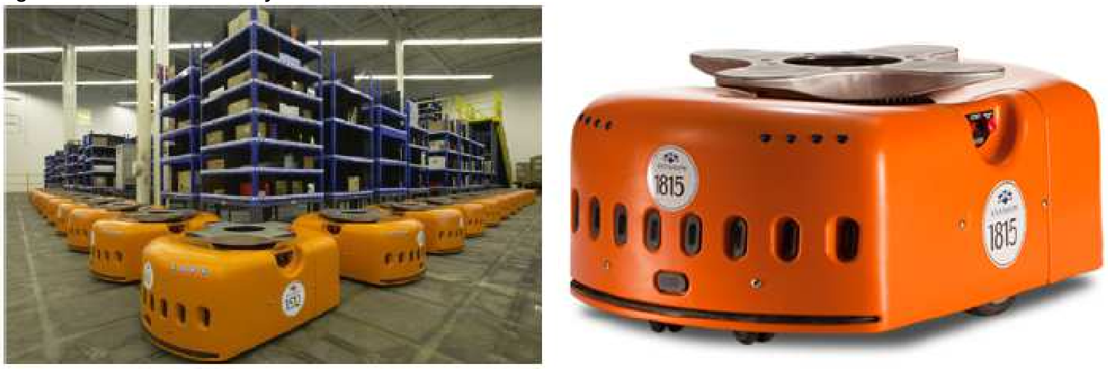
Fig. 2.4. - An sample Kiva picking station



- Robot (Fig. 2.5.): robots are shared and scheduled resources of the Kiva system. They are used to carry the inventory pods and order pods. The robot performs the task in the following steps:
  1. Robot receives the massage from central server to pick the specific pod from the picking area;
  2. Robot move from its current location to the pod's location;
  3. Robot lift the pod and carry it to the specified picking or shipping station;
  4. If there are other robots then robots stay in queue until they have their turn to pick the items from the human operator;
  5. After the operator picks the items the robot move gets back to some parking space and keep the pods back

There are two basic models that handle different weight capacities. The most commonly deployed robot handles up to 500 kilograms. The more expensive heavy-duty models handles up to 1500 kilograms of weight. The smaller robots measure about 0,6x0,75 and 0,3 of height and weight about 100 kilograms. They are equipped with a corkscrew type lifting mechanism to elevate the pods off the floor prior to transport. The lifting mechanism is in effect a custom-built ball screw powered by a single DC drive motor. To keep the pod motionless in transit, the robot rotates its wheels in the opposite direction and at the exact speed. The robots travel at a speed of about 1 meter per second, which is similar to walking speed. The robots run on rechargeable lead-acid batteries that are charged at frequent intervals throughout the day such that there is no battery change-out process required. The robots simply travel to designated charge station every couple of hours where they receive a 5-minute battery recharge before returning to production. The robots travel around a street and highway grid that is mapped out with 2D barcode that are placed every meter along the street grid. The robot is equipped with a camera that looks upward and one camera that look downward. The cameras detect the stickers which enables the central computer to know where the robot is within the grid. Every robot is equipped with sensors that detect if there is an obstacle in the way which if encountered prevents the robot from moving forward. This is to ensure that the robot does not hit an operator or a product that may have fallen off of a pod. The work area where the robots travel is not intended to be accessible by warehouse associates in any way so the sensors are really intended to ensure that the robots do not crash into each other. One of the hidden benefits of Kiva´s robots is that they can work in the dark.

Fig. 2.5. - Robot in Kiva system



2.2.3. Kiva software

The Kiva software is integrated with the client's enterprise system, and typically the primary interface point is a warehouse management system. Kiva system software is a multi-agent system and can be categorized into three major agents Job Manager (JM), Drive Unit and inventory system.

Fig. 2.6. - Logical system relationships

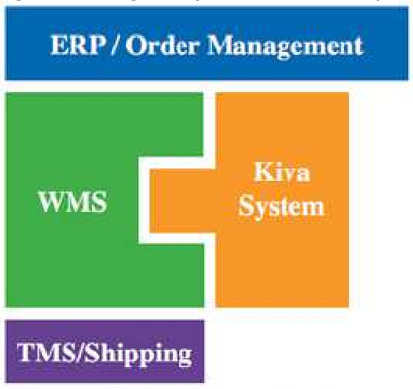
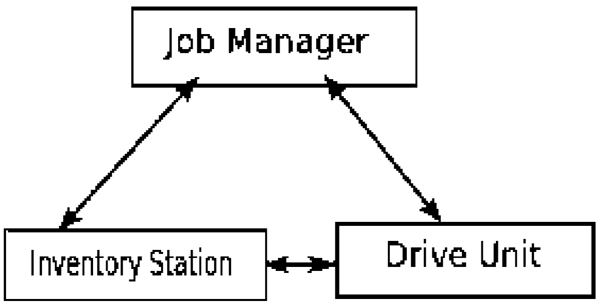


Fig. 2.7. - Multi-agent architecture of Kiva system



Job Manager (JM)

JM is supposed to be the core agent. The responsibility of JM is to receive the customer orders that need to be fulfilled in real time. After receiving the order

JM assigns drive, pods and stations to carry out the tasks. It also keeps the information of warehouse management system. The prime responsibility of JM is the resource allocation.

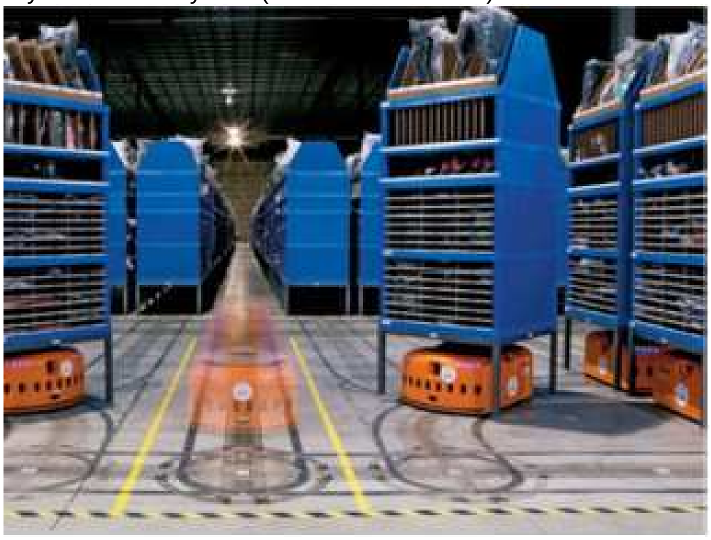
The main objective of the resource allocation is to fulfil large number of orders within a short time using less number of robot and inventory pods. The constraint for good resource allocation is to keep the pickers at the station as busy as possible and use less robots and pods. It is difficult to have optimized resource allocation in Kiva due to the dynamic nature of the system because orders are quite large, interaction of vehicles is dynamic and human interaction response time with the system is unpredictable.

Drive Unit

Drive unit agents are the mobile autonomous robots in the Kiva system. The job of the mobile robot is to pick up the pods from the storage area. Then the robot carries pods from the storage to the picking station. The drive unit is also responsible for optimal path planning, motion planning and obstacle detection. The robot are equipped with sensor on back and front for the obstacle detection. The floor of warehouse are placed with fiducial markers. The robots' navigation system involves a combination of dead reckoning[2] and cameras that look for these markers. When pod has to reach from storage to station then algorithm is used for the path finding.

[2] Dead reckoning is the method of calculating robot's current position by using previously determined position and advancing that position based upon the estimated speeds over elapsed time.

Fig. 2.8. - Navigation system of Kiva system (with fiducial markers)



Inventory station agent

It is the human being at the picking station. Laser pointers identifies the proper inventory on the shelving unit to pick up by the human being. The operators pick the inventory and put it into the outgoing shipping cartons. The workers do not require to walk around to search for the product like in traditional warehouse.

3. Assumption and parameters

In this chapter all the parameters assumed in the simulation are listed. Then for the parameters that require an explanation for their choice, they will be discussed in the following paragraphs.

3.1. General assumption for all systems

Parameters used on all simulated picking systems are:

- Number of picking station: there is 1 pick station. The pick station may have single or multiple orders open simultaneously. The replenishment of items into the shelves is not taken into consideration.
- Number and type of shelves: there are 192 racks/mobile racks. The shelves have standard dimensions. Each shelf measures 1 meter in width, 1 meter in depth and 1,8 meters in height. Each shelf has three trays. Every shelves have an usable volume by goods of 0,8 cubic meters.
- Number and type of SKUs: there are 650 SKUs. They are divided into three categories based on their frequency. Each item has its own size (small, medium or large) in order to fill the shelves keeping in mind their maximum capacity.
- Number and type of daily orders: as mentioned in the introduction, the randomly created orders, equal for each system, are meant to be online food retailing orders. Then with a large number of order lines per order. There are 150 customer orders and a total of 4500 order lines.
- Picking time and other time components.

3.1.1. Number and dimensions of shelves

To have a realistic comparison, the number of shelves that make up the warehouse must be adequate. For picker-to-parts picking systems this is not a problem because they have a low investment cost and so can be implemented on warehouses of all sizes, obviously with a higher or lower number of operators. Instead for the parts-to-picker system the number of shelves must be higher than the minimum to allow the systems to function optimally, because the investment costs are high and therefore it is advantageous to use it in a warehouse where, in the case of picker-to-parts, the required work force is high. So as to break down the number of human operators.

In order to find the right size of the warehouse to be simulated, a research was carried out in the literature. Going to look for scientific articles in which practical examples of kiva system have been reported. In this work, as written above, there will be only one picking station, so all the data found were divided by the number of station to which they were associated (tab 3.1). Finding, in this way, the number of shelves to associate with a station.

Tab. 3.1. – Research number of shelves

Scientific article	Stored shelves	Number of pick station	Stored shelves associated to one pick station
“Assignment rules in robotic mobile fulfilment system of online retailers“ Bipan Zou, Yeming Gong, Xianhao Xu and Zhe Yuan, 2017	800	4	200
“Simulation based performance analysis in robotic mobile fulfilmet system“ Thomas Liernert, Tobias Staab, Christopher Ludwig and Johannes Fottner, 2017	360	4	90
“Path planning for robotic mobile fulfilment system“ Marius Merschformann, Lin Xie and Daniel Erdmann, 2017	795	3	265
	550	4	138
	1951	16	122
	2726	16	171
“Decision rules for robotic mobile fulfilment system“ M.Merschformann, T.Lamballais, M.B.M. de Koster, L. Suhl, 2018	1150	6	192
“Estimating performance in a robot mobile fulfilment system“ T. lamballais, D. Roy and M.B.M. de Koster, 2015	1500	5	300
“A simulation-based comparison of wo goods-to-person order picking systems in an online retail setting“ Yavuz A. Bozer and Francisco J. Aldarondo, 2018	924	4	231
Average number of stored shelves associated to one pick station:			190

The number of shelves must be around 190, considering the quantity of items to be stored (subsequently discussed) and the arrangement of the shelves, 192 shelves were used. The dimension of the shelves are actual dimension used by the Kiva method found in the catalog, 1 meter in width, 1 meter in depth and 1,8 meters in height. From a total volume of 1,8 cubic meters the usable one is 0,8 cubic meters, because it has been removed:

- The space lower than first trays in order to allow the passage of the robot Kiva system, with a height of 0,3 meters;
- The part occupied by the three trays, with a total height of 0,1 meters;
- And the rest for the space for gaps between items and to allow an easy grip to the operator.

3.1.2. Number and type of SKUs

To calculated the number of SKUs, it was considered that it had to be suitable for the size of the warehouse and the previous simulation and examples reported in the literature. From the research results a number of SKUs equal to see Tab 3.2.

Tab. 3.2. – Research number of SKUs and total order lines.

Scientific article	Number of SKUs	Number of total order lines
“Dual-tray Vertical Lift Module for order picking :a performance and storage assignment preliminary study” Battini Daria, Calzavara Martina, Persona Alessandro, Roncari Manuel and Sgarbossa Fabio	1200	10000
“Using simulation to analyse picker blocking in manual order picking systems” Behnam Bahrami, El-Houssaine Aghezzaf and Veronique Limere	300	3000
“A comparison of picking, storage and routing policies in manual order picking” Charles G.Peterson and Gerald Aase	1000	10000
“Determining the number of zone in a pick and sort order picking system” Rene´B.M. de Koster, Tho Le-Duc and Nima Zaerpour	1000	2500
“An approach to order picking optimization in warehouse” Matic Horvat	100	2500
“Parts-to-picker based order processing in a rack-moving mobile robots environment” Nils Boysen, Dirk Briskorn and Simon Emde	1000	500
Average Values:	766	4750

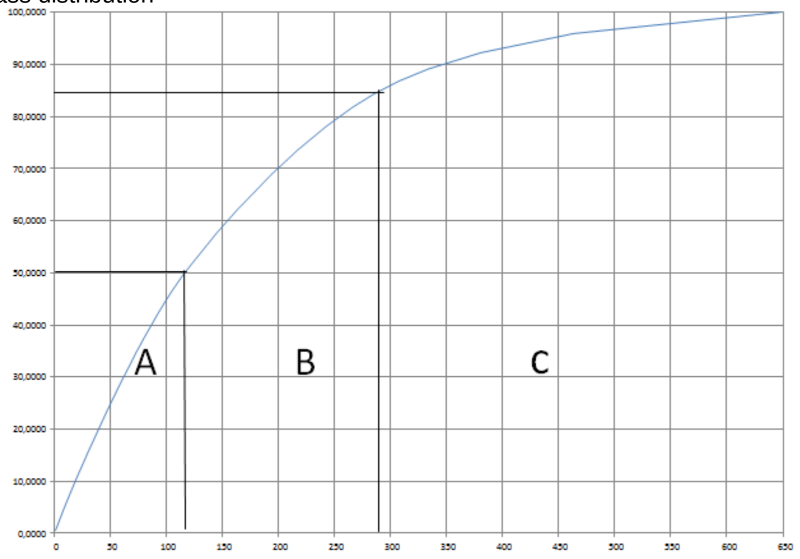
Considering the value obtained, wanting to simplify the simulation to reduce the calculation times, I decided to round down the number of SKUs to 650.

The articles were created using the Excel program. Each article has:

- an associated dimension size, so as to determine, depending on the article, how many items can be stored on each shelf.
- its own level of request so as determine three categories based on the speed that they must have. To share the article into the classes was used the Pareto method, in which a small part of articles holds a large part of the request (Chart 3.3). The class A with 20% of items moves

50% of the picks, for B class 30% of items moving to further 35%. And the last 15% of picking line for the C class made up of 50% of the articles.

Chart 3.3. - Class distribution



To calculated the quantity to be stored in the picking warehouse was considered a stock of 6 working days and then the request of the articles was multiplied by the days of stock, thus finding the total number of items in the warehouse, equal to 72844, and the volume that they occupy, equal to 152.16 cubic meters. From the total volume of the items stored and knowing the capacity of each shelves, the number of shelves to be used can be calculated, equal to 192, number previously reported. I assume that there will be not stock out.

3.1.3. Number and type daily orders

As already mentioned, a batch of 150 customer order is generated randomly for each day, for a period of six months. All systems are simulated with the same set of daily order. The orders are meant to be online food retailing orders, so each order may contain a large number of items from 10 to 60 items, with a total of 4500 order lines. Most orders contain a large number of items. In one order for each picked item, the quantity is always one unit. Because in this way there is the most critical situation, that is the travel time, in the case of picker-to-parts systems, and most probably the number of mobile shelves that must be taken, in the case of parts-to-picker, will increase.

3.1.4. Picking time and other time components

The following table (Tab 3.4) indicates the times for the different activities involved in the simulation for each type of system. These times are fixed for each picked item.

Tab. 3.4. - Time components

	Barcode system	Pick to light system	Kiva system
Picking time [s]	4.87	2.86	4
Packing time [s]	2	2	2
Search time [s]	7.96	0	0
Get information time [s]	2.98	4.85	0
Confirm time [s]	4.02	0.98	0

3.2. Assumption of the picker-to-parts systems

Parameters used on picker-to-parts systems are:

- Forward area: There are 12 aisles and 2 cross-aisles. The aisles and crossings have a width of 2 meters in order to allow the shelves to be filled even with the help of vehicles. The shelves in the storage area are divided according to the class they belong to. There are 70 pods of class A, 52 of class B and the remaining 70 of class C. The class positions vary if is used a return or traversal routing policy.

Return policy (Fig. 3.1) : where an order picker enters and leaves each aisle from the same end. Only aisles with picks are visited. Class A items are on the shelves near the entrance to each aisles, followed by those in class B. Class C items are on the shelves at the end of the aisles. An example shows in Figure 3.2.

Fig 3.1 - Left: forward area with return routing; right its top view

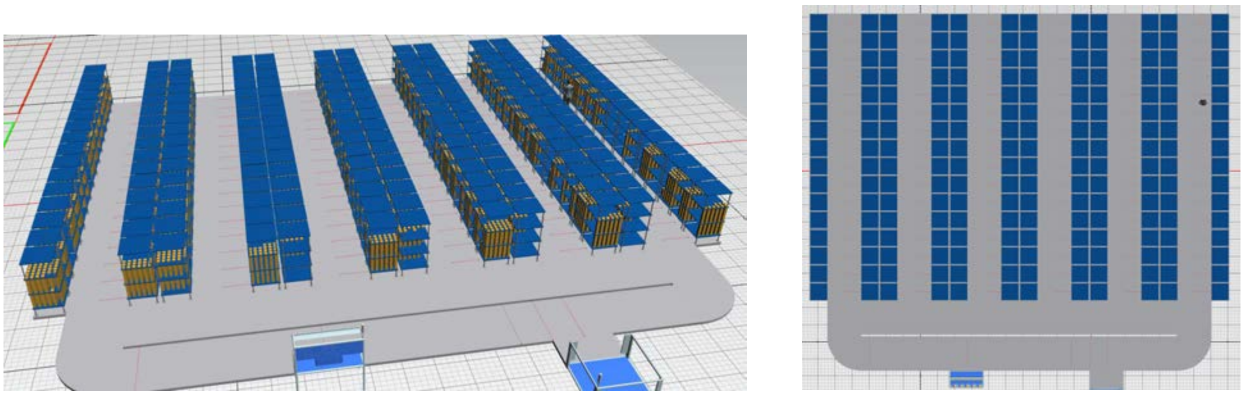
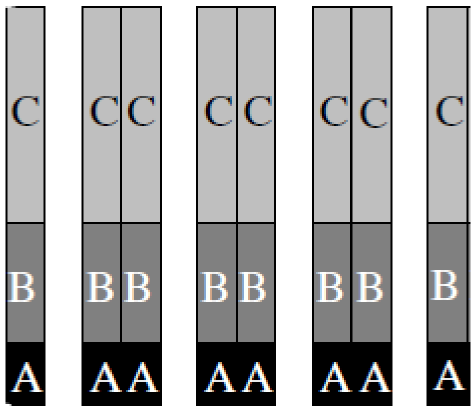


Fig 3.2 - Example of class distribution in warehouse with return routing.



Traversal policy (Fig. 3.3): routing order pickers by using the S-shape method means that any aisle containing at least one pick is traversed entirely. Class A items are on the shelves of the first aisles that can be visited, followed by those in class B. Class C items are on the shelves of the aisles that can be visited. Aisles without picks are not entered. From the last visited aisle, the order picker returns to the depot. An example shows in Figure 3.4.

Fig 3.3 - Left: forward area with traversal routing; right its top view

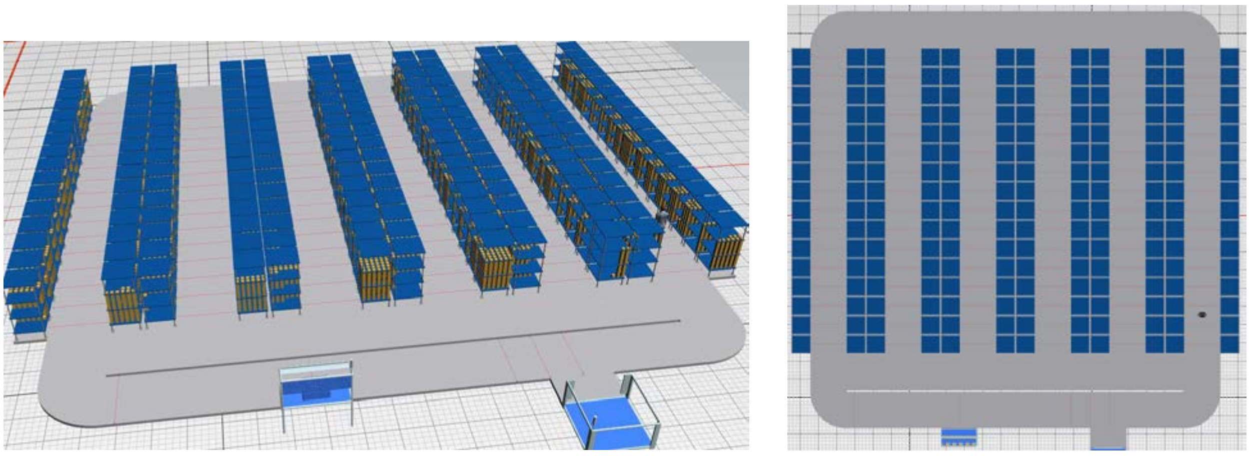
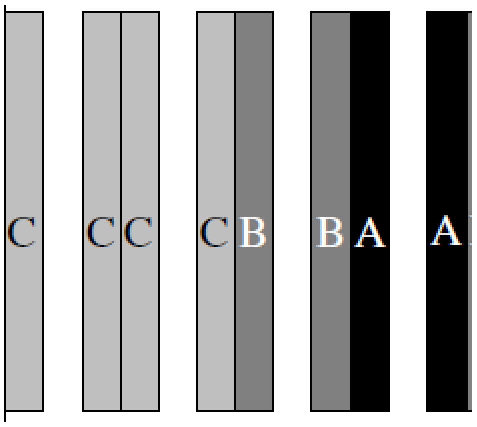


Fig 3.4 - Example of class distribution in warehouse with traversal routing.



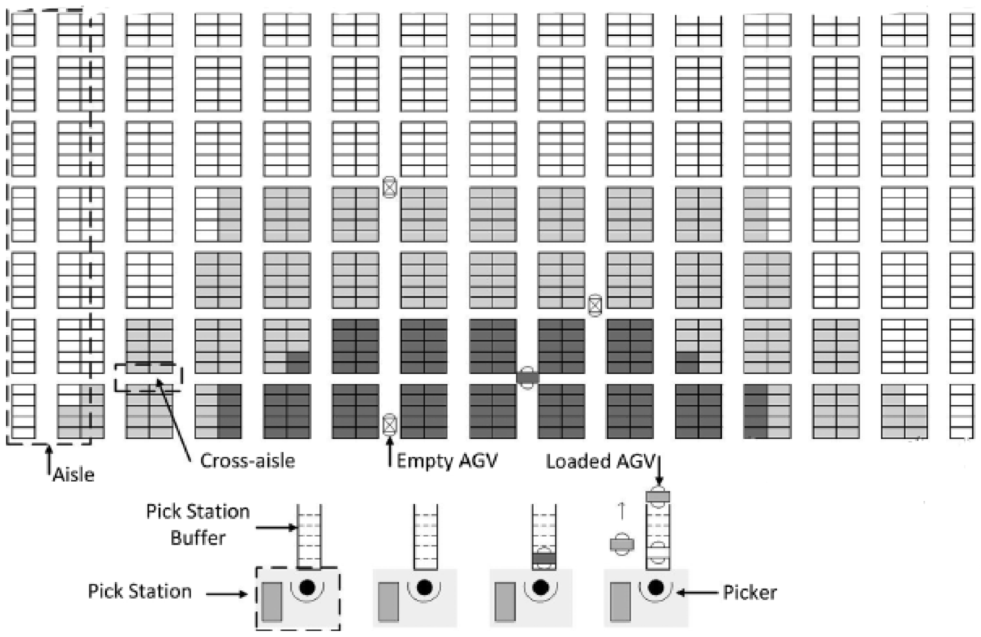
- Velocity of picker: his speed is 1 meter per second.

3.3 Assumption of the parts-to-picker system

Parameters used on parts-to-picker system are:

- Forward area: The pods in the storage area are divided according to the class they belong to. In the nearest area the pods containing articles of class A, then those with class B and C. An example is shown in figure 3.5.

Fig. 3.5. - Example of class distribution in Kiva storage area



There are 70 pods of class A, 52 of class B and the remaining 70 of class C. There are 12 aisles of 1,10 meter width, enough width to allow the passage of a robot with shelf. In fact, the operators do not have to enter in this area to recharge the pods, this operation is done in other stations. An advantageous feature that allow to increase the percentage of occupied warehouse space. To minimize congestion, cross aisles are placed every four pods and between the picking station and storage area there is a free zone of 3 meters (Fig 3.6 and Fig. 3.7).

Fig. 3.6. - Forward area of Kiva system by PlantSimulation14

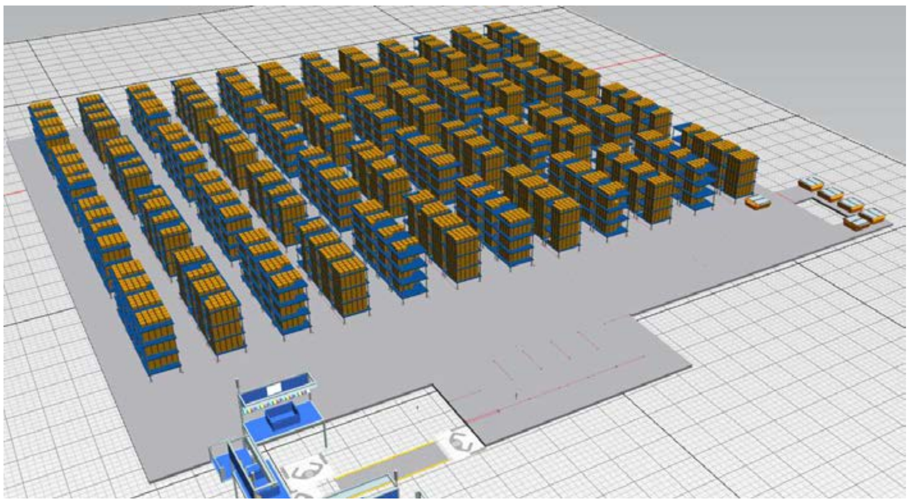
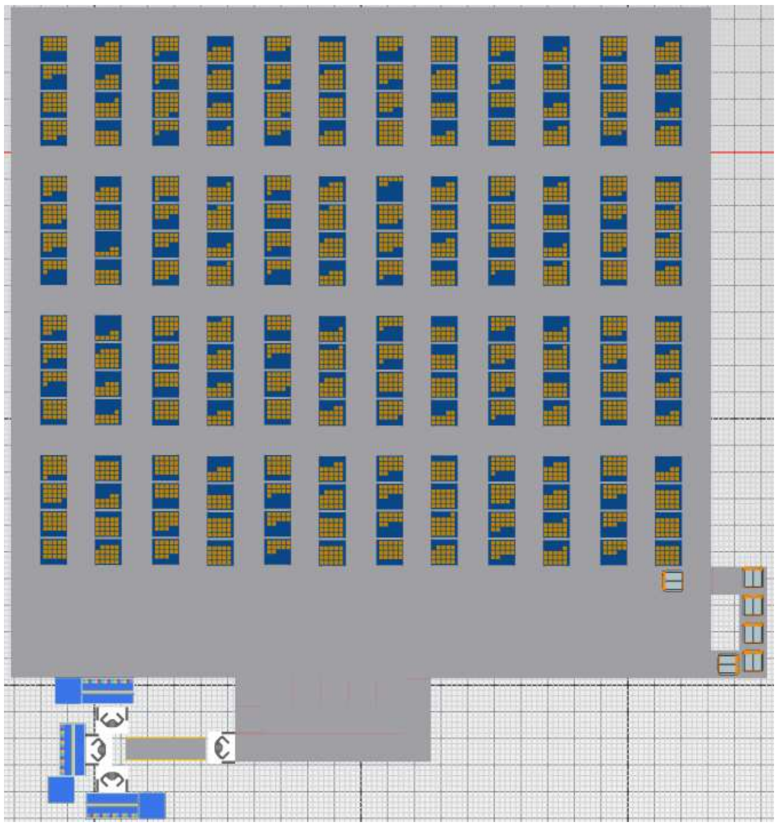


Fig 3.7 - Top view of forward area of Kiva system by PlantSimulation14



- AGVs: the simulations are performed with a number of AGVs from 3 to 6, to determine how the number of robots change the picker utilization. The size of each AGV is 0,6 in width, 0,8 in length and 0,3 in height and with a speed of 1 meter per second. The collision between AGVs is not allowed, each section of route is created so that only one robot can be present at a time above it. To travel between two points, an AGV follows the shortest path. At the start of the simulation, all the AGVs are idle and located immediately outside the front of the forward area.
  The downtime or battery charging and the effects of acceleration or deceleration are not considered.
- AGV control logic: Each AGV follows a dual command cycle; when picking is completed, the AGV travels from the picking station to the forward area to store the pod in its location, then it travels to the next pod to be retrieved and subsequently returns to the picking station. When

there is a retrieval request the first available AGV is used. If no AGVs are available, the first one to become available is assigned to the request.

4. Overview of Plant Simulation

4.1 What is simulation?

Simulation modelling is an excellent tool for analysing and optimizing dynamic processes. Specifically, when mathematical optimisation of complex systems becomes infeasible, and when conducting experiments within real systems is too expensive, time consuming or dangerous, simulation becomes a powerful tool. The aim of simulation is to support objective decision making by means of dynamic analysis, to enable managers to safety plan their operations, and to save cost.

Simulation aims to achieve results that can be transferred to a real world installation. In addition, simulation defines the preparation, execution and evaluation of carefully directed experiments within a simulation model. As a rule, you will execute a simulation study using the following steps:

- first check out the real-world installation you want to model and gather the data you need for creating your simulation model;
- then abstract this real-world installation and create your simulation model according to the aims of the simulation studies;
- after this, you run experiments. This will produce a number of results;
- the next step will be to interpret the data the simulation runs produce;
- finally, management will use the results as a base for its decision about optimizing the real installation.

Developing the simulation model is a cycling and evolutionary process. It will start out with a first draft of the model and then refine and modify. Plant Simulation is software for integrated, graphic and object-oriented modelling, simulation and animation. Many complex systems may be modelled and displayed in great detail closely resembling reality.

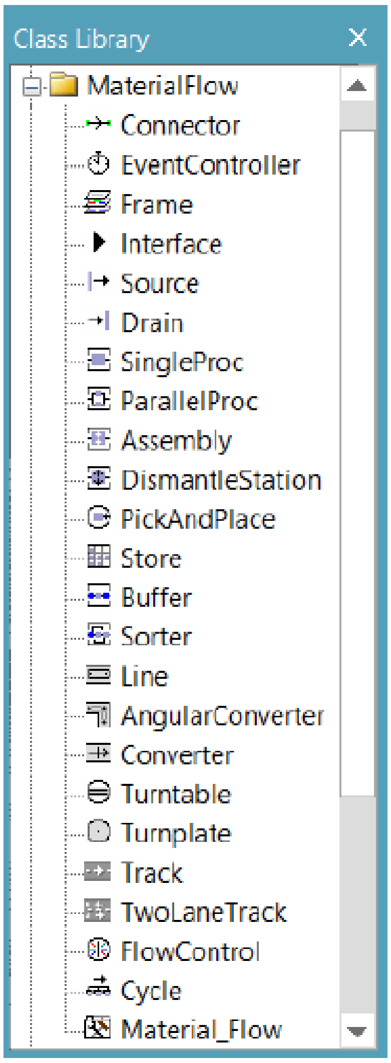
In the next chapter the final models, of the three types of picking systems studied in this paper, will be explained and commented. There will be no intermediate versions or steps to get to the final structure. In the following pages are listed and defined the most common objects of Plant Simulation, also used to create models.

4.2 Basic objects of Plant Simulation

Plant Simulation provides a set of basic objects, grouped in different folders in the Class Library. Now present the most commonly used basic objects from the standard library.

4.2.1 Material Flow

The Material Flow folder contains objects to serve for transporting or processing mobile/moving unit (MUs) objects within models and storing parts and displaying tracks on which parts are moved. The most important objects in this folder will be briefly presented.



Connector: establishes connections between MaterialFlow object, such that MUs can move through the model. An arrow in the middle of the connector indicates the direction. A single connection can only point in one direction.

EventController: PlantSimulation is a discrete event simulation, so the program only inspects those points in time, where events take place within the simulation model. The EventController manages and synchronises these event.

Frame: serves for grouping objects and to build hierarchically structure models. Each new model starts with a Frame where the EventController is placed on.

Interface: represents entry and exit interfaces on a Frame. It is used to connect multiple Frames with each other, such that MUs can flow through them.

Source: creates MUs and attempts to pass them on. It is used at places where a MU is created/generated (usually at the start of a process). The time between the consecutive creations of MUs can be specified by a random variable.

Drain: destroys MUs after processing them. It is used at places where MUs should leave the system.

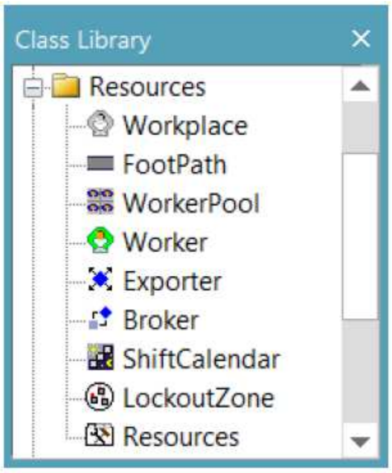
SingleProc: receives a MUs, retains it during the processing time and then attempts to pass it on. For example, a machine with capacity 1.

ParallelProc: receives a MU, retains it during the processing time and then attempts to pass it on. Several MUs may be processed at the same time. Processing times may differ and MUs may pass each other. For example, a machine with capacity>1.

Store: receives passive MUs. A MUs remains in the Store until it is removed by a user control. It can be used for a store shelving system.

Buffer: receives a MUs, retains it during a given dwell time, and then attempts to pass it on. When the preceding stations are unavailable (occupied or failure), the MU stays in the Buffer. MUs can exit the Buffer in the same order in which they entered it (FIFO) or in the opposite direction (LIFO).

4.2.2. Resources



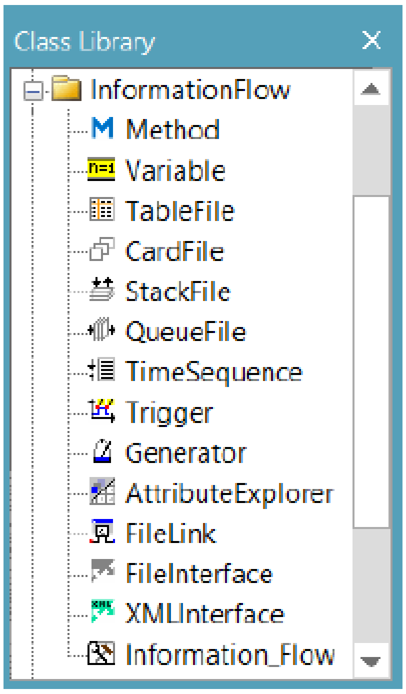
The resource objects serve for adding human workers to a processing station and let workers move on paths between workstations related to production stations.

Workplace: creates a position where a worker can stop and complete an activity. It must be placed next to another object in the material flow folder.

FootPath: is the path that can be followed by the operator to move from one workstation to another. It is walkable in both directions.

WorkerPool: the Workers are created in the WorkerPool and they stay there when they do not work and are waiting for an order.

4.2.3. Information Flow



The InformationFlow objects serve for exchange of information between objects. List are provided to record large amounts of data, to store them and to make them available during simulation. They provide the functionality of a database in a real world installation. Plant Simulation provides StackFile, QueueFile, CardFile and TableFile. These lists differ in their dimension and the Methods provided for accessing them.

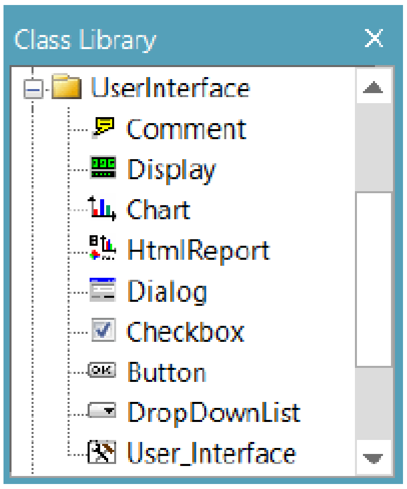
Method: enables the modeller to program custom logic into the model, using the programming language SimTalk 2.0.

Variable: is a global variable that may be accessed by all objects.

TableFile: is one of the most important information flow objects in Plant Simulation. It serves as a two-dimensional data container. Its elements may be randomly accessed. In addition, a number of search and read functions is available.

Generator: allows to call method at predefined times during the simulation.

4.2.4. User Interface



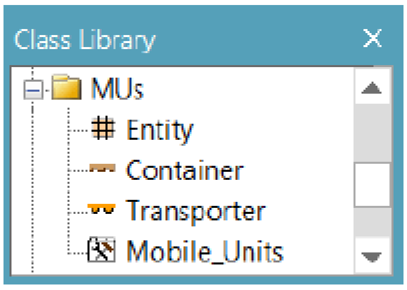
User interface objects facilitate the interaction between the user and a model, for example, Dialog for model input and Charts and Reports for model output .

Comment: enables to add additional descriptions and notes to the model.

Display: displays values during a simulation run. Values can be displayed in string form or as bars.

Chart: can be used to visualise the data generated by a model.

4.2.5 Mobile Units



These classes can represent every kind of product, pallet, container or vehicle that moves through a logistic system.

Entity: this is the object or Moving Unit (MU) that gets moved around in a simulation model. It can represent anything that must pass different stations for processing orders.

Container: similar to the Entity, this is a mobile object during the simulation. It has a loading space that may contain MUs. It represent any kind of container, pallets and boxes.

Transporter: similar to the Container, but the Transporter is self-propelled and its speed is user-defined. It represent any kind of Transporter, AGVs and forklifts.

5. Simulation models

In this chapter the models and algorithms used in each system are explained. The simulation models were developed with PlantSimulation14, where model development starts with adding elements to the physical layout and them identifying the entity that flows through the layout. For the picker-to-parts systems, the element in the layout are: forward area, routing path and the pick station. For the Kiva system, the element in the layout are: forward area, path grid, pick station input buffers and the pick station. The customer orders were created outside of PlantSimulation14 as an excel file. An extensive number of runs, equal to six months, were made to validate the models.

The simulations of the picker-to-parts systems are divided into 4 parts:

- Creation of stored shelves and filling with associated items;
- Choice of the next shelf to be visited and pick up items;
- Placing orders in the station;
- Data collection.

The first part is executed before the simulation, the next two parts during the simulation and the last one executed after the simulation.

The simulation of the parts-to-picker system is divided in 5 parts:

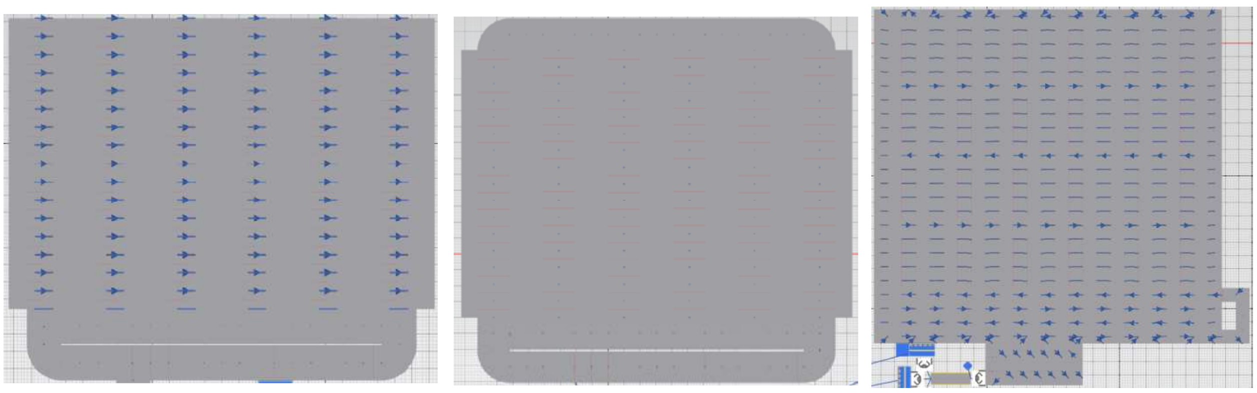
- Creation of stored shelves and filling with associated items;
- Choice of the next shelf to be transported;
- Identification of Kiva robot that will transport the next shelf and pick up items;
- Placing orders in the station;
- Data collection.

The first two parts are executed before the simulation, the next two are composed of algorithms that work during the simulation and the last one executed after the simulation.

5.1. General parts for all systems

Before writing the scripts to create the elements mentioned above, the base was created. The base includes the space for the position of the shelves and for the tracks. So this base will be different for each model especially for the track that will be created as described in the previous chapter.

Fig 5.1 – In order from left to right: the base for picker-to-parts system with return routing, the base for picker-to-parts system with traversal routing and the base for parts-to-picker system.



Below will be explained all common parts of the simulation.

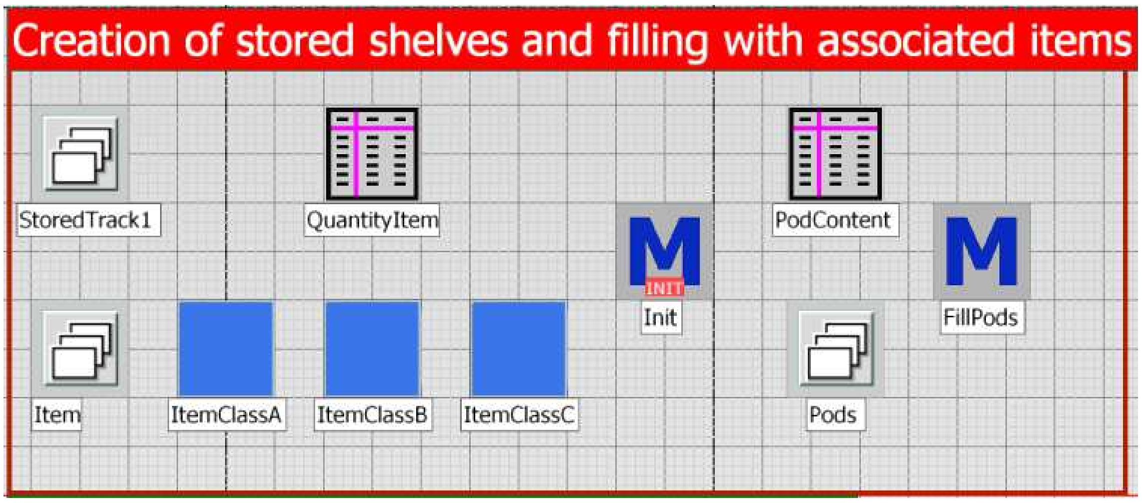
5.1.1. Creation of stored shelves and filling with associated items

The aims of this parts of the model are:

- To create the shelves in specific positions, different for each system configuration;
- To create all the items to be stored;
- To fill the shelves with the associated items, determined by a matrix created on excel that considers the capacity of each shelf.

Now we see in detail which commands have been used to achieve the aims listed above. The commands used for this part are shown below (Fig. 5.2):

Fig 5.2 – Screenshot: commands to create shelves and fill them with associated articles



CardFile object (), it is a single column list in which each row corresponds to an element (object, string, number, etc.). The elements used of this object are:

- StoredTrack: for each row a piece of track (object) will be stored. The order of the list must respect the class of the shelves, that is first the positions that contain the shelves of class A then those of B and C. So it is formed by 192 lines.
- Item: For each row there is an article, listed by class. It is formed by 650 lines.
- Pods: list that contains all the shelves, consisting of 192 lines.

TableFile object (), it is a table with variable number of rows and columns. If the contents of the boxes are numbers, it can simulate a matrix. The elements used of this object are:

- QuantityItem: it is a table in which in the first column are listed all the articles and in the second column there is the number of copies of that article which must be stored in the warehouse. It needs to know how many articles to create. Matrix size 650x1.
- PodContents: the x-axis formed from the list of shelves and the y-axis from the list of articles. Each crossing indicates the number of pieces of an article n (row) on shelf i (column). It needs to know which and how many items are on the shelves. Matrix size is 192x650.

Buffer object (🟦), it is a container that can accommodate entities, in this case the articles before being moved to their associated shelves. To reduce the calculation time, three buffers (ItemClassA, ItemClassB, ItemClassC) were created that will contain the items according to their class.

Method object (M), in this object is possible to write codes, with SimTalk programming languages. It is therefore possible to create controls that are called and started by other objects. The elements that used this object are:

- Init: Initial code to create shelves, articles and AGVs, if used in the Kiva system, and to call a subsequent FillPods method and Choicepod (if used Kiva system). Code shows in the appendix A.1.
- Fillpods: method used to move items from buffers to associated shelves, based on the PodContents matrix. Code shows in the appendix A.2.

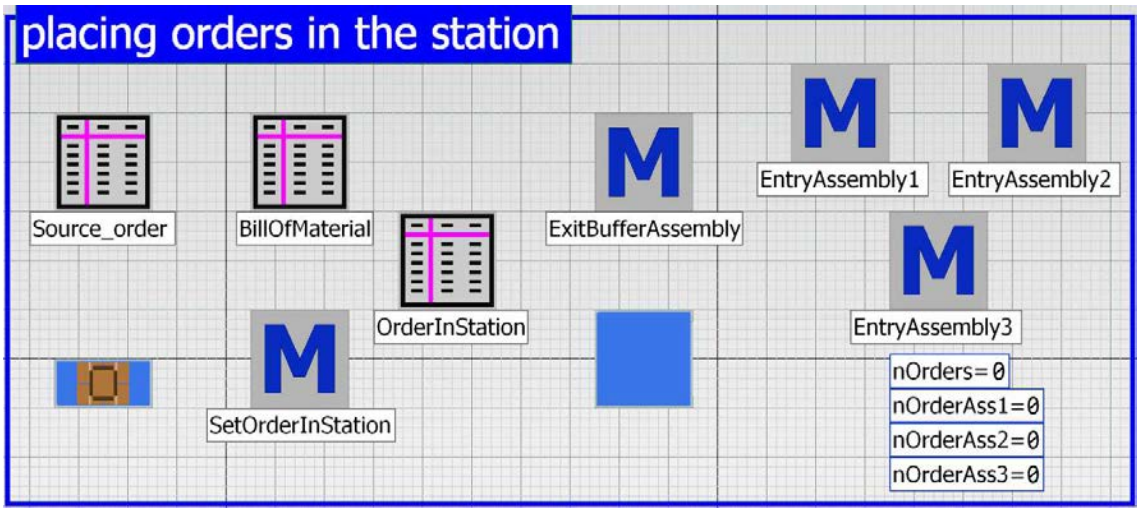
5.1.2. Placing orders in the station

The aims of this parts of the model are:

- Create orders
- Send them to the station in an adequate number respecting the batching policy
- Create the picking list

Now we see in detail which commands have been used to achieve the aims listed above. The commands used for this part are shown below (Fig. 5.3):

Fig 5.3 – Screenshot: commands to create order and send them in the station



Source object (), it is used to create incoming entities, in this case it is used to create orders according to the Source_Order table. This object is connected to the picking station and therefore, depending on the batching policy adopted by system, a number of orders are transferred to the station. When an order is finished, success takes its place, until the orders are completed.

The table objects used in this part of the simulation are:

- Source_Order: list of orders to be created by the Source object. Size of the matrix is 150x1;
- BillOfMaterial: the x-axis formed from the list of orders and the y-axis from the list of articles. Each crossing indicates the number of pieces of an article n (row) in the order i (column). It indicates which items are part of the order. Matrix size is 150x650.
- OrderInStation: the x-axis formed from the list of articles and the y-axis from the list of open order in the station, this depending of batching policy. It indicates which items must be picked up from the station to complete the order.

The method object used in this part of the simulation are:

- SetOrderInStation: method used to assign the pick list to the station, or pick lists if there is a batching policy, according to the BillOfMaterial table. Code shows in the appendix A.3.
- ExitBufferAssembly: method that performs a control function for the simulation. If the operator takes a wrong item, it is sent to a special buffer to contain the errors.
- EntryAssembly1, EntryAssembly2, EntryAssembly3: methods to view the orders that are processed in the station.

5.1.3. Data collection

The aim of this part is to collect the data after the end of the simulation. The data collected are:

- Elapsed time to complete daily order
- Picker throughput calculated in order lines per hour
- Average time for order
- Number of shelves visited to complete an order
- Times and percentages for each activity of the picker (getting information, searching, picking, confirm and travel)
- Picker utilization
- Times and percentages for each activity of the Kiva robot (waiting and working)
- Kiva robot utilization

The commands use for this part are:

- EndSim method: to calculate the results of each simulation day, allow their display on screen and save them in a table and then obtain the average. Code shows in the appendix A.4.
- TimeWorker: it is a table in which are reported the times of permanence for each visited shelf.

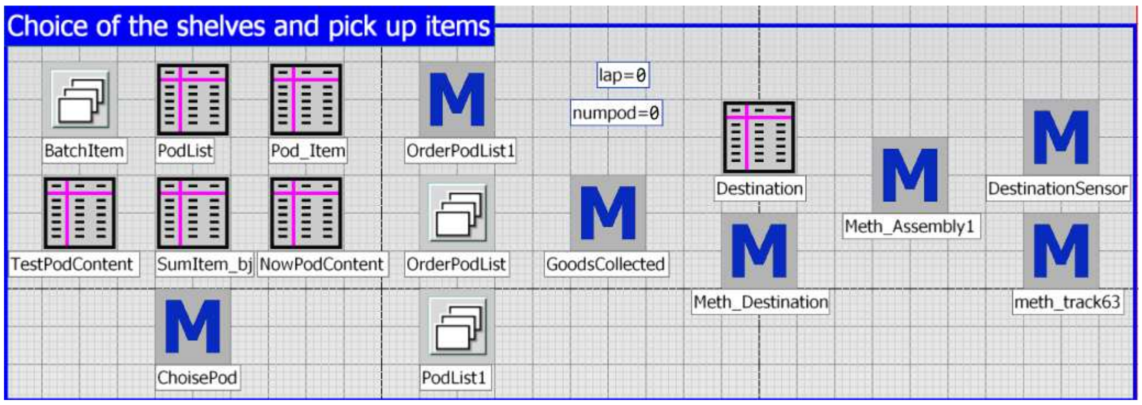
5.2. Choice of the shelves and pick up items in picker-to-parts systems

The aims of this model part are:

- create a general picking list, which includes all the items requested in the orders present in the station;
- determine the shelves to visit to complete the picking list;
- create an ordered picking list considering the path of the operator, that is compared to the shelves that will be visited first, then the items at the top of the list are also the first ones that will be picked up;
- create the path that the operator must perform, with stops on the shelves defined for the completion of the picking list.

Now we see in detail which commands have been used to achieve the aims listed above. The commands used for this part are shown below (Fig. 5.4):

Fig 5.4 – Screenshot: commands to Choice of the shelves and pick up items in picker-to-parts systems



To determine the shelves from which to take the items the method chosen is to minimize the stops, that is the number of shelves to be visited in a lap. To do this for each shelf you determine the maximum number of items that can be taken and then choose the one with the largest number. Once the shelf has been chosen, the items are removed from the list and the procedure is repeated until all the items in the picking list (BatchItem table) are taken.

The table objects used in this part of the simulation are:

- Pod_item: the x-axis is formed from all the shelves in the storage area and the y-axis from all the items in the picking list. In each intersection box indicates if on shelf i (row) there is or not the item n (column). The size of matrix is variable, it depends from the number of items in the picking list.
- SumItem_bj: on the y axis all the available shelves are shown, for each of them the number of items that can be taken is calculated by adding the values of the respective column of the Pod_Item table. The size of matrix is variable, it depends from the number of items in the picking list. This is used to determine the shelf with several removable items and then insert it into the operator's picking path. Once it is defined, the picked items are removed from the list.
- NowPodContent and TestPodContent: they are tables that have the memory function; they memorize, after each withdrawal, the number of quantities of each item still present on the respective shelves. the x-axis is formed from all the shelves in the storage area and the y-axis from all the items. The size of matrix is 192x650.
- Destination: on the y axis are shown all the shelves, next to each shelf there is the grid section in which it is located.
- PodList: complete list of shelves to visit in the next round to complete the order. They are ordered from the shelf with the largest number of items that can be withdrawn to the one with the lowest number, therefore not ordered with respect to the path of the operator.

The CardFile used in this part of the simulation are:

- BatchItem: is a list of all the items needed to complete the order in the station. It is a dynamic list, it is modified every time that the next shelf to be visited changes. Therefore it is a sort of picking list in which the items already picked are deleted.
- OrderPodList: is a static list, it is a complete list of shelves ordered with respect to the path of the operator.

- PodList1: using a method, explained below, the PodList list is reordered according to the path of the operator obtaining a series of shelves ordered from the closest to the furthest in which the operator during his lap will have to stop to pick up at least one item.

The method used in this part of the simulation are:

- ChoicePod: method used to determine the right shelf, that is the one with the most items, considering the list of items requested by the station. the method is activated by the operator at the beginning of his lap and, to find a solution it uses and modifies the tables SumItem_bj, TestPodcontent and NowPodContent. Code shows in the appendix A.5
- OrderPodList: method used to change the order of the shelves chosen to complete the orders in the picking station. Passing from an ordered list based on the number of possible items to be taken to an ordered list that follows the path of the operator, ie the first shelves of the list will also be the first visited. So the method transforms PodList into PodList1.
- GoodsCollected: method that identifies which items the operator must take when he stops in front of a shelf in the list.
- Meth_Destination: method that moves the operator to move from one shelf to another.
- DestinationSensor: method inserted in each sensor in front of each shelf. It starts when the operator stops in front of a shelf. It is used to allow the operator to physically take the items identified by the GoodsCollected method.
- Meth_Assembly: method present in front of the station, it is activated when the operator returns to the station with all the items taken. Here he discharges the items to complete the order and starts another cycle if there are still orders in the queue. Code reported in the appendix A.6.

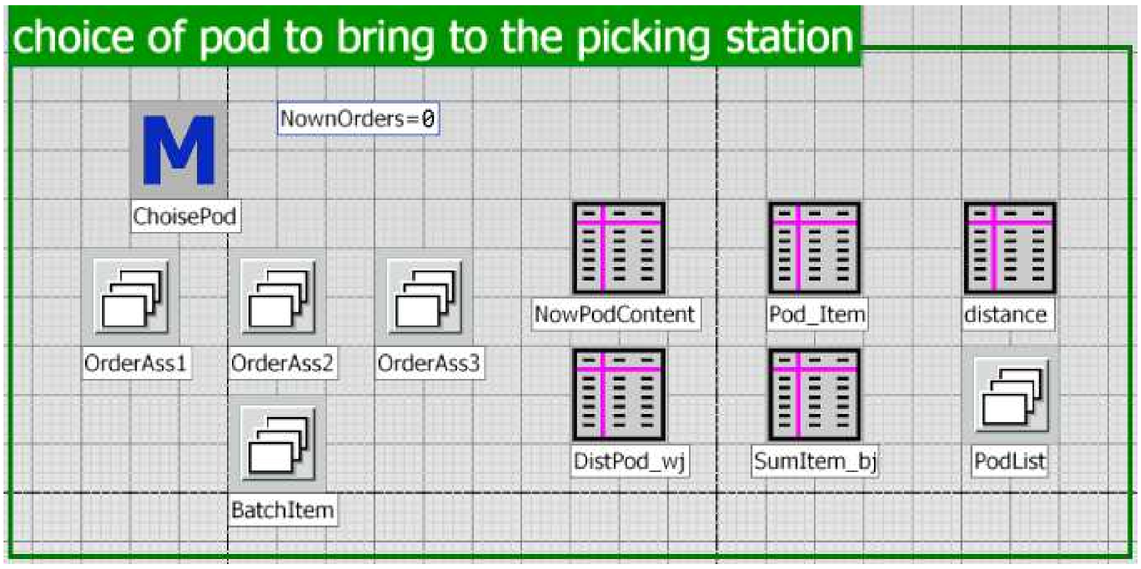
5.3. Choice of the pods to bring to the station in Kiva systems

The aim of this model part is:

- define the most suitable shelves to pick up and take to the picking station.

Below we see, in detail, which commands were used to fulfill the aim. The commands used for this part are shown below (Fig. 5.5):

Fig 5.5 – Screenshot: commands to Choice of the most suitable in Kiva systems



The choice of the shelf to be taken is taken using a heuristic algorithm in order to determine the optimal picking list that minimizes travel times and the number of trips to be performed at the same time. The algorithm used is reported by Li Z., Zhang J., Zhang H. and Hua G. in the scientific article "Optimal selection of movables shelves under cargo to person picking mode". The criteria for this choice are:

- quantity of items that could be withdrawn;
- total distance that the robot must complete, to go to take the shelf and then take it to the picking station starting from the position occupied in a certain moment;

- possibility of completing an order.

The input that are known are:

- the number of type of goods [M]
- the number of shelves [K]
- the quantity vector of goods stored on a shelf [$R_j = a_{1j}, a_{2j},..,a_{Mj}$ with j=1,2,..,K], calculated for each shelf.
- the quantity vector of goods in the batch of orders to be picked [$Q=q_i$ with i=1,2,..,M]
- roundtrip time to move shelf j to the picking paltform [w_j with j=1,2,..,K]

The algorithm is divided into two phases. First you determine if there is a shelf that with the goods stored inside it can complete an order and thus free up a space in the picking station for the next order. This is determined by calculating, for each shelf, the number of items that can be withdrawn [b_j]:

$P_j = p_{1j},p_{2j},..,p_{Mj}$ where $p_{ij} = \min\{a_{ij},q_i\}$ with j =1,2,..,K and i =1,2,..,M
$b_j = \sum_{i=1}^{M} p_{ij}$

Then comparing this value with the sum of items needed to complete an order given by $\sum_{i=1}^{M} q_i$. If exists $\sum_{i=1}^{M} p_{ij} = \sum_{i=1}^{M} q_i$ and it is unique that shelf becomes the next on the list. If it exists but is not unique, the choice is made based on the roundtrip time of those shelves. The one with a shorten travel time is the chosen one. While if it does not exist it goes to the second phase.

In the second phase the choice is made based on the number of items that can be taken and the distance to be travelled to take the shelf to the picking station. Therefore the ideal choice would be a shelf with many items and near the station or an advantageous combination of these two values. So we need a parameter that defines a ranking of the shelves and leads to the choice of the most optimal shelf. This parameter is determined by the relationship between distance and quantity of articles, rate(P_j) = $\frac{w_j}{b_j}$. The shelf with the lower ratio will be the optimal one to take $j^* = \arg\min\{\text{rate}(P_j)\}$.

The table elements used in this part of simulation are:

- Distance: table that allow reading only, in which all distances are reported: both those between any two shelves and those between each shelf and picking station.

  To determine the distance we always follow the shortest route. Below, the method used to assign these distance is explained through an example (Fig. 5.6 and Fig 5.7).

Fig 5.6 – Graphic example

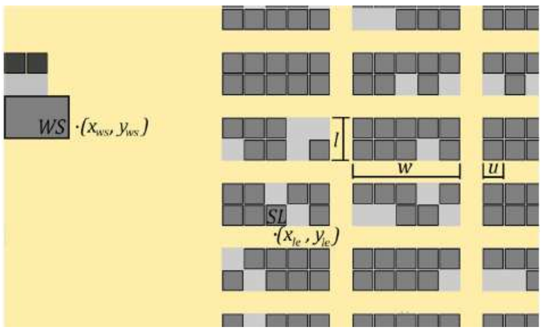
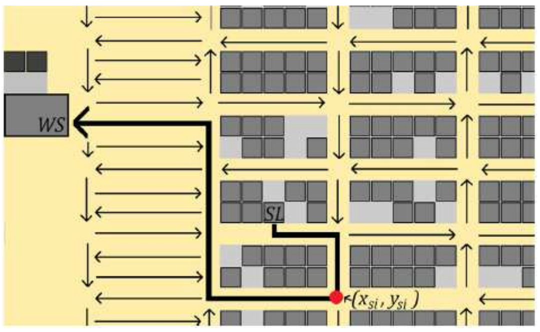


Fig 5.7 - Shortest route



The general formula for calculating the distance is:

$$D = u + d_{le,si} + |x_{si} - x_{ws}| + |y_{si} - y_{ws}| + \Delta_{le,ws}$$

The robot's journey is always straight and each lane has only one direction of transit. To calculate the route, first must be found the point of start intersection (X_{si},Y_{si}), ie the point at which the distance from the station begins to decrease, this point varies according to the position of the robot.

The shortest distance can e divided into 4 parts, the first is the exit from the shelf storage area (u). The second part is the distance $d_{le,si}$ between the starting point and the point of start intersection (X_{si},Y_{si}). The third part is the Manhattan distance between the point of start intersection and the picking station, equal to $|x_{si} - x_{ws}| + |y_{si} - y_{ws}|$. The fourth and last part is the section to be travelled inside the picking station in the buffer area.

- Pod_item, SumItem_bj, NowPodContent and PodList perform the same function as in the picker to parts system, previously seen;

- OrderAss1, OrderAss2, OrderAss3 are respectively the lists of first, second and third order present in the station. In the algorithm are the first lists used to determine if there is a shelf that completes them. Then if there is no solution, these lists are merged into the BatchItem list.

Method element used in this simulation part is:

- ChoicePod: method which develops the algorithm described above and as a result gives the list of shelves to be taken. Method activated whenever a Kiva robot is free. Code reported in the appendix A.6.

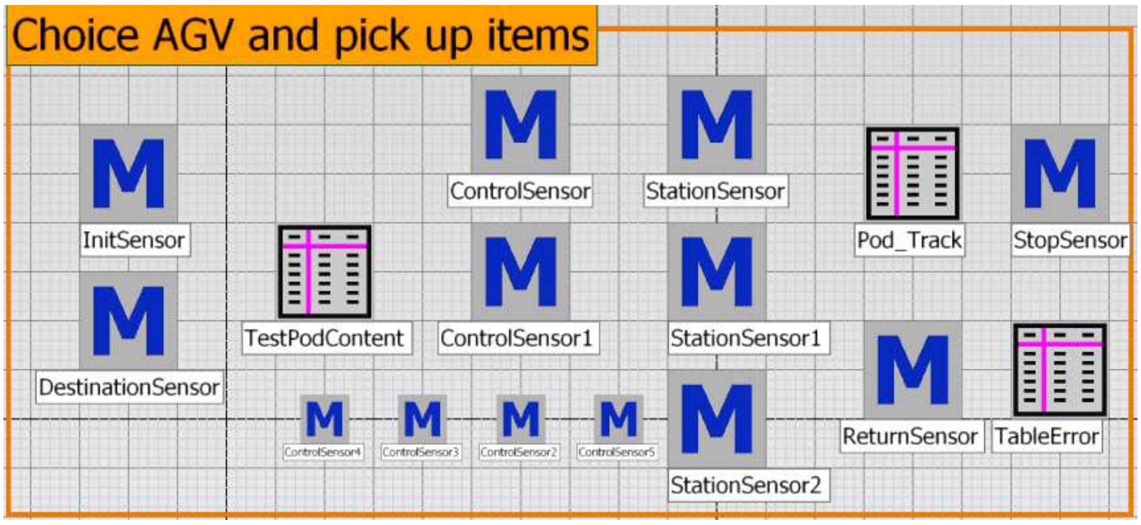
5.4. Choice of the AGV and pick up items in the station in Kiva systems

Here the commands to control the robot destinations are defined, so the purposes of this simulation part are:

- start the robots with the arrival of the first order
- drive the Kiva robots to their destination, which can be a shelf or picking station
- check the movements of the robots within the waiting area of the station, so as to respect the optimal sequence determined
- stop the robots in front of the pickup area for as long as necessary
- bring the robots to the parking area when the orders are finished

Below we see, in detail, which commands were used to fulfill the aim. The commands used for this part are shown below (Fig. 5.8):

Fig 5.8 – Screenshot: commands to choice and movement AGVs



This part consist mainly of method elements that are inserted in sensor present in the transit area. Each time a robot activates them, the method sends new indications to them.

The method elements used in this part of simulation are:

- InitSensor: it is used to start the robots, moving them from their stop position to their target. It is actived only at the beginning of the working day when the first order arrives.
- DestinationSensor: method applied to the sensor placed in each shelf position. When the robot enters, if the robot is empty then the method allow it to pick up the shelf and direct it to the station. Instead if the robot is already loaded, the method make it discharge the shelf and launches the ChoicePod method, seen previously, that allows to find a new goal. When the new target has been found the method sends indications to the robot about its position and sends it there.
- ControlSensor methods: they are used to direct the flow inside the waiting area of the station. If the shelf that arrives is the next one to be worked, this is sent to the operator's queue. While if it is not the next one to be worked, it is sent to the waiting area and passed when the time is right.

- StationSensor methods: they are used to stop the robots in front of the operator to allow manual removal of all the products that are needed. When the operation is completed the robots can restart and leave the picking station. Code reported in the appendix A.9. The initial part of the code serves to unlock the robots that are in the waiting area of the station and insert them in the operator's queue correctly.
- ReturnSensor: method applied to the sensor at hte exit of the picking station. Its purpose is to address each robot to the position associated with the platform that it is transporting. To know this position the sensor reads the number of the shelf and finds it in the Pod_Track table.
- StopSensor: method that is activated only when all orders have been completed. It is used to send the robots to their parking area.

6. Results

As introduced in the previous chapters, there are different types of setup born from the combination of:

- number of operator (from 1 to 5)
- types of picking systems used (pickers to parts or parts to picker), but more in detail by the technology used (barcode or pick to light for the first type and Kiva system for the second type)
- number of open orders in the picking station (1,3 or 5)
- collection path performed by the operator (return or traversal path)

After having processed daily orders for a period of six months, have returned average values regarding the following items:

- Elapsed time to complete daily order
- Picker throughput calculated in order lines per hour
- Average time for order
- Number of shelves visited to complete an order
- Times and percentages for each activity of the picker (getting information, searching, picking, confirm and travel)
- Picker utilization
- Times and percentages for each activity of the Kiva robot (waiting and working)
- Kiva robot utilization

The results are shown below, broken down by type of picking system used. The results for the Barcode Handheld system are shown in Table 6.1, the results for the pick-to-light system are shown in Table 6.2 and the results for the Kiva system are shown in Table 6.3. As you can see from the following tables, the number of simulated operators for each type of system is different, this choice was made because on each picking system the total time for the completion of the orders must be at least lower than a work shift of 8 hour.

Tab 6.1 – BarcodeHandheld system results

Cases	No. Pickers	Open Order	Path	Elapsed Time [h]	Throughput [order lines/h]	Average time for order [min]	Number mobile racks required for order [mobile racks]	Picking time Picker [h]	Packing time Picker [h]	GetInformation time Picker [h]	Confirm time Picker [h]	Search time Picker [h]	Displacement time Picker [h]	Picker Utilization [%]
BarcodeHandheld_1BatchOrder_1Worker	1	1	return	33.51.58	132	13.32	25	06.05.15	02.30.00	03.43.30	05.01.30	09.57.00	06.34.49	25,3
	1	1	traversal	32.19.31	139	12.55	25	06.05.15	02.30.00	03.43.30	05.01.30	09.57.00	05.02.16	26,6
BarcodeHandheld_1BatchOrder_2Worker	2	1	return	16.55.59	133	06.46	25	03.02.37	01.15.00	01.51.45	02.30.45	04.58.30	03.17.24	25,3
	2	1	traversal	16.09.45	139	06.27	25	03.02.37	01.15.00	01.51.45	02.30.45	04.58.30	02.31.08	26,6
BarcodeHandheld_1BatchOrder_3Worker	3	1	return	11.17.19	133	04.31	25	02.01.45	00.50.00	01.14.30	01.40.30	03.19.00	02.11.36	25,3
	3	1	traversal	10.46.30	139	04.18	25	02.01.45	00.50.00	01.14.30	01.40.30	03.19.00	01.40.45	26,6
BarcodeHandheld_1BatchOrder_4Worker	4	1	return	08.27.59	133	03.23	25	01.31.18	00.37.30	00.55.52	01.15.22	02.29.15	01.38.42	25,3
	4	1	traversal	08.04.52	139	03.13	25	01.31.18	00.37.30	00.55.52	01.15.22	02.29.15	01.15.34	26,6
BarcodeHandheld_1BatchOrder_5Worker	5	1	return	06.46.23	133	02.42	25	01.13.03	00.30.00	00.44.42	01.00.18	01.59.24	01.18.57	25,3
	5	1	traversal	06.27.54	139	02.35	25	01.13.03	00.30.00	00.44.42	01.00.18	01.59.24	01.00.27	26,6
BarcodeHandheld_3BatchOrder_1Worker	1	3	return	29.25.06	152	11.46	20	06.04.21	02.29.38	03.42.57	05.00.45	09.55.32	02.12.22	29,1
	1	3	traversal	28.35.34	156	11.26	20	06.04.21	02.29.38	03.42.57	05.00.45	09.55.32	01.22.48	30,0
BarcodeHandheld_3BatchOrder_2Worker	2	3	return	14.42.33	152	05.53	20	03.02.10	01.14.49	01.51.28	02.30.22	04.57.46	01.06.11	29,1
	2	3	traversal	14.17.47	156	05.43	20	03.02.10	01.14.49	01.51.28	02.30.22	04.57.46	00.41.24	30,0
BarcodeHandheld_3BatchOrder_3Worker	3	3	return	09.48.22	152	03.55	20	02.01.27	00.49.52	01.14.19	01.40.15	03.18.30	00.44.07	29,1
	3	3	traversal	09.31.51	156	03.48	20	02.01.27	00.49.52	01.14.19	01.40.15	03.18.30	00.27.36	30,0
BarcodeHandheld_3BatchOrder_4Worker	4	3	return	07.21.16	152	02.56	20	01.31.05	00.37.24	00.55.44	01.15.11	02.28.53	00.33.05	29,1
	4	3	traversal	07.08.53	156	02.51	20	01.31.05	00.37.24	00.55.44	01.15.11	02.28.53	00.20.42	30,0

Tab 6.2 – PicktoLight system results

Cases	No. Pickers	Open Order	Path	Elapsed Time [h]	Throughput [order lines/h]	Average time for order [min]	Number mobile racks required for order [mobile racks]	Picking time Picker1 [h]	Packing time Picker1 [h]	GetInformation time Picker1 [h]	Confirm time Picker1 [h]	Displacement time Picker1 [h]	Picker1 Utilization [%]	Picking time Picker2 [h]	Packing time Picker2 [h]	GetInformation time Picker2 [h]	Confirm time Picker2 [h]	Displacement time Picker2 [h]	Picker2 Utilization [%]	Picking time Picker3 [h]	Packing time Picker3 [h]	GetInformation time Picker3 [h]	Confirm time Picker3 [h]	Displacement time Picker3 [h]	Picker3 Utilization [%]
Pick-to-light_1BatchOrder_1Worker	1	1	return	22.03.40	203	08.49	25	03.34.30	02.30.00	06.03.45	01.13.30	08.42.00	27,5	--	--	--	--	--	--	--	--	--	--	--	--
	1	1	traversal	20.20.56	221	08.08	25	03.34.30	02.30.00	06.03.45	01.13.30	06.59.11	29,9	--	--	--	--	--	--	--	--	--	--	--	--
Pick-to-light_1BatchOrder_2Worker	2	1	return	11.04.13	203	04.25	25	01.47.09	01.14.56	03.01.42	00.36.43	04.23.21	27,4	01.47.20	01.15.04	03.02.02	00.36.47	04.23.00	27,5	--	--	--	--	--	--
	2	1	traversal	10.12.54	220	04.05	25	01.47.26	01.15.08	03.02.11	00.36.48	03.31.30	29,8	01.47.03	01.14.52	03.01.33	00.36.41	03.32.37	29,7	--	--	--	--	--	--
Pick-to-light_1BatchOrder_3Worker	3	1	return	07.25.15	202	02.58	25	01.11.18	00.49.52	02.00.55	00.24.26	02.59.04	27,2	01.11.47	00.50.12	02.01.44	00.24.35	02.56.48	27,5	01.11.21	00.49.54	02.01.00	00.24.27	02.57.36	27,4
	3	1	traversal	06.51.02	219	02.44	25	01.11.35	00.50.04	02.01.24	00.24.32	02.23.45	29,6	01.11.30	00.50.00	02.01.15	00.24.30	02.24.02	29,6	01.11.50	00.50.14	02.01.49	00.24.36	02.22.42	29,7
Pick-to-light_3BatchOrder_1Worker	1	3	return	16.58.01	264	06.47	20	03.33.58	02.29.38	06.02.51	01.13.19	03.37.50	35,7	--	--	--	--	--	--	--	--	--	--	--	--
	1	3	traversal	15.47.15	284	06.18	20	03.33.58	02.29.38	06.05.51	01.13.19	02.27.28	38,4	--	--	--	--	--	--	--	--	--	--	--	--
Pick-to-light_3BatchOrder_2Worker	2	3	return	08.18.48	270	03.19	20	01.47.46	01.15.22	03.02.45	00.36.55	01.36.21	36,7	01.46.43	01.14.38	03.00.59	00.36.34	01.38.50	36,4	--	--	--	--	--	--
	2	3	traversal	07.53.45	284	03.09	20	01.47.46	01.15.22	03.02.45	00.36.55	01.10.50	38,6	01.46.43	01.14.38	03.00.59	00.36.34	01.13.51	38,4	--	--	--	--	--	--
Pick-to-light_3BatchOrder_3Worker	3	3	return	05.38.45	265	02.15	20	01.11.58	00.50.20	02.02.03	00.24.39	01.09.10	36,1	01.11.44	00.50.10	02.01.39	00.24.34	01.09.48	36,0	01.10.52	00.49.34	02.00.12	00.24.17	01.12.54	35,6
	3	3	traversal	05.18.25	282	02.07	20	01.11.50	00.50.14	02.01.48	00.24.36	00.50.05	38,4	01.11.04	00.49.42	02.00.31	00.24.21	00.51.52	38,0	01.11.38	00.50.06	02.01.29	00.24.32	00.48.52	38,4
Pick-to-light_5BatchOrder_1Worker	1	5	return	15.34.39	288	06.13	16	03.34.30	02.30.00	06.03.45	01.13.30	02.12.54	39,0	--	--	--	--	--	--	--	--	--	--	--	--
	1	5	traversal	14.52.42	300	05.57	16	03.33.24	02.29.14	06.01.53	01.13.07	01.35.13	40,6	--	--	--	--	--	--	--	--	--	--	--	--
Pick-to-light_5BatchOrder_2Worker	2	5	return	07.48.36	288	03.07	16	01.46.33	01.14.31	03.00.42	00.36.30	01.10.28	38,6	01.47.56	01.15.29	03.03.02	00.36.59	01.03.04	39,3	--	--	--	--	--	--
	2	5	traversal	07.28.53	300	02.59	16	01.47.32	01.15.12	03.02.44	00.36.50	00.42.19	41,1	01.46.57	01.14.48	03.01.23	00.36.39	00.45.09	40,9	--	--	--	--	--	--
Pick-to-light_5BatchOrder_3Worker	3	5	return	05.18.58	282	02.07	16	01.10.38	00.49.24	01.59.47	00.24.12	00.54.51	37,6	01.11.57	00.50.19	02.02.01	00.24.39	00.47.09	38,6	01.11.01	00.49.40	02.00.26	00.24.20	00.47.49	38,6
	3	5	traversal	05.03.42	297	02.01	16	01.10.58	00.49.38	02.00.21	00.24.19	00.37.31	39,8	01.11.21	00.49.54	02.01.00	00.24.27	00.36.24	40,0	01.12.07	00.50.26	02.02.18	00.24.42	00.33.57	40,3

Tab 6.3 – Kiva system results

Cases	No. Pickers	Open Orders	No. AGV	Elapsed Time [h]	Throughput [order lines/h]	Average time for order [min]	Number mobile racks required for order [mobile racks]	Working time Picker [h]	Waiting time Picker [h]	Picker utilization [%]	Working time AGV1 [h]	Waiting time AGV1 [h]	AGV1 utilization [%]	Working time AGV2 [h]	Waiting time AGV2 [h]	AGV2 utilization [%]	Working time AGV3 [h]	Waiting time AGV3 [h]	AGV3 utilization [%]	Working time AGV4 [h]	Waiting time AGV4 [h]	AGV4 utilization [%]	Working time AGV5 [h]	Waiting time AGV5 [h]	AGV5 utilization [%]	Working time AGV6 [h]	Waiting time AGV6 [h]	AGV6 utilization [%]
KivaSystem_3BatchOrder_3AGV	1	3	3	10.00.56	449	04.00	7	07.30.42	02.29.58	75.0	08.33.51	01.26.06	85.5	08.34.26	01.26.55	85.6	08.30.51	01.28.47	85.1	--	--	--	--	--	--	--	--	--
KivaSystem_3BatchOrder_4AGV	1	3	4	08.34.36	524	03.25	7	07.31.45	01.02.40	87.8	06.28.10	02.05.51	75.4	06.24.50	02.08.32	74.8	06.28.03	02.08.52	75.4	06.23.29	02.10.30	74.5	--	--	--	--	--	--
KivaSystem_3BatchOrder_5AGV	1	3	5	07.59.56	562	03.11	7	07.32.15	00.27.47	94.2	05.08.05	02.52.09	64.2	05.08.34	02.50.59	64.3	05.09.17	02.51.23	64.4	05.07.49	02.52.18	64.1	05.05.56	02.54.02	63.8	--	--	--
KivaSystem_3BatchOrder_6AGV	1	3	6	07.45.37	579	03.06	7	07.32.35	00.12.56	97.2	04.19.56	03.25.25	55.8	04.16.19	03.29.18	55.0	04.17.24	03.28.14	55.3	04.17.14	03.28.35	55.2	04.16.32	03.28.53	55.1	04.18.32	03.27.33	55.5
KivaSystem_5BatchOrder_3AGV	1	5	3	08.39.52	519	03.27	5	07.31.39	01.09.13	86.8	06.57.30	01.43.59	80.3	06.59.11	01.41.47	80.6	06.56.18	01.43.31	80.0	--	--	--	--	--	--	--	--	--
KivaSystem_5BatchOrder_4AGV	1	5	4	07.52.41	571	03.09	5	07.32.23	00.21.07	95.5	05.14.58	02.37.47	66.6	05.11.11	02.40.30	65.8	05.12.09	02.40.14	66.0	05.12.51	02.40.49	66.2	--	--	--	--	--	--
KivaSystem_5BatchOrder_5AGV	1	5	5	07.41.09	585	03.04	5	07.32.53	00.08.10	98.2	04.09.14	03.30.23	54.0	04.10.13	03.31.23	54.3	04.08.43	03.33.29	53.9	04.09.32	03.32.25	54.1	04.08.49	03.30.41	45.7	--	--	--
KivaSystem_5BatchOrder_6AGV	1	5	6	07.39.09	588	03.03	5	07.33.36	00.05.32	98.8	03.31.01	04.08.40	45.9	03.29.36	04.10.57	45.6	03.29.27	04.10.20	45.6	03.28.00	04.10.22	45.3	03.27.37	04.10.50	45.2	03.28.02	04.11.34	45.3

6.1 Validation of the models

After having developed and executed all the models born from the combination of the various setups, there was the need to know the quality of each model, that is with what refinement they simulate the real cases. From this it derives the validity and correctness of the results and only with a positive feedback they can be compared.

The feature considered to value the correctness of the models is the quantity of items that one operator can collect in an hour of work. Which corresponds to a kind of worker's performance, the higher the number the more productive it is. Getting these values for the types of simulated picking systems is easy, but we also need to determine the reference values that come from real cases. These samples have been found in the literature, see the following table:

Tab 6.4 – Literature and simulation throughput

PICKING SYSTEMS	**LITERATURE THROUGHPUT**	**SIMULATION THROUGHPUT**	**VALIDATED**
BARCODE SYSTEM	100 ÷ 200	≈ 130	YES
PICK TO LIGHT SYSTEM	200 ÷300	≈ 210	YES
KIVA SYSTEM	600 ÷ 700	≈ 600	YES

Therefore, assumining what has been said, the models are considered sufficiently suitable to represent real cases.

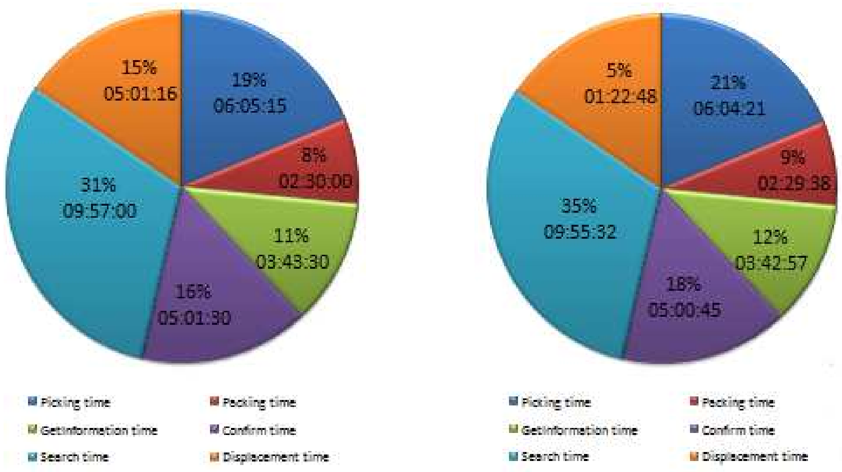
6.2. Displacement time

The time with greater importance and linked to the quality of the model, which is found only after the simulations have been processed, is the time dedicated to travel, called displacement time. The time is also linked to the size and configuration of the warehouse and to the collection characteristics that the operator must follow, for example the route and the type of picking system. While the rest of the times (picking time, packing time, getinformation time, confirm time and search time) are defined fixed, for each article collected, the displacement time is variable. More articles you pick up in one lap, more the time allocated to one article will be less. For the Kiva system, on the other hand, there is no displacement time but there is a time in which the operator is not busy, called waiting time.

Below, diagrams are shown (from chart 6.5 to chart 6.10) show the division and incidence of times in the different picking systems. The times are shown in the case which only one operator works, because in the case of multiple pickers the percentages of times are identical but only the value changes. For the Kiva system times are also seen with the variant of the number of robots, because these work for one picker and this affects his productivity.

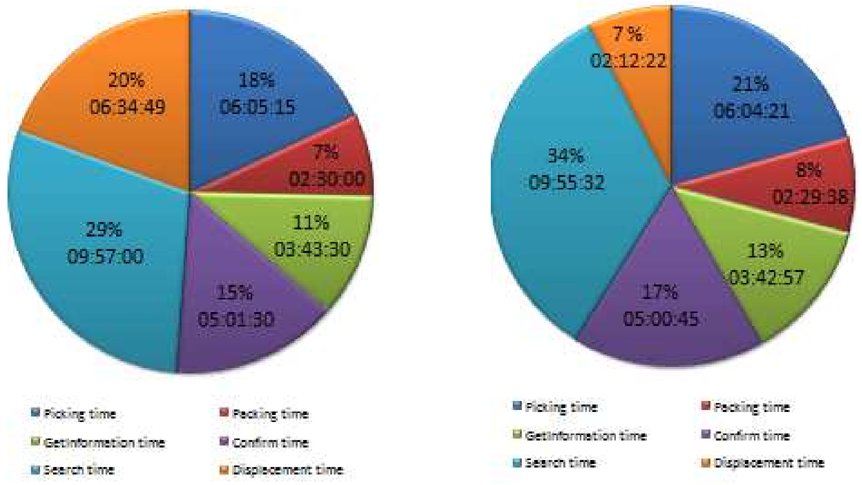
- Times in the barcode system with traversal path and 1 or 3 batch orders

Chart 6.5 – Rappresentation of the division of time in the barcode system, with traversal path and 1(left) or 3 (right) batch orders



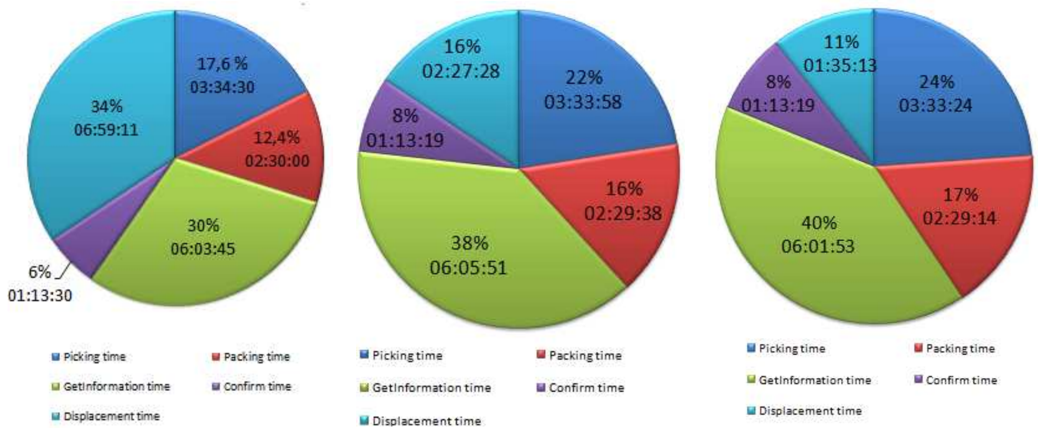
- Times in the barcode system with return path and 1 or 3 batch orders

Chart 6.6 – Rappresentation of the division of time in the barcode system, with return path and 1(left) or 3 (right) batch orders



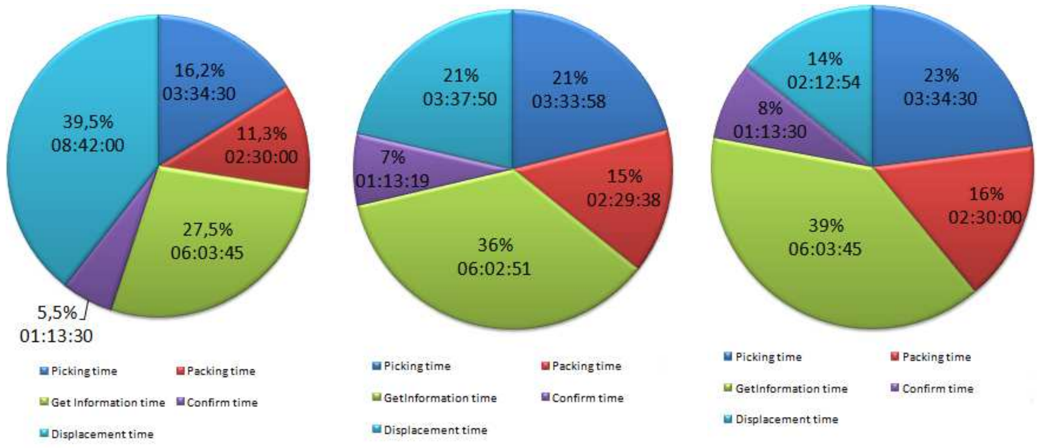
- Times in the pick to light system with traversal path and 1,3 or 5 batch orders

Chart 6.7 – Rappresentation of the division of time in the pick to light system, with traversal path and 1(left), 3 (middle) and 5(right) batch orders



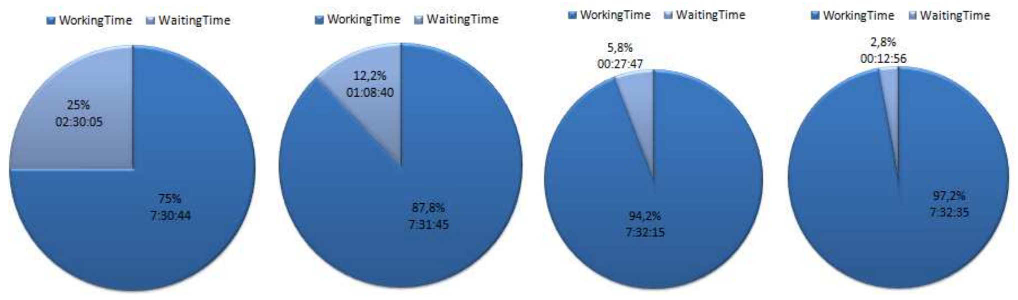
- Times in the pick to light system with return path and 1,3 or 5 batch orders

  Chart 6.8 – Rappresentation of the division of time in the pick to light system, with return path and 1(left), 3 (middle) and 5(right) batch orders



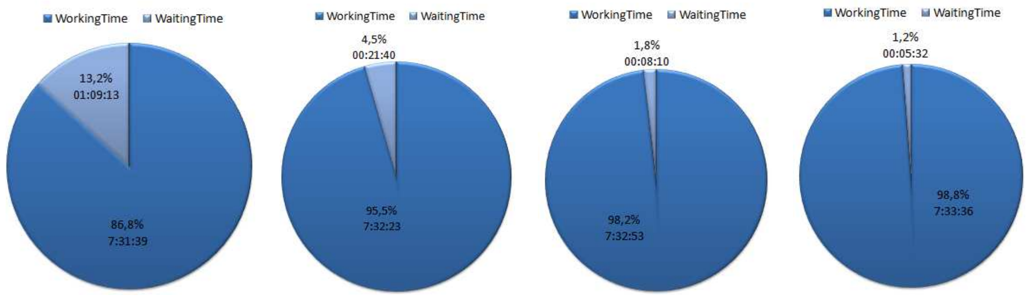
- Times in the Kiva system with 3 batch orders

  Chart 6.9 – Rappresentation of the division of time in the Kiva system, with 3 batch orders and 3,4,5 and 6 robots



- Times in the Kiva system with 5 batch orders

  Chart 6.10 – Rappresentation of the division of time in the Kiva system, with 5 batch orders and 3,4,5 and 6 robots



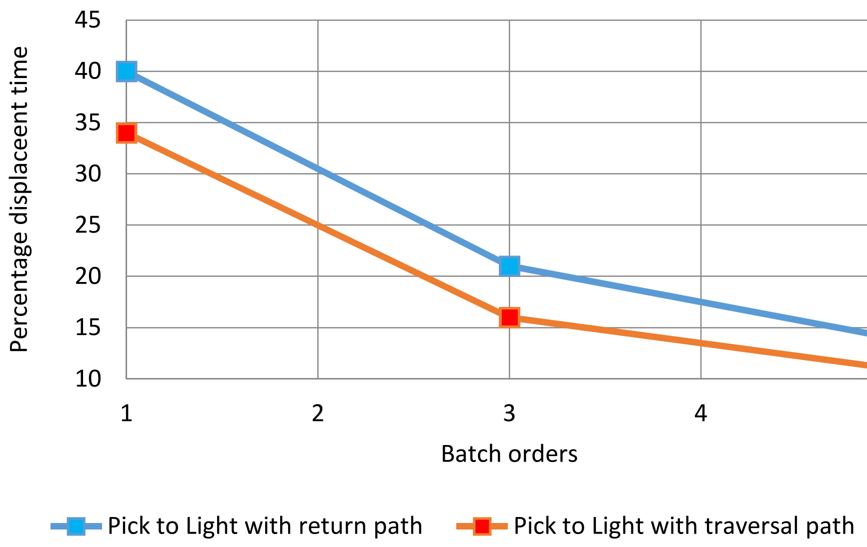
To see in detail, the percentages of the displacement time are isolated in a table (Tab 6.11).

Tab 6.11 – Detail percentage displacement time

	Path	1 Open Order	3 Open Orders		5 Open Orders	
Barcode system	Return	20 %	7 %		/	
	Traversal	15 %	5 %		/	
Pick to Light system	Return	40 %	21 %		14%	
	Traversal	34 %	16 %		11 %	
Kiva system	/	/	3 AGVs	25 %	3 AGVs	13,2 %
			4 AGVs	12,2 %	4 AGVs	4,5 %
			5 AGVs	5,8 %	5 AGVs	1,8 %
			6 AGVs	2,8 %	6 AGVs	1,2 %

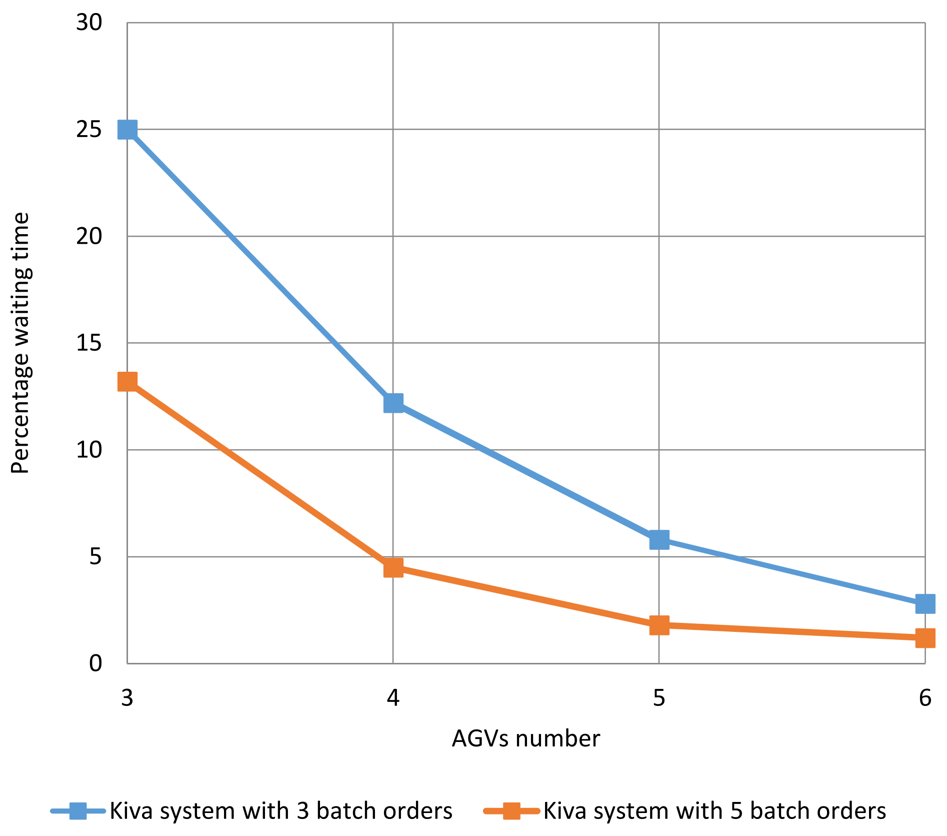
For picker to parts systems, we note that a system that uses a return path, compared to a traversal path, has a longer displacement time. In the case of only one order in the station this difference is about 5%, if the orders opened at the station increase the time difference between the routes decreases to 4% with 3 open orders and 3% with 5. Therefore, it can be deduced, that for this configuration the traversal path is more advantageous. Now looking at the same picking system and modifying only the number of batch order, we can see that the displacement time drastically decrease, in fact for each lap the operator picks up more items along his route avoiding to go over the same area several times. A particular thing that can be seen is that passing from 1 to 3 open orders the reduction of displacement time occurs in a greater way than passing from 3 to 5 batch orders (Chart 6.12).

Chart 6.12 – Difference between displacement times compared to the number of batch orders



For the Kiva system, on the other hand, the operator does not move through the warehouse to pick up the articles but waits for them to be brought directly to the station, so in this system there is no displacement time but it is replaced by the waiting time. Time in which there is no shelf in the picking area and therefore must wait for the next robot arrive with the next shelf. It can be seen, as with the same number of AGVs with a greater number of open orders, the waiting time decreases. It is a normal reaction because on average each shelf brought to the station has a greater number of picked items, so the operator will take more time for the pick up, giving more time to the next robot to reach the station. While observing the value of the percentage of waiting time by varying the number of AGVs, we note that there is an exponential decrease, passing from 3 to 4 robots the percentage is halved and it is the same thing passing from 4 to 5 and from 5 to 6 robots (Chart 6.13).

Chart 6.13 – Difference between waiting times compared to the number of AGVs



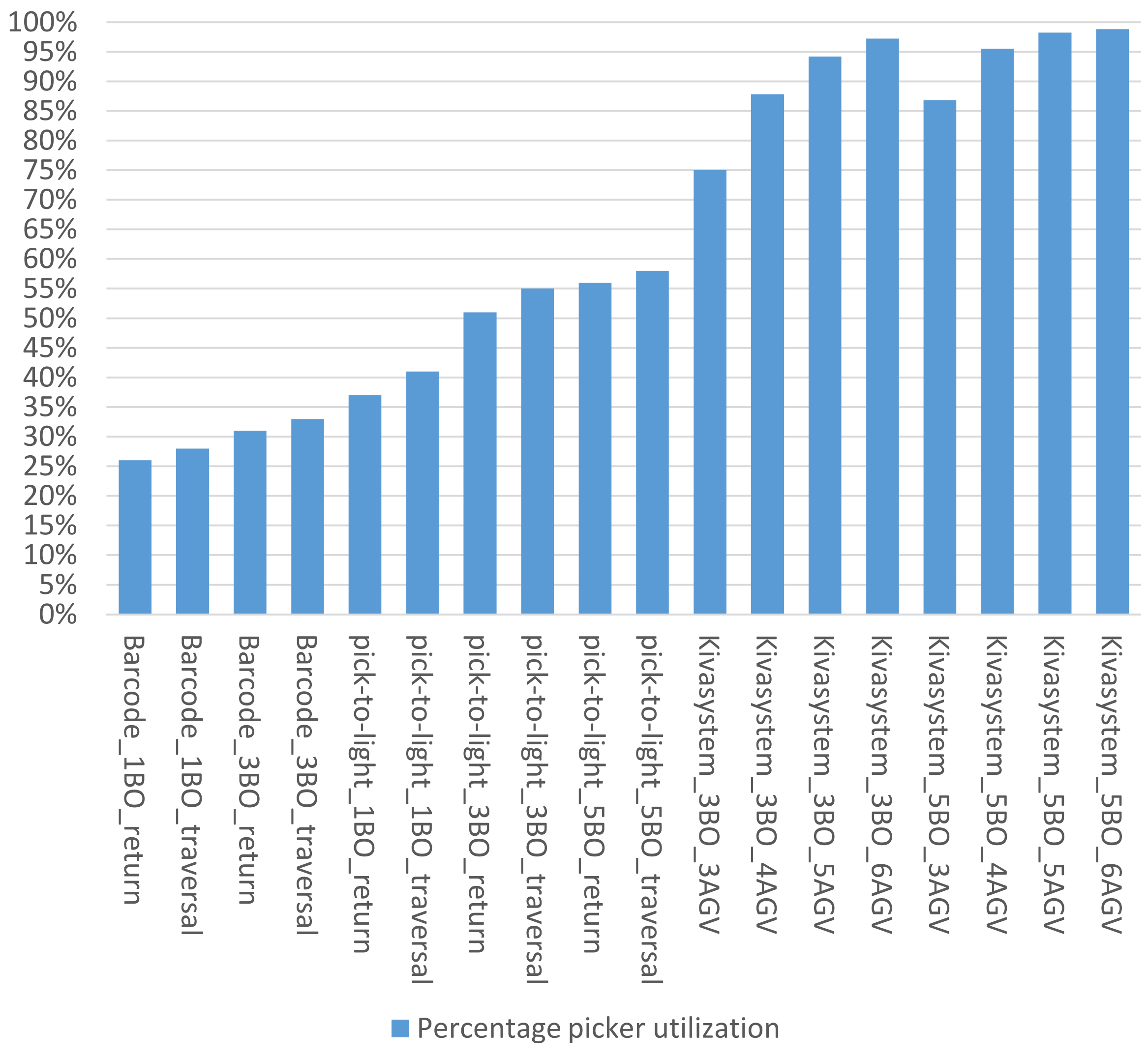
Now we move on to compare the percentages shown in table 6.11 also between the various systems, this is possible thinking of using a different number of operators for each system so that the total time to process all orders is at least 8 hours, that is a work shift. The number of operators does not affect the percentages and, having also a similar total time, they can be compared with each other. For barcode systems 4 operators are used, for the pick to light system with 1 open order they are 3 and the rest of the pick to light systems 2 operators are used. We see that for the parts to picker system it is more convenient to use 5 open orders, while for the other type of system using 3 or 5 batch orders does not have an noticeable effect, so even to facilitate collection I think it is more convenient to use 3 open orders.

It is evident that if the aim is to reduce the lost time due to the displacement the best choice is to use the Kiva system which reduces the waiting time up to the order of 1-2 % of the total time.

6.3. Picker utilization

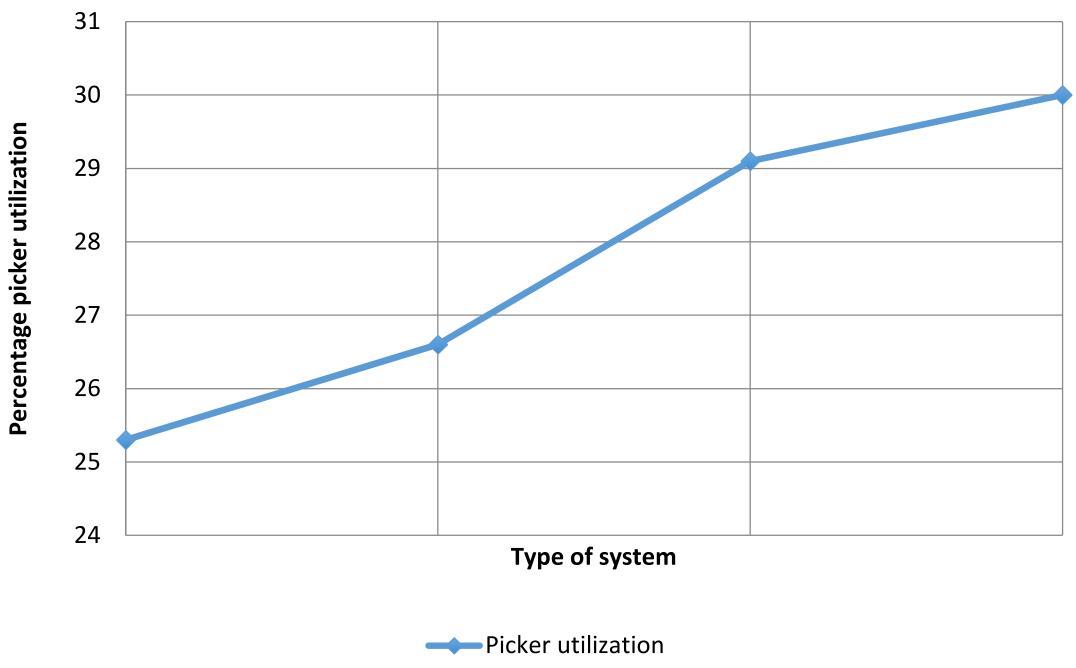
Reading the data in tables 6.2,6.3 and 6.4, we can see how the percentage of operator use varies, this values are shown in the following graph (Chart 6.14):

Chart 6.14 – Percentage picker utilization



The percentage of use of the operator represents the time in which the worker interacts directly with the items to be taken and performs actions that lead to the conclusion of the order respect to the total work time. The operations considered to calculate this percentage are the manual pick up activity and the movement activity of the picked product from the trolley to the order box. Therefore all other activity, like confirmation, getting information, research and displacement are activity that do not increase the value of the product and do not increase the operator productivity. From the graph it can be seen at best which are the ranges of percentages obtained for each picking systems. For the barcode system we pass from a minimum of 25,3% to a maximum of 30%, view in chart 6.15, very similar percentages which remain low even with the modification of the type of route and number of open orders.

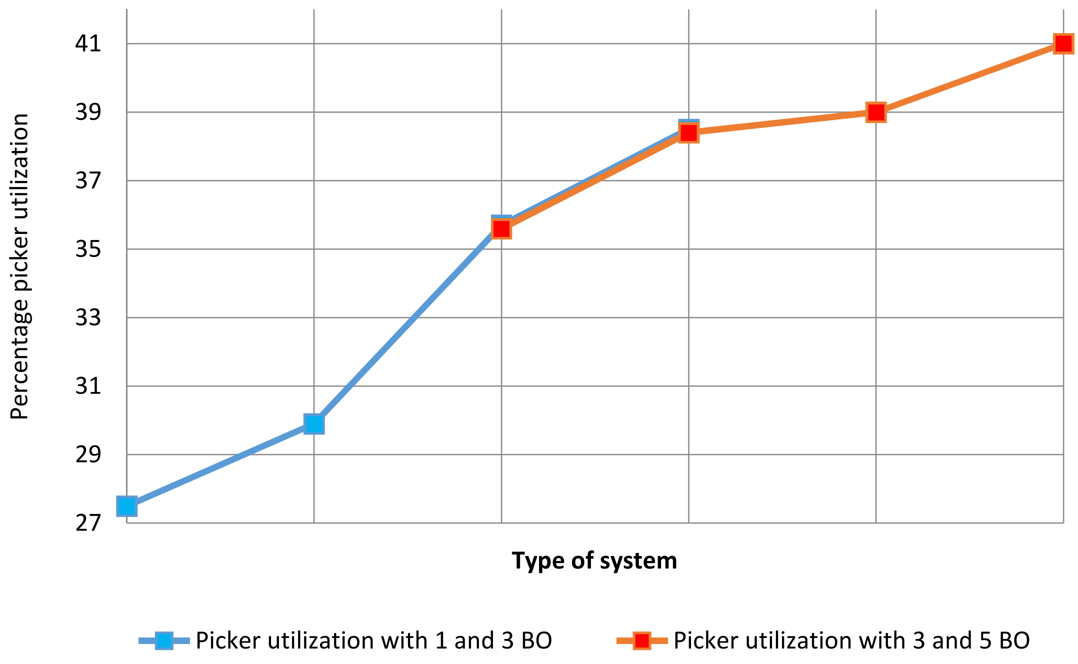
Chart 6.15 – Picker utilization of Barcode system



It is important to note that the percentage of use does not depend on the number of operators.

For the pick to light system the trend is very similar to the one just seen, going from a minimum use percentage of 27,5 to a maximum of 38,4% considering 1 and 3 open orders as for the barcode system. It reaches a maximum percentage of 41 if 5 batch orders are used.

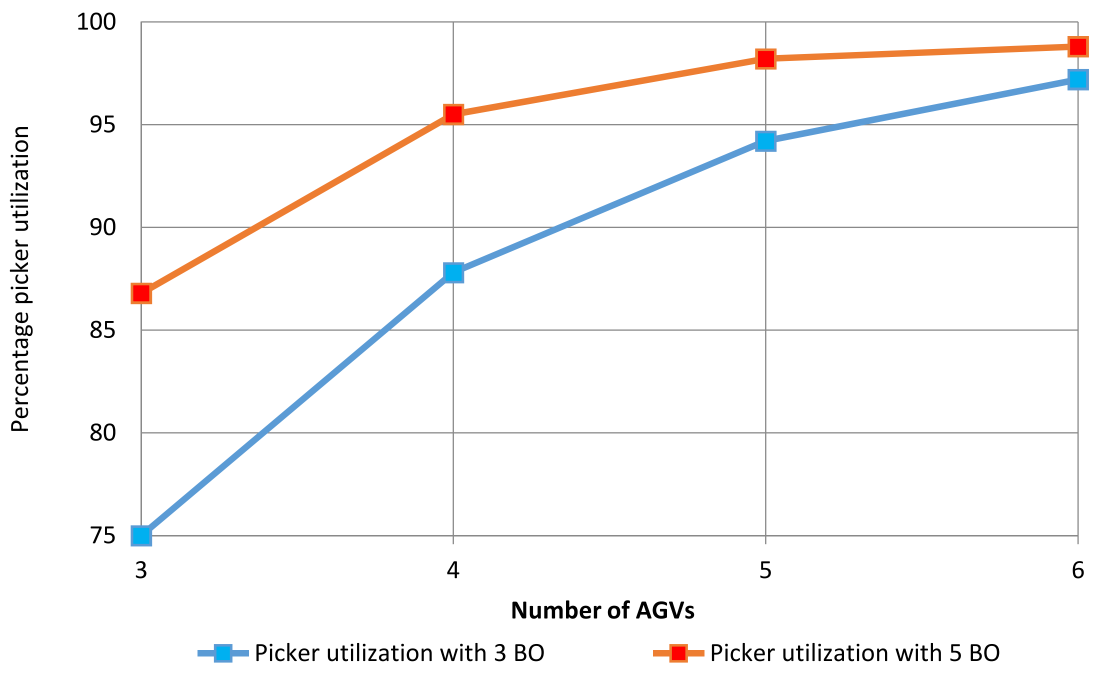
Chart 6.16 – Picker utilization of Pick to Light system



Here we can see better how the passage from 1 to 3 batch orders gives more pronounced advantage compared to the passage from 3 to 5 batch order and how the change of the type of path affects only 1-2%.

Moving on to the Kiva system, the percentage of use is very high and to change it you can act on the number of open orders in the station or on the number of robots that follow the work of the operator. For systems with 3 batch orders it goes from a minimum of 75% with 3 robots to a maximum of 97,2% with 6 robots, while for systems with 5 batch orders it passes from a minimum of 86,8% with 3 robots to a maximum of 98,8% with 6 robots, see chart 6.17

Chart 6.17 – Picker utilization of KIva system



Beside the percentage of use of the operator in the Kiva system it is necessary to speak also of the percentage of use of the single robot. When the number of robot used is minimum, that is 3, their performance is high around the 80-85%, so they are often in movement to fulfill their purpose and briefly stuck in the picking zone. This high percentage is however linked to a discrete use of the operator, 75% if 3 batch orders are used or 86% with 5, because the operator must wait for the robots that reach the station. Therefore, to have even greater use of the operator, the number of AGVs must be increased, passing to a percentage of their use of 45-50% linked to a maximum use of the operator, 97-98%.

There are very high percentages compared to those seen for the other systems, this because the time of displacement has been eliminated and the technology of the system tends to reset the times to obtain information and for the research, through the use of a laser pointer, and the confirmation time through a code reader.

6.4. Throughput

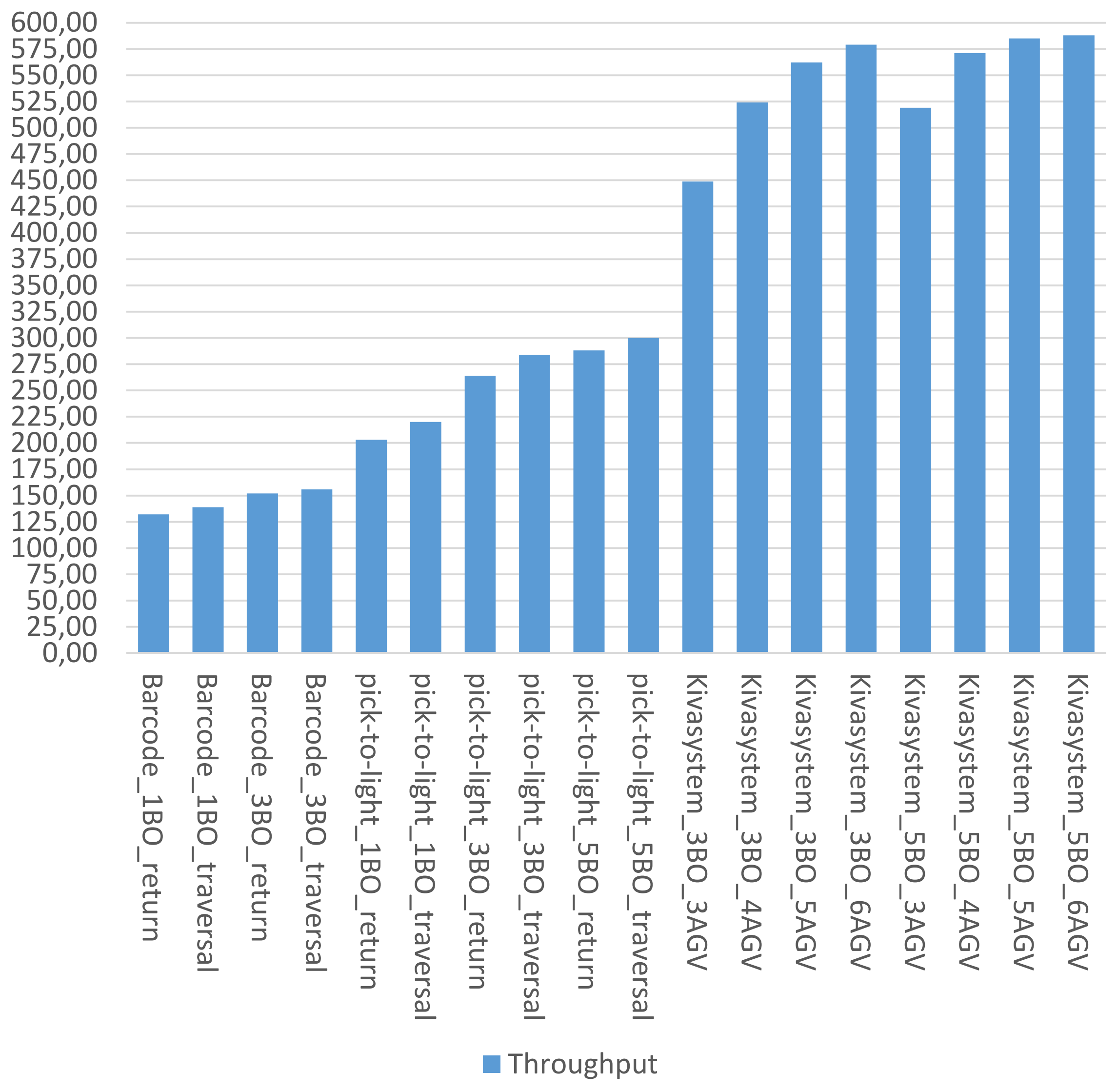
As already defined the throughput is the operator's performance. It determines how many order lines the operator can complete in an hour of work. So this value depends on how many and what operations the picker has to perform (like get information, search, confirm, pick up and move) and how long he takes to do these operations. To give an example, the confirmation time for barcode system is 4,02 seconds, for the pick to light system it is 0.98 seconds while for the Kiva system it is not an activity to perform thanks to the technology used.

When two or more operators work in the system, their performance is the same because they work on different orders and, except for some occasions, their path is out of phase and different.

In the barcode system there is a throughput of 132 items/hour using one open order and a return path, this increases to 139 items/hour if the type of path is modified. Using 3 batch order the throughput is 152 items/hour with return path and 156 items/hour with traversal path. In the pick to light system there is a throughput of 203 items/hour using one open order and a return path, this increases to 220 items/hour if the type of path is modified. Using 3 batch order the throughput is 264 items/hour with return path and 284 items/hour with traversal path. Using 5 batch order the throughput is 288 items/hour with return path and 300 items/hour with traversal path.

In the Kiva system there is a throughput of 449 items/hour using 3 open orders and 3 AGVs, this increases to 579 items/hour if the number of AGVs is increased up to 6. Using 5 batch order the throughput is 519 items/hour with 3 AGVs and 588 items/hour with 6 AGVs. (Chart 6.18)

Chart 6.18 – Throughput of the picking systems



7. Economic evaluation

Starting by defining the cost function, which is useful for conducting an economic evaluation and comparing the different systems. This function, called the hourly cost function C_h^j , where j id to define the type of technology to which it refers, includes 4 cost components:

- Hourly cost depending on the number of stock locations, $C_{h,SL}^j$;
- Hourly cost depending on the number of pickers, $C_{h,P}^j$;
- Hourly fixed cost, $C_{h,F}^j$;
- Hourly cost for the KIva system implementation, $C_{h,K}^j$.

$$C_h^j = C_{h,SL}^j + C_{h,P}^j + C_{h,F}^j + C_{h,K}^j$$

Considering the notation reported in table 7.1, the previous formula can now be set out as follows:

$$C_h^j = \frac{\mathrm{n_{SL}} * \mathrm{C_{SL}^j}}{\mathrm{h_{SL}}} + \left(\mathrm{C_{h,P}} + \frac{\mathrm{C_{d,P}^j}}{\mathrm{h_{d.P}}}\right) * \left\lceil \frac{\mathrm{n_R}}{\mathrm{p^j}} \right\rceil + \frac{\mathrm{C_F^j}}{\mathrm{h_F}} + \frac{\mathrm{C_K^j}}{\mathrm{h_K}}$$

Table 7.1 – Hourly cost function components and notation

Cost component	*Expression*	*Notation*	*Description*
Stock location hourly cost	$C^j_{h,SL} = \frac{n_{SL} * C^j_{SL}}{h_{SL}}$	n_{SL} C^j_{SL} [€] h_{SL} *[h]*	*Number of available stock location* *Stock location unitary cost** *Stock location devices total usage hours*
Picker hourly cost	$C^j_{h,P} = \left(C_{h,P} + \frac{C^j_{d,P}}{h_{d.P}}\right) * \left\lceil \frac{n_R}{p^j} \right\rceil$	$C_{h,P}$ *[€/h]* $C^j_{d,P}$ [€] $h_{d.P}$ *[h]* n_R $[rows]$ p^j *[rows/h]*	*Picker hourly cost* *Picker device cost** *Picker devices total usage hours* *Number of requested picking rows* *Throughput**
Fixed hourly cost	$C^j_{h,F} = \frac{C^j_F}{h_F}$	C^j_F *[€]* h_F *[h]*	*FIxed cost** *Fixed elements total usage hours*
Kiva system technology hourly cost	$C^j_{h,K} = \frac{C_K}{h_K}$	C_K *[€]* h_K $[h]$	*Kiva system technology cost* *Kiva technology total usage hours*

*Variable according to the considered technology j

p^j is the performance of the system and remains fixed for a given simulated model, while n_R is the number of order lines and this depends on the daily orders. Therefore their report indicates the number of operators that must be used to complete the orders. The hourly cost depends on the number of order lines, therefore increasing or decreasing the order lines will have a different cost and the choice of the most convenient system to use will also change.

The table 7.2 reports some of the possible main cost items for each of the considered picking systems. The costs related to the number of stock locations generally consist of two main components: the cost of purchase of the required specific equipment and the cost of installation of such materials. The picker costs relate to the devices supplied to the picker and the hourly pay. Finally, for all the technologies the reported fixed costs concern the purchase of the management server and of the software and about maintenance.

Table 7.2 – Picking technology main cost

	C_{SL}^{j}	$C_{d,P}^{j}$	C_K	C_F^{j}
Barcode handheld	*- Barcode cost* *- Barcode installation cost*	*- Barcode reader cost* *- Picker cart cost*	/	*- server and software cost*
Pick to light	*- Lights cost* *- Confirmation device cost* *- lights and confirmation device cost*	*- Handheld cost* *- Picker cart cost*	/	*- server and software cost*
Kiva system	/	/	*- start up kit* *- kit installation*	*- maintenance cost*

In this chapter, in addition to defining the hourly cost, we also modify some factors that determine its value, such as fixed costs and the hourly cost of the operator. Once the new cost has been calculated, it is possible to determine the impact that these components have on the cost. The table 7.3 reports the various cost components obtained from specific industry catalogues and from information derived from the warehouse managers interviews. For the calculation of $h_{SL}, h_F\,, h_{d,P}\ \ and\ \ h_K$ ten years were considered, with an eight-hour work shift for 220 days a years.

Table 7.3 – Cost components values

Cost component	Factor	Barcode system	Pick to light system	Kiva system
$C_{h,SL}^{j} = \frac{n_{SL} * C_{SL}^{j}}{h_{SL}}$	n_{SL} C_{SL}^{j} [€] h_{SL} [h]	2000 1,10 € 17600 h	2000 50 € 17600 h	/
$C_{h,P}^{j} = \left(C_{h,P} + \frac{C_{d,P}^{j}}{h_{d.P}}\right) * \left\lceil\frac{n_R}{p^j}\right\rceil$	$C_{h,P}$ [€/h] $C_{d,P}^{j}$ [€] $h_{d.P}$ [h] n_R [rows] p^j [rows/h]	30 €/h 2800 € 17600 h variable from 132 to 156	30 €/h 2800 € 17600 h variable from 203 to 297	30 €/h / 17600 h variable from 449 to 588
$C_{h,F}^{j} = \frac{C_{F}^{j}}{h_F}$	C_F^j [€] h_F [h]	300000 € 17600 h	300000 € 17600 h	300000 € 17600 h
$C_{h,K}^{j} = \frac{C_K}{h_K}$	C_K [€] h_K [h]	/	/	1000000 €

In the following, the first parameter that has been varied in the plotting of the hourly cost function for all systems is the number of picked rows n_R. The systems are divided into two graphs based on their type of collection path, those with return path are shown in the figure 7.4 and those with traversal path in the figure 7.5. Both graphs show the trend of the Kiva system, which is compared to traditional methods. The Kiva system considered is the one with 5 batch orders and 6 AGVs, therefore with a yield of 588 items/hour, this is because the company does not advertise prices but approximate values are given for typical warehouse setup with less than 50 robots. Besides the line charts of the different solutions, in the lower part a bar graph is shown, reporting the areas of convenience for the various systems; that is, the most convenient system, according the different numbers of requested picking rows, is each time reported.

From the observation of the graph reported for the different cost values in figures 7.4 and 7.5 some interesting considerations and comparisons can be performed. The trend of the curves and the graph below ,that shows the best strategy to use, are identical. The difference in value of items/hour between the same system with a return or traversal path is not high enough to modify the economic evaluation and make that system the best in different area of the chart. Therefore only one graph will be analyzed, knowing that the results obtained are the same for the two types of path.

In general it is observed that the increase in the number of picked rows leads to an increase in the trend observable for all the curves, mainly due to the increase in the number of pickers needed to satisfy the requested warehouse performance. Focusing on the different technologies considered, for low values of n_R the most advantageous solution is the barcode system, which can boast a low cost of the equipment. The picking times for this system, as already seen, are high which guarantees a limited collection performance. In fact this system is the best up to a request of 160 items/hour. Increasing the demand, the most economical system is the pick to light system, which uses a more expensive technology but guarantees shorter collection times and therefore, with a high demand, a lower quantity of manpower. These features lead the system to be the most advantageous up to 780 or 860 items/hour if a system with 3 or 5 batch orders is used respectively. For the Kiva technology, instead, the cost is absolutely impacting making this picking solution non-competitive for low values of n_R. It begins to be competitive at around 600 items/hour and becomes the absolute best after 860 items/hour. The system is characterized by much shorter picking times and therefore higher performance than all other systems.

This last representation is also employed in the subsequent analysis in which fixed costs have been changed with respect to the starting configurations (see figure 7.6). For the analysis of the variation of the annual fixed costs it was decided to find solutions with different modifications of the parameter. The cases are going to increase first to 60000 and then to 90000 the annual fixed costs of the most technologically advanced systems, that is the pick to light system and the Kiva system. With these changes you can see a variation in the area with low values on n_R, barcode system becomes the most convenient solution in several intervals, while the cases in which it was more appropriate to use the pick to light system decrease. At high values of n_R the most convenient system does not change, the Kiva system always remains the best system for requests exceeding 850 items/hour.

Table 7.4 – Hourly cost function for system with return path

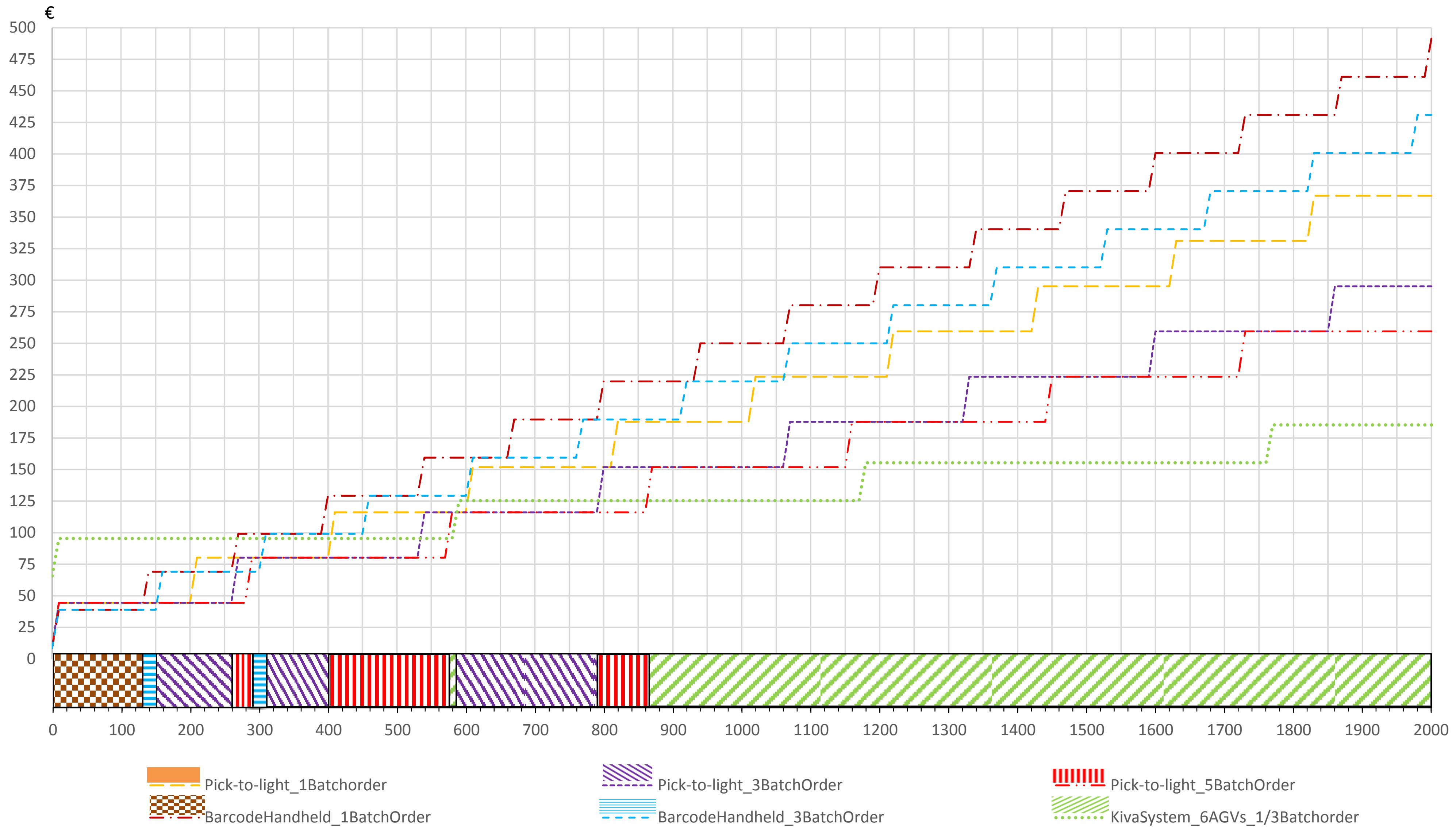


Table 7.5 – Hourly cost function for system with traversal path

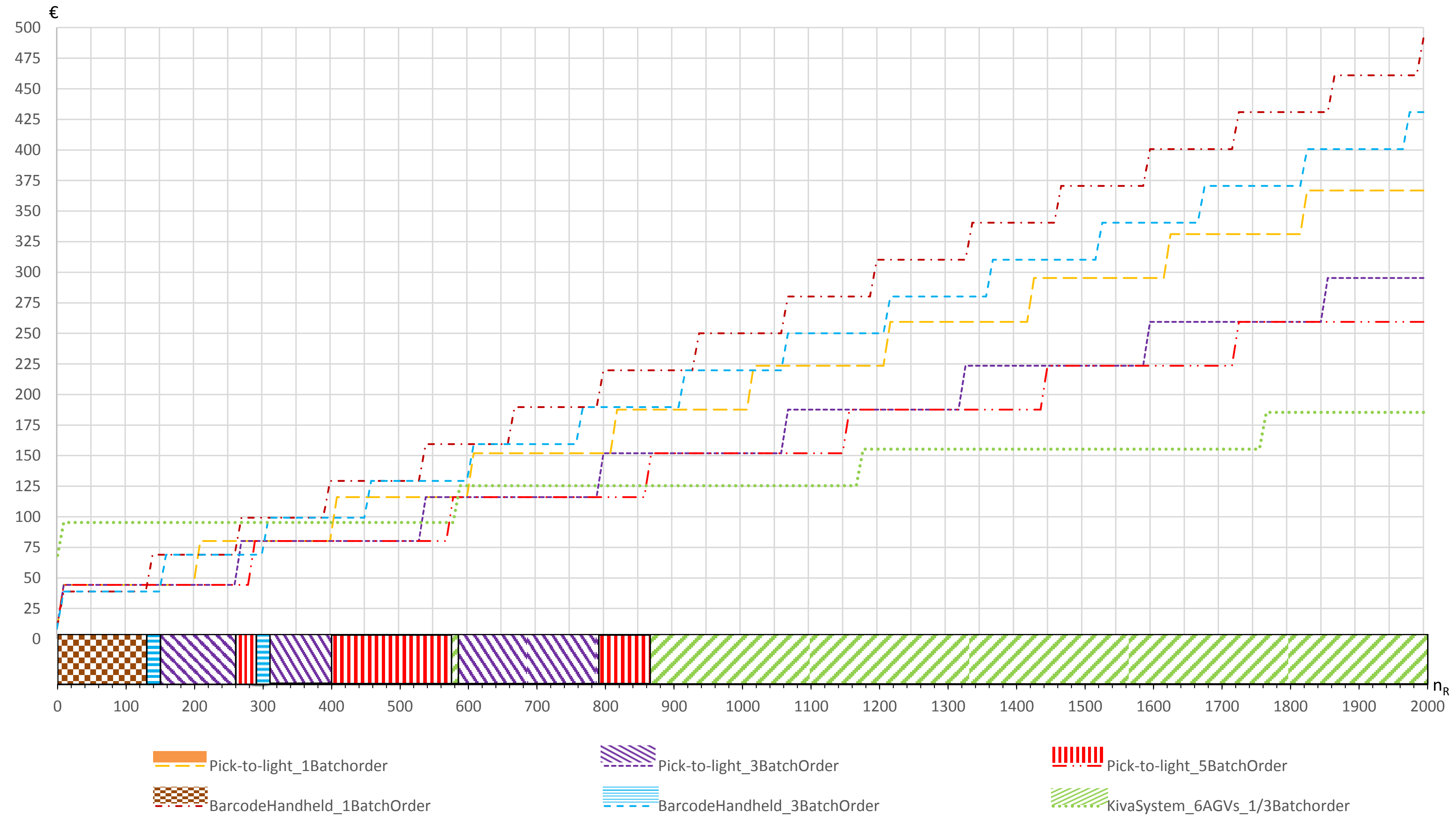
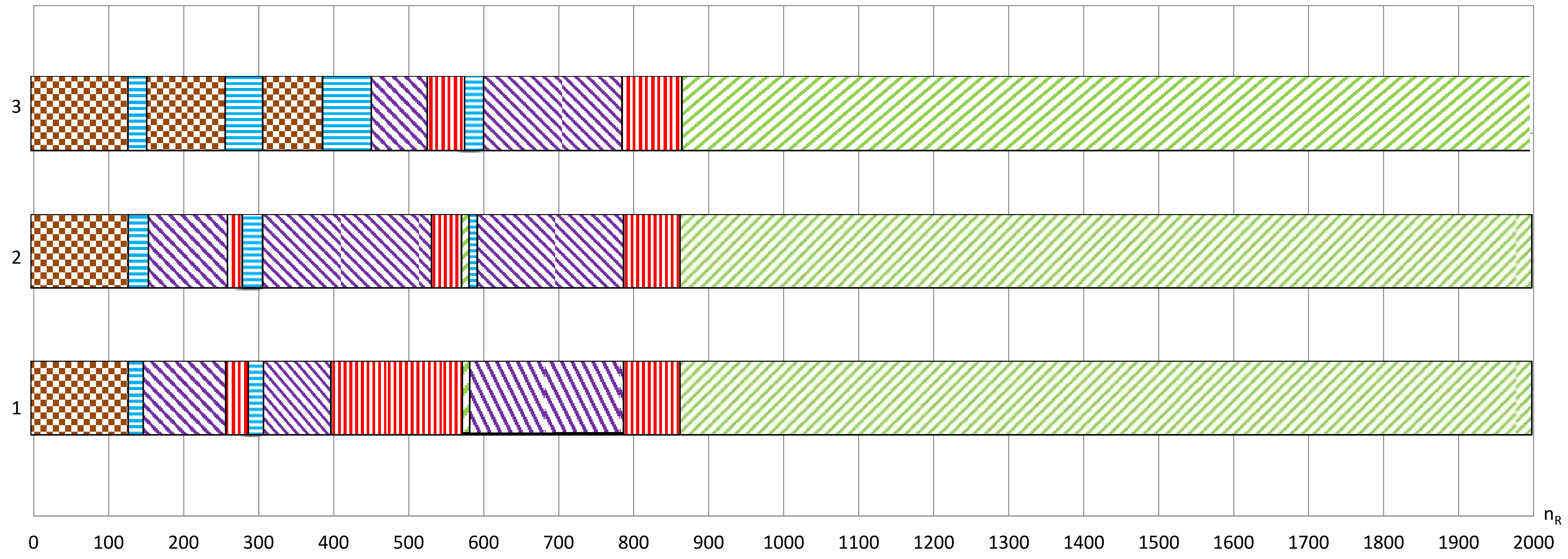


Table 7.6 – Hourly cost function changing C_F



Case	Parameter changed	Barcode system 1 Batch order ▦	Barcode system 3 Batch orders ☰	Pick to Light system - 1 BO 🟧	Pick to Light system - 3 BO ▧	Pick to Light system - 5 BO ▥	Kiva system ▨
1	C_F	30000 €/year	30000 €/year	30000 €/year	30000 €/year	30000 €/year	30000 €/year
2	C_F	30000 €/year	30000 €/year	60000 €/year	60000 €/year	60000 €/year	60000 €/year
3	C_F	30000 €/year	30000 €/year	90000 €/year	90000 €/year	90000 €/year	90000 €/year

8.Conclusion

Choosing the type of picking system is one o the most critical activities in the warehouse. Using the simulation we compared the performance of three different picking systems: barcode, pick to light and Kiva systems. Given the assumptions and the parameters of the systems, a first conclusion is that the performance of the Kiva system has a value twice that of the pick to light system and five times greater than the barcode system. Obviously this is reflected in a lower use of manpower but with a high percentage of utilization. It also appears that the number of open orders is an important control parameter to ensure a high percentage of use.

A second result, obtained from the simulation, is the value of the displacement/waiting time. Incident time for the picker to parts systems which therefore this reduces performance, while very reduced for the Kiva syatem with the possibility of reducing it to 0 by increase the number of AGVs.

While from the economic evaluation, given the configuration of the warehouse, it is concluded that the Kiva system is the only choice for requests over 850 items/hour, while the other two systems are used for the lower values of the number of order lines to hour and the intervals of competence depend on the parameters used, generally barcode system for low values of n_R and pick to light for average values.

Appendices

A.1. Init method

```
// variables of the code
var i: integer
var c: integer
var j, n: integer
var fixpods, pod, t, f: object
var num: integer
var buffer: object

//create 192 pod in own position
for i:=1 to 192
        t := StoredTrack[i]
        pod:= .MUs.Pod.create(t)
next

//create type of item (650)
.Mus.Item1.create(ItemclassA)
.Mus.Item2.create(ItemclassA)
.Mus.Item3.create(itemclassB)
.Mus.Item4.create(itemclassB)
.Mus.Item5.create(itemclassB)
.Mus.Item6.create(itemclassB)
.Mus.Item7.create(itemclassC)
.Mus.Item8.create(ItemclassC)
..
.. // until Item650, each in own buffer

// create all items (item for its stored quantity)
for n:=1 to 117
        num:=QuantityItem[2,n]
        for j:=1 to num
                QuantityItem[1,n].create(itemClassA)
        next
next
for n:=118 to 292
        num:=QuantityItem[2,n]
        for j:=1 to num
                QuantityItem[1,n].create(itemclassB)
        next
next
for n:=293 to 650
        num:=QuantityItem[2,n]
        for j:=1 to num
                QuantityItem[1,n].create(bufferOrders)
        next
next

//create AGVs (if Kiva system)
.MUs.AGV1.create(root.Track128)
.MUs.AGV2.create(root.Track127)
.MUs.AGV3.create(root.Track126)
..
..
```

```
//call method for subdivide items
&FillPods.execute

//choice of pod to bring to the picking station (if Kiva system)
&ChoisePod.execute
```

A.2. FillPods method

```
// variables of the code
var i, j, k, m: integer
var pod_: object
var item_obj: object
var item_class: object
var item_string: string
var num: integer
var part: string

// move item of class A from buffer (itemclassA) to the respective pod using table PodContents
for k:=1 to 70    //70 is the number of shelves of classA
        pod_:=str_to_obj(Pods.read(k))
        //search and select in the column of the pod
        for i:=1 to PodContent.yDim
                if PodContent[k,i]>0
                //read quantity of item
                        part:= obj_to_str(item[i])
                        num:=PodContent[k,i]
                        for j:=1 to num
                        // move the right parts to the pod
                                for m:=itemClassA.numMU downto 1
                                        item_obj:=itemClassA.MU(m)
                                        item_class :=item_obj.class
                                        item_string := obj_to_str(item_class)
                                        if item_string = part
                                                item_obj.move(pod_)
                                                exitLoop
                                        end
                                next
                        next
                end
        next
next
// move item of class B from buffer (itemclassB) to the respective pod using table PodContents
for k:=71 to 122 // 52 is the number of shelves of classB
        pod_:=str_to_obj(Pods.read(k))
        //search and select in the column of the pod
        for i:=1 to PodContent.yDim
                if PodContent[k,i]>0
                //read quantity of item
                        part:= obj_to_str(item[i])
                        num:=PodContent[k,i]
                        for j:=1 to num
                        // move the right parts to the pod
                                for m:=itemclassB.numMU downto 1
                                        item_obj:=itemclassb.MU(m)
                                        item_class :=item_obj.class
                        item_string := obj_to_str(item_class)
                        if item_string = part
                            item_obj.move(pod_)
                            exitLoop
                        end
                    next
                next
            end
        next
    next
// move item of class B from buffer (itemclassB) to the respective pod using table PodContents
for k:=123 to 192 // 70 is the number of shelves of classC
    pod_:=str_to_obj(Pods.read(k))
    //search and select in the column of the pod
    for i:=1 to PodContent.yDim
        if PodContent[k,i]>0
        // read quantity of items
            part:= obj_to_str(item[i])
            num:=PodContent[k,i]
            for j:=1 to num
            // move the right parts to the pod
                for m:=bufferOrders.numMU downto 1
                    item_obj:=bufferOrders.MU(m)
                    item_class :=item_obj.class
                    item_string := obj_to_str(item_class)
                    if item_string = part
                        item_obj.move(pod_)
                        exitLoop
                    end
                next
            next
        end
    next
next
```

A.3. SetOrderInStation method

```
// I want that when an assembly order is finisched an new order come to assign following the
table Source_Orders

@.~.assemblyList.deleteContents

var _BillOfMaterial : object := current.BillOfMaterial
var _OrderInStation : object := current.OrderInStation
var i:integer

for i:=1 to BillOfMaterial.yDim
    if _BillOfMaterial[@.origin,i]>0
        _OrderInStation[_BillOfMaterial[0,i],@.~] += _BillOfMaterial[@.origin,i]
        @.~.AssemblyList.AppendRow(_BillOfMaterial[0,i].name,_BillOfMaterial[@.origin,i])
    end
next
```

A.4. End Sim method (for Kiva System)

```
var num: real
var num_str: string
var workingtime,endtime_num, a : integer
var endtime: time
var billofmaterialDay : object

// workingtime operator
TotalTime := eventcontroller.simTime
workingtimeWorker:= timeworker.sum({8,1}..{8,*})
workingtime := timeworker.sum({8,1}..{8,*})
endtime := eventcontroller.simtime
endtime_num := time_to_num(endtime)
num := (workingTime / endtime) * 100
num_str := num_to_str(num)
workingtimeworker100 := num_str

//waitingtime operator
waitingTimeWorker := endtime -workingtimeworker
num := (waitingTimeWorker / endtime) * 100
num_str := num_to_str(num)
waitingtimeworker100 := num_str

// throughput and order time
a := timeworker.sum({2,1}..{2,*})
throughput := (a / endtime_num ) * 3600
AverageTimeOrder:= endtime_num/150
AverageNumberPodOrder := numpodlist/150

// agv1
waitingtimeAGV1 := timeaGV1.sum({3,1}..{3,*})
num := (waitingtimeAGV1 / endtime ) *100
num_str := num_to_str(num)
waitingtimeAGV1100 := num_str
workingtimeAGV1 := endtime - waitingTimeAGV1
num := (workingtimeAGV1 / endtime ) *100
num_str := num_to_str(num)
workingtimeagv1100 := num_str
..
.. // agv2,agv3…..agv6

// save data
dati[1,orderday] := orderday
dati[2,orderday] := totalTime
dati[3,orderday] := througput
dati[4,orderday] := workingTimeWorker
dati[5,orderday] := workingTimeWorker100
dati[6,orderday] := waitingTimeWorker
dati[7,orderday] := averageNumberPodOrder
dati[8,orderday] := averageTimeOrder
dati[9,orderday] := workingTimeAGV1
dati[10,orderday] := waitingTimeAGV1
dati[11,orderday] := workingTimeAGV2
dati[12,orderday] := waitingTimeAGV2
dati[13,orderday] := workingTimeAGV3
dati[14,orderday] := waitingTimeAGV3
```

```
// change order day and restart
orderday += 1
if orderday = 2
        .models.frame.billOfMaterial.deleteContents
        .models.frame.orderDay2.copyrangeTo({0,0}..{*,*},billOfMaterial,0,0)
        eventcontroller.reset
        eventcontroller.start
end
..
.. // for all 6 months
```

A.4. End Sim method (for picker-to-parts system)

```
var time_ : time
var num, a, b, n: integer

time_ := eventcontroller.simTime
TotalTime := time_
num := tableerror.sum({4,1}..{4,*})
// picking time worker
// time between pick-to-light and barcode are different
PickingTimeWorker := num * 0:04.8700
Pickingtimeworker100 := num_to_str(time_to_num(Pickingtimeworker / totaltime ) * 100)
// packing time worker
PackingTimeWorker := num * 0:02.0000
Packingtimeworker100 := num_to_str(time_to_num(Packingtimeworker / totaltime ) * 100)
WorkingTimeWorker := pickingtimeworker + packingtimeworker
Workingtimeworker100 := num_to_str(time_to_num(workingtimeworker / totaltime ) * 100)
// getinformation time worker
GetInformationTimeworker := num * 0:02.9800
GetInformationtimeworker100 := num_to_str(time_to_num(GetInformationtimeWorker / totaltime
) * 100)
// confirm time worker
ConfirmTimeworker := num * 0:04.0200
Confirmtimeworker100 := num_to_str(time_to_num(ConfirmtimeWorker / totaltime ) * 100)
// search time worker (only for barcode method)
SearchTimeworker := num * 0:07.9600
Searchtimeworker100 := num_to_str(time_to_num(searchtimeWorker / totaltime ) * 100)
// displacement time worker
displacementTimeWorker := totaltime - WorkingTimeWorker - Getinformationtimeworker -
confirmtimeworker
displacementtimeworker100 := num_to_str(time_to_num(displacementtimeworker / totaltime ) *
100)
// throughput and order time
num:= tableerror.sum({7,1}..{7,*})
averageNumberPodOrder := num/150
averageTimeOrder := eventcontroller.simTime / 150
averagetimelap := eventcontroller.simTime / 50
num:= tableerror.sum({4,1}..{4,*})
througput := (num / time_to_num(totalTime))*3600
// save data
dati[1,orderday] := orderday
dati[2,orderday] := totalTime
dati[3,orderday] := througput
dati[4,orderday] := pickingTimeWorker
dati[5,orderday] := pickingTimeWorker100
dati[6,orderday] := packingTimeWorker
```

```
dati[7,orderday] := packingTimeWorker100
dati[8,orderday] := workingTimeWorker
dati[9,orderday] := workingTimeWorker100
dati[10,orderday] := getinformationTimeWorker
dati[11,orderday] := getinformationTimeWorker100
dati[12,orderday] := confirmTimeWorker
dati[13,orderday] := confirmTimeWorker100
dati[14,orderday] := displacementTimeWorker
dati[15,orderday] := displacementTimeWorker100
dati[16,orderday] := averagenumberpodorder
dati[17,orderday] := averagetimeorder
dati[18,orderday] := averagetimelap
// change order day and restart
orderday += 1
if orderday = 2
        .models.frame.billOfMaterial.deleteContents
        .models.frame.orderDay2.copyrangeTo({0,0}..{*,*},billOfMaterial,0,0)
        eventcontroller.reset
        eventcontroller.start
end
..
.. // for all 6 months
```

A.5. ChoicePod method (for picker-to-parts systems)

```
param SensorID: integer, Front: boolean

var i,n,m,t,b,s :integer
var item,choise_pod : object
var j:integer := 0
var maxi,num: integer
var npodlist :integer
var choise_pod_string:string
var pod, item_obj, buffer : object
var item_str, item_string :string
var item_class: object

if lap > 0 //save time for error checking
        tableerror[2,lap] := eventcontroller.simTime
        tableerror[3,lap] := tableerror[2,lap] - tableerror[1,lap]
end
@.stopped := true
lap += 1
numpod := 0
.models.frame.batchitem.deletecontents
.models.frame.podlist.deleteContents
.models.frame.podlist1.deleteContents
// create batch order
for i:=1 to OrderInStation.YDim
        for n:=1 to OrderInStation.XDim
                if OrderInStation[n,i] = 1
                        item := OrderInStation[n,0]
                        j += 1
                        BatchItem[j]:= item
                end
        next
```

```
next
num:= batchItem.dim
tableerror[1,lap] := eventcontroller.simTime
if batchItem.Dim = 0
        @.destination := track98
end
// loop: search shelf with more items available
for npodlist:= 1 to num
        .models.frame.pod_Item.deleteContents
        .models.frame.sumItem_bj.deleteContents
        .models.frame.testpodContent.deletecontents
        .models.frame.nowpodContent.copyrangeTo({0,0}..{*,*},testpodContent,0,0)
        for i:=1 to BatchItem.Dim
                Pod_Item[0,i]:=BatchItem[i]
        next
        for n:=1 to Pods.Dim
                Pod_Item[n,0] := Pods[n]
        next
        for n:=1 to testPodContent.XDim
                for t:=1 to BatchItem.Dim
                        for m:=1 to testPodContent.YDim
                                if testPodContent[0,m] = batchItem[t] and testPodContent[n,m]
                                > 0
                                        Pod_item[n,t] := 1
                                        testPodContent[n,m] -= 1
                                end
                        next
                next
        next
        for n:=1 to pod_item.xdim
                sumItem_bj[0,n] := pod_item[n,0]
                sumItem_bj[1,n] := Pod_item.sum({n,1}..{n,*})
        next
        maxi:= sumItem_bj.max({1,1}..{1,*}) //choice shelf
        if maxi /= 0
                for b:=1 to SumItem_bj.YDim
                        if sumItem_bj[1,b] = maxi
                                choice_pod:= sumItem_bj[0,b]
                        end
                next
                Podlist[1,npodlist]:=choice_pod //save the shelf
                podList[2,npodlist]:= maxi
                Choise_pod_string:=obj_to_str(choice_Pod)
                for n:=1 to NowPodContent.Xdim
                        if NowPodContent[n,0] = Choice_pod_string
                                for t := 1 to pod_item.ydim
                                        if pod_item[n,t] = 1
                                                for s:=1 to nowpodcontent.ydim
                                                        if nowpodContent[0,s] = pod_item[0,t]
                                                        and nowpodContent[n,s]>0
                                                                nowpodContent[n,s] -=1
                                                                //delete picked items
                                                        end
                                                next
                                        end
                                next
                                for m:=1 to Pod_Item.Ydim
                                        if Pod_Item[n,m] > 0
```

```
                              item := pod_Item[0,m]
                              for b:=1 to BatchItem.dim
                                  if item = batchItem[b]
                                  .models.frame.batchItem.remove(b)
                                      item := void
                                      //delete picked items
                                  end
                              next
                          end
                      next
                  end
              next
          end
      next  // loop until complete the picking list
      //save time for error checking
      tableerror[4,lap] := podlist.sum({2,1}..{2,*})
      tableerror[7,lap] := podlist.ydim
      tableerror[8,lap] := tableerror[4,lap] * 10.6900
      @.stopped := false
```

A.6. Meth_assembly

```
param SensorID: integer, Front: boolean

var numItem : integer
var time_ : time
var n,m : integer
var item_obj, item_class, item : object

numItem := orderinStation.sum({1,1}..{*,1})
time_ := numItem * 0:02.0000  //packing time
// move item to the station
for n:= BufferOrder.numMu downto 1
    item_obj := BufferOrder.MU(n)
    item_class := item_obj.class
    for m:= 1 to orderinStation.xdim
        if orderinStation[m,1] = 1
            item := orderinStation[m,0]
            if item = item_class
                item_obj.move(assembly1)
                orderinstation[m,1] -= 1
            end
        end
    next
next
track651.startPause(time_)
```

A.7. ChoicePod method (for Kiva system)

```
param SensorID: integer, Front: boolean
.models.frame.pod_item.deleteContents
.models.frame.sumItem_bj.deletecontents
.models.frame.batchItem.deletecontents
.models.frame.testPodContent.deletecontents
.models.frame.nowpodcontent.copyrangeTo({0,0}..{*,*},testpodContent,0,0)
```

```
var track,pod_: object
var pod_str, pod_stringa :string
var s,dist,num: integer
var times: time
var j: integer :=0
var j1: integer :=0
var j2: integer :=0
var j3: integer :=0
var i,b,n,t,m:integer
var choise_pod,item:object
var pod:object
var choise_pod_string:string
var mini:real
var colonna : integer
var maxi,riga , row:integer
var npodlist,colomn:integer
numpodlist += 1
if orderass1.empty = true and orderass2.empty = true and orderass3.empty = true
        podlist[numpodList] := void
        -- when the order are finished
end
if orderass1.empty = false or orderass2.empty = false or orderass3.empty = false
-- find the table indexes pod_item
        for b:=1 to orderass1.dim
                pod_Item[0,b]:=orderass1[b]
        next
        for n:=1 to pods.dim
                pod_item[n,0]:=pods[n]
        next
        -- find the contents of the table pod_item
        for n:=1 to testPodContent.xdim
                for t:=1 to orderass1.dim
                        for m:=1 to testpodContent.ydim
                             if testpodContent[0,m] = orderass1[t] and testpodContent[n,m] > 0
                             -- if it's true can take that article from the pod
                               pod_Item[n,t]:= 1
                               testpodcontent[n,m] -= 1
                             end
                        next
                next
        next
        -- define the table sumitem_bj
        for n:=1 to pod_Item.xdim
                sumitem_bj[0,n]:=pod_Item[n,0]
                sumItem_bj[1,n]:= pod_Item.sum({n,1}..{n,*})
                if sumItem_bj[1,n] = 0
                        sumItem_bj[1,n] := 0.01
                end
        next
        -- unavable shelves are excluded
        numpodlist -= 3
        pod := podlist.read(numpodlist)
        pod_stringa := obj_to_str(pod)
        for n:=1 to sumItem_bj.ydim
                if sumItem_bj[0,n] = pod_stringa
                        sumItem_bj[1,n] := 0.01
                end
```

```
next
numpodlist +=1
pod := podlist.read(numpodlist)
pod_stringa := obj_to_str(pod)
for n:=1 to sumItem_bj.ydim
        if sumItem_bj[0,n] = pod_stringa
                sumItem_bj[1,n] := 0.01
        end
next
numpodlist +=1
pod := podlist.read(numpodlist)
pod_stringa := obj_to_str(pod)
for n:=1 to sumItem_bj.ydim
        if sumItem_bj[0,n] = pod_stringa
                sumItem_bj[1,n] := 0.01
        end
next
numpodlist +=1
maxi := sumItem_bj.max({1,1}..{1,*})
-- know the amount of goods that each pod can give to the order, I have to see if there
is a pod that can complete the order
if maxi = orderass1.dim and orderass1.empty = false
        for row:= 1 to distance.ydim
                if distance[0,row] = pod_str
                        riga := row
                end
        next
        for n:=1 to sumItem_bj.ydim
                if sumItem_bj[1,n] = maxi
                        sumItem_bj[2,n]:= distance[n,riga] / sumItem_bj[1,n]
                end
                if sumItem_bj[1,n] /= maxi
                        sumItem_bj[2,n] := 1000
                end
        next
        mini := sumItem_bj.min({2,1}..{2,*})
        for b:=1 to sumItem_bj.ydim
                if sumItem_bj[2,b] = mini
                        choise_pod := sumItem_bj[0,b]
                end
        next
        -- if it's true that pod completes the order
        -- if I find a result, that pod is put in the list of pods to be taken
        podlist[numpodlist]:= choise_pod
        choise_pod_string := obj_to_str(choise_pod)
        -- found the pod, but this can also contain items that serve other orders
        -- delete the contents of the pod_item table because it contains only articles per
        order of the station 1
        .models.frame.pod_Item.deletecontents
        -- find the sum of the items requested by the three stations
        j:=0
        for j1:=1 to OrderAss1.Dim
                j+=1
                batchItem[j] := OrderAss1[j1]
        next
        for j2:=1 to OrderAss2.Dim
                j+=1
                BatchItem[j]:= OrderAss2[j2]
```

```
next
for j3:=1 to OrderAss3.Dim
        j+=1
        BatchItem[j]:= OrderAss3[j3]
next
-- find the new table indexes pod_item
for b:=1 to BatchItem.Dim
        Pod_Item[0,b]:= BatchItem[b]
next
for b:=1 to Pods.Dim
        Pod_item[b,0] := Pods[b]
next
-- find the new contents of the table pod_item
for n:=1 to NowPodContent.XDim
        if NowPodContent[n,0] = Choise_pod_string
                for t:=1 to BatchItem.Dim
                        for m:=1 to NowPodContent.YDim
                                if NowPodContent[0,m] = batchItem[t] and
                                        NowPodContent[n,m] > 0
                                        Pod_item[n,t] := 1
                                        nowpodcontent[n,m] -=1
                                end
                        next
                next
                for m:=1 to pod_Item.ydim
                        if pod_Item[n,m] > 0
                                item := pod_item[0,m]
                                for b:=1 to Orderass1.dim
                                    if item = orderAss1[b]
                                    -- delete the order lines of order of station 1
                                        .models.frame.orderass1.remove(b)
                                        item := void
                                    end
                                next
                                for b:=1 to Orderass2.dim
                                    if item = orderAss2[b]
                                    -- delete the order lines of order of station 2
                                        .models.frame.orderass2.remove(b)
                                        item := void
                                    end
                                next
                                for b:=1 to Orderass3.dim
                                    if item = orderAss3[b]
                                    -- delete the order lines of order of station 3
                                         .models.frame.orderass3.remove(b)
                                         item := void
                                    end
                                next
                        end
                next
        end
next
TableError[1,numpodlist] := choise_pod_string
for n:=1 to pod_item.xdim
        if pod_item[n,0] = choise_pod_string
                TableError[2,numpodlist] := pod_item.sum({n,1}..{n,*})
        end
next
```

```
TableError[6,numpodlist] := orderass1.dim
TableError[7,numpodlist] := orderass2.dim
TableError[8,numpodlist] := orderass3.dim
if TableError[6,numpodlist] = 0 or TableError[7,numpodlist] =0 or
        TableError[8,numpodlist]=0
        TableError[9,numpodlist] := eventcontroller.simTime
end
-- if the order of the station 1 has been concluded I will assign another order to
the station
if orderAss1.empty
        if nownOrders < 150
                NownOrders += 1
                j1:=0
                for i:=1 to BillOfMaterial.YDim
                        if BillOfMaterial[NownOrders,i] = 1
                                j1+=1
                                orderAss1[j1] := BillOfMaterial[0,i]
                        end
                next
        end
end
-- if the order of the station 3 has been concluded I will assign another order to
the station
if orderAss3.empty
        if nownorders < 150
                NownOrders += 1
                j3:=0
                for i:=1 to BillOfMaterial.YDim
                        if BillOfMaterial[NownOrders,i] = 1
                                j3+=1
                                orderAss3[j3] := BillOfMaterial[0,i]
                        end
                next
        end
end
-- if the order of the station 2 has been concluded I will assign another order to
the station
if orderAss2.empty
        if nownorders < 150
                NownOrders += 1
                j2:=0
                for i:=1 to BillOfMaterial.YDim
                        if BillOfMaterial[NownOrders,i] = 1
                                j2+=1
                                orderAss2[j2] := BillOfMaterial[0,i]
                        end
                next
        end
end
.models.frame.pod_Item.deletecontents
.models.frame.batchItem.deletecontents
-- if I have assigned another order to the station it means that for that station
maybe there will be articles that can be taken then control
-- find the sum of the items requested by the three stations
j:=0
for j1:=1 to OrderAss1.Dim
        j+=1
        batchItem[j] := OrderAss1[j1]
```

```
next
for j2:=1 to OrderAss2.Dim
        j+=1
        BatchItem[j]:= OrderAss2[j2]
next
for j3:=1 to OrderAss3.Dim
        j+=1
        BatchItem[j]:= OrderAss3[j3]
next
-- find the new table indexes pod_item
for b:=1 to BatchItem.Dim
        Pod_Item[0,b]:= BatchItem[b]
next
for b:=1 to Pods.Dim
        Pod_item[b,0] := Pods[b]
next
-- find the new contents of the table pod_item
for n:=1 to NowPodContent.XDim
        if NowPodContent[n,0] = Choise_pod_string
                for t:=1 to BatchItem.Dim
                        for m:=1 to NowPodContent.YDim
                                if NowPodContent[0,m] = batchItem[t] and
                                        NowPodContent[n,m] > 0
                                        Pod_item[n,t] := 1
                                        nowpodcontent[n,m] -=1
                                end
                        next
                next
                for m:=1 to pod_Item.ydim
                        if pod_Item[n,m] > 0
                                item := pod_item[0,m]
                                for b:=1 to Orderass1.dim
                                    if item = orderAss1[b]
                                        -- delete the order lines of order of
                                        station 1
                                        .models.frame.orderass1.remove(b)
                                        item := void
                                     end
                                next
                                for b:=1 to Orderass2.dim
                                    if item = orderAss2[b]
                                        -- delete the order lines of order of
                                        station 2
                                        .models.frame.orderass2.remove(b)
                                        item := void
                                    end
                                next
                                for b:=1 to Orderass3.dim
                                    if item = orderAss3[b]
                                        -- delete the order lines of order of
                                        station 3
                                        .models.frame.orderass3.remove(b)
                                        item := void
                                    end
                                next
                        end
                next
        end
```

```
next
-- check again if the station 1 order is not empty after picking up the items
-- otherwise I will assign a new order
if orderAss1.empty
        if nownOrders < 150
                NownOrders += 1
                j1:=0
                for i:=1 to BillOfMaterial.YDim
                        if BillOfMaterial[NownOrders,i] = 1
                                j1+=1
                                orderAss1[j1] := BillOfMaterial[0,i]
                        end
                next
        end
end
for n:=1 to pod_item.xdim
        if pod_item[n,0] = choise_pod_string
                TableError[14,numpodlist] := pod_item.sum({n,1}..{n,*})
        end
next
TableError[6,numpodlist] := orderass1.dim
TableError[7,numpodlist] := orderass2.dim
TableError[8,numpodlist] := orderass3.dim
if TableError[6,numpodlist] = 0 or TableError[7,numpodlist] =0 or
        TableError[8,numpodlist]=0
        TableError[9,numpodlist] := eventcontroller.simTime
end
-- check again if the station 3 order is not empty after picking up the items
-- otherwise I will assign a new order
if orderAss3.empty
        if nownOrders < 150
                NownOrders += 1
                j3:=0
                for i:=1 to BillOfMaterial.YDim
                        if BillOfMaterial[NownOrders,i] = 1
                                j3+=1
                                orderAss3[j3] := BillOfMaterial[0,i]
                        end
                next
        end
end
-- check again if the station 2 order is not empty after picking up the items
-- otherwise I will assign a new order
if orderAss2.empty
        if nownOrders < 150
                NownOrders += 1
                j2:=0
                for i:=1 to BillOfMaterial.YDim
                        if BillOfMaterial[NownOrders,i] = 1
                                j2+=1
                                orderAss2[j2] := BillOfMaterial[0,i]
                        end
                next
        end
end
elseif maxi /= orderass1.dim or orderass1.empty = true
.models.frame.pod_item.deletecontents
.models.frame.sumitem_bj.deletecontents
```

```
.models.frame.batchItem.deletecontents
.models.frame.testPodContent.deletecontents
.models.frame.nowpodcontent.copyrangeTo({0,0}..{*,*},testpodContent,0,0)
-- check if there is a pod that completes the order 2
..
..
-- repeat the commands executed for the station order 1 using order elements 2
..
..
-- if there are no pods that complete the order 2
elseif maxi /= orderass2.dim or orderass2.empty =  true
.models.frame.pod_item.deletecontents
.models.frame.sumitem_bj.deletecontents
.models.frame.batchItem.deletecontents
.models.frame.testPodContent.deletecontents
.models.frame.nowpodcontent.copyrangeTo({0,0}..{*,*},testpodContent,0,0)
-- check if there is a pod that completes the order 3
..
..
-- repeat the commands executed for the station order 1 and 2 using order
elements 3
..
..
--if there are no pods that complete the order 3
elseif maxi /= orderass3.dim or orderass3.empty= true
.models.frame.pod_item.deletecontents
.models.frame.sumitem_bj.deletecontents
.models.frame.batchItem.deletecontents
.models.frame.testPodContent.deletecontents
.models.frame.nowpodcontent.copyrangeTo({0,0}..{*,*},testpodContent,0,0)
-- find the sum of the items requested by the three stations
j:=0
for j1:=1 to OrderAss1.Dim
        j+=1
        batchItem[j] := OrderAss1[j1]
next
for j2:=1 to OrderAss2.Dim
        j+=1
        BatchItem[j]:= OrderAss2[j2]
next
for j3:=1 to OrderAss3.Dim
        j+=1
        BatchItem[j]:= OrderAss3[j3]
next
-- find the table indexes pod_item
for b:=1 to BatchItem.Dim
        Pod_Item[0,b]:= BatchItem[b]
next
for b:=1 to Pods.Dim
        Pod_item[b,0] := Pods[b]
next
-- find the contents of the table pod_item
if batchItem.dim /= 0
        for n:=1 to testPodContent.XDim
                for t:=1 to BatchItem.Dim
                        for m:=1 to testPodContent.YDim
                           if testPodContent[0,m] = batchItem[t] and
                                testPodContent[n,m] > 0
```

```
                              Pod_item[n,t] := 1
                              testpodcontent[n,m] -=1
                         end
                    next
          next
next
-- find the contents of the table sumitem_bj
for n:=1 to pod_item.xdim
          sumItem_bj[0,n] := pod_item[n,0]
          sumItem_bj[1,n] := Pod_item.sum({n,1}..{n,*})
          if sumItem_bj[1,n] = 0
                    sumItem_bj[1,n]:=0.01
          end
next
-- delete the pod not available
numpodlist -= 3
pod := podlist.read(numpodlist)
pod_stringa := obj_to_str(pod)
for n:=1 to sumItem_bj.ydim
          if sumItem_bj[0,n] = pod_stringa
                    sumItem_bj.cutrow(n)
          end
next
numpodlist +=1
pod := podlist.read(numpodlist)
pod_stringa := obj_to_str(pod)
for n:=1 to sumItem_bj.ydim
          if sumItem_bj[0,n] = pod_stringa
                    sumItem_bj[1,n] := 0.01
          end
next
numpodlist +=1
pod := podlist.read(numpodlist)
pod_stringa := obj_to_str(pod)
for n:=1 to sumItem_bj.ydim
          if sumItem_bj[0,n] = pod_stringa
                    sumItem_bj.cutrow(n)
          end
next
numpodlist +=1
--choice pod
maxi := sumItem_bj.max({1,1}..{1,*})
maxi := 0.9 * maxi
for row:=1 to distance.ydim
          if distance[0,row] = pod_str
                    riga:= row
          end
next
for n:=1 to SumItem_bj.YDim
          if sumItem_bj[1,n] >= maxi
                    pod_stringa:= sumItem_bj[0,n]
                    for colomn:=1 to distance.xdim
                              if pod_stringa = distance[colomn,0]
                                        colonna := colomn
                              end
                    next
                    sumItem_bj[2,n] := distance[colonna,riga] /
                    sumItem_bj[1,n]
```

```
                        end
                        if sumItem_bj[1,n] < maxi
                                sumItem_bj[2,n] := 1000
                        end
                next
                mini := sumItem_bj.min({2,1}..{2,*})
                for b:=1 to SumItem_bj.YDim
                        if sumItem_bj[2,b] = mini
                                choise_pod:= sumItem_bj[0,b]
                        end
                next
                Podlist[numpodlist]:=choise_pod
                Choise_pod_string:=obj_to_str(choise_Pod)
                -- delete the order lines that are picked up and then update the number
                of the item in the pod
                for n:=1 to NowPodContent.Xdim
                    if NowPodContent[n,0] = Choise_pod_string
                        for t := 1 to pod_item.ydim
                            if pod_item[n,t] = 1
                                for s:=1 to nowpodcontent.ydim
                                  if nowpodContent[0,s] = pod_item[0,t] and
                                        nowpodContent[n,s]>0
                                        nowpodContent[n,s] -=1
                                  end
                                next
                            end
                        next
                        for m:=1 to Pod_Item.Ydim
                                if Pod_Item[n,m] > 0
                                        item := pod_Item[0,m]
                                        for b:=1 to Orderass1.dim
                                                if item = orderAss1[b]
                                        -- delete the order lines of order of station 1
                                                .models.frame.orderass1.remove(b)
                                                item := void
                                                end
                                        next
                                        for b:=1 to Orderass2.dim
                                                if item = orderAss2[b]
                                        -- delete the order lines of order of station 2
                                                .models.frame.orderass2.remove(b)
                                                item := void
                                                end
                                        next
                                        for b:=1 to Orderass3.dim
                                                if item = orderAss3[b]
                                        -- delete the order lines of order of station 3
                                                .models.frame.orderass3.remove(b)
                                                item := void
                                                end
                                        next
                                end
                        next
                    end
                next
            end
        end
end
```

```
          end
```

A.8. Station sensor method

```
param SensorID: integer, Front: boolean

var pod,pod_:object
var pod_string:string
var a,x,y,b,i,n :integer
var item: object
var item_obj:object
var item_class: object
var num,succ:integer
var item_string : string
var times : time

@.stopped:=true
-- the initial part of the code serves to unlock the robots that are in the waiting area of the station
and insert them in the operator's queue correctly
pod:= @.cont
pod_string:=obj_to_str(pod)
for n:=1 to podlist.dim
          if pod = podlist[n]
                    num:= n
                    num += 1
                    succ := num
          end
next
pod_ := podlist.read(succ)
if track139.cont /= void
          if track139.cont.cont = pod_
                    track139.cont.stopped := false
                    track139.cont.destination := track95
                    numpod +=1
                    TableError[3,numpod]:= pod_.numMU
          end
end
if track135.cont /= void
          if track135.cont.cont = pod_
                    track135.cont.stopped := false
                    track135.cont.destination := track95
                    numpod +=1
                    TableError[3,numpod]:= pod_.numMU
          end
end
if track136.cont /= void
          if track136.cont.cont = pod_
                    track136.cont.stopped := false
                    track136.cont.destination := track95
                    numpod +=1
                    TableError[3,numpod]:= pod_.numMU
          end
end
if track137.cont /= void
          if track137.cont.cont = pod_
                    track137.cont.stopped := false
                    track137.cont.destination := track95
                    numpod +=1
                    TableError[3,numpod]:= pod_.numMU
          end
end
if track145.cont /= void
          if track145.cont.cont = pod_
                    track145.cont.stopped := false
                    track145.cont.destination := track95
                    numpod +=1
                    TableError[3,numpod]:= pod_.numMU
          end
end
-- manual discharge
pod:= @.cont
numpod2 +=1
TimeWorker[1,numpod2] := eventcontroller.simtime
num := TableError[2,numpod2] + TableError[14,numpod2]
timeworker[2,numpod2] := num
timeworker[3,numpod2] := timeworker[2,numpod2]*0:05.5000
timeworker[4,numpod2] := timeworker[1,numpod2] + timeworker[3,numpod2]
timeworker[5,numpod2] := timeworker[4,numpod2]-timeworker[1,numpod2]
for y:=1 to OrderInStation.YDim
          for x:=1 to OrderInStation.XDim
                    if OrderInStation[x,y] = 1
                              item := OrderinStation[x,0]
                              for b:= pod.numMU downto 1
                                        item_obj:=pod.MU(b)
                                        item_class :=item_obj.class
                                        item_string := obj_to_str(item_class)
                                        if item_string = obj_to_str(item)
                                                  item_obj.move(BufferAssembly)
                                                  item := void
                                                  stopuntil Bufferassembly.numMu=0 prio 1
                                                  --exitloop
                                        end
                              next
                    end
          next
next
track95.startpause(3.5)
@.stopped:=false
```

References

Andriolo Alessandro, Battini Daria, Calzavara Martina, Gamberi Mauro, Peretti Umberto, Persona Alessandro, Pilati Francesco, Sgarbossa Fabio, "New pick-to.light system configuration: a feasibility study", XVIII Summer School Francesco Turco - Industrial Mechanical Plants

Bartholdi John, Hackman Steven T. (2011) "Warehouse and distribution science", ReseachGate

Battini D. (2018), notes from lectures of the course "Logistica Industriale", Mechanical Engineering, Università degli studi di Padova

Battini Dari, Calzavara Martina, Alessandro Persona, Fabio Sgarbossa, (2014) "A comparative analysis of different paperless picking systems", Industrial Management & Data Systems, 115 (3), 483-503

Battini Dari, Calzavara Martina, Alessandro Persona, Roncari Manuel, Fabio Sgarbossa, "Dual-tray vertical lift module for order picking: a performance and storage assignment preliminary study", XX Summer School Francesco Turco - Industrial Systems Engineering

Behram Bahrami, El-Houssaine Aghezzaf, Veronique Limere (2017) "Using simulation to analyze picker blocking in manual order picking systems", Procedia Manufacturing, 11, 1798-1808

Bipan Zou, Yeming Gong, Xianhao Xu, Zhe Yuan (2017) "Assignment rules in robotic mobile fulfilment systems for online retailers", International Journal of Production Research

Boysen Nils, Briskorn Dirk, Emde Simon, (2016) " Parts-to-picker based order processing in a rack-moving mobile robots environment", European Journal of Operational Research

Bozer Yavuz A., Francisco J.Aldarondo, (2018) "A simulation-based comparison of two goods-to-person order picking systems in an online retail setting", International Journal of Production Research, 56 (11), 3838-3858

Dallari Fabrizio, Marchet Gino, Melacini Marco, (2008) "design of order picking system", Springer, 42, 1-12

D'Andrea Raffaello, Wurman Peter, (2008) "Future challenges of coordinating hundreds of autonomous vehicles in distribution facilities", IEEE

De Koster R., VAn der Poort E. (1998) "Routing orderpickers in a warehouse between optimal and heuristics solutions", IIE, 30, 469-480

De Koster René, Le-Duc Tho, Roodbergen Kees Jan (2006) "Design and control of warehouse order picking: A literature review", European Journal of Operational Research, 182 (2007), 481-501

De Koster René, Le-Duc Tho, Nima Zaerpour (2012) "Determing the number of zones in a pick-and-sort order picking system", International Journal of Production Research

Enright John J., Wurman Peter R., (2009) "Optimization and coordinated autonomy in mobile fulfillment systems", AAAI

Guizzo Erico, (2008) "Kiva Systems: Three Engineers, Hundreds of Robots, One Warehouse", IEEE

Horvat Matic (2012) "An approach to order picking optimization warehouse", SemanticScholar

Lamballais T., Roy D., De Koster M.B.M. (2015) "Estimating performance in a robotic mobile fulfillment system", International Journal of Operational Research

Lamballais T., Roy D., De Koster M.B.M. (2017) "Inventory allocation in robotic mobile fulfilment systems"

Li Jun-tao, Liu Hong-jian, (2016) "Design optimization of Amazon robotics", Science publishing group, 4 (2), 48-52

Li Z. P., Zhang J. L., Zhang H.J., Hua G.W. (2017) "Optimal selection of movable shelves under cargo-to-person picking mode", SemanticScholar

Lienert Thomas, Staab Tobias, Ludwig Christopher, Fottner Johannes (2017) "Simulation-based performance analysis in robotic mobile fulfilment systems"

Merschformann M., Lamballais T., De Koster M.B.M., Suhl L. (2018) " Decision Rules for robotic mobile fulfillment systems", AAAI

Merschformann Marius, Xie Lie, Erdmann Daniel (2017) "Path planning for robotoc mobile fulfillment systems", AAAI

Merschformann M., Xie L., Li H. (2018) "RAWSim-O: A simulation framework for robotic mobile fulfilment systems", BVL

Pareschi A., Ferrari F., Persona A., Regattieri A., "Logistica integrata e flessibile"

Petersen Charles G., Aase Gerald R. (2003), "A comparison of picking, storage and routing policies in manual order picking", International Journal of Production Economics, 92 (2004), 11-19

Petersen Charles G., Aase Gerald R. (2016) "Improving order picking efficiency with the use of cross aisles and storage policies", Open Journal of Business and Management, 5 (1)

Poudel Dev Bahadur, (2013) "Coordinating hundreds of cooperative, autonomous robots in warehouse", AAAI

Riccardo Manzini, Mauro Gamberi, Alessandro Persona, Alberto Regattieri (2006), "Design of a class based storage picker to product order picking system", Springer

Sgarbossa F., notes from lectures of the course " Impianti industriali", Mechanical Engineering, Università degli studi di padova

Tompkins James, White John, Bozer Yazur, Tanchoco J.M.A. (2010) "Facilities Planning", John Wiley & Sons

Wurman Peter R., D'Andrea Raffaello, Mountz Mick, (2007) "Coordinating Hundreds of cooperative, autonomous vehicles in warehouse", Association for the Advancement of Artificial Intelligence (AAAI)

www.mwpvl.com, "Is Kiva system a good fit for your distribution center? An unbiased distribution consultant evaluation"

www.roboticstomorrow.com, "How kiva systems and warehouse management systems interact"