FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Task Scheduling for Multiple Robots in an Industrial Environment

Vítor Emanuel Santos Lousas Alves Mota

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. Dr. Paulo José Cerqueira Gomes da Costa

Second Supervisor: Prof. Dr. José Luís Lima

October 30, 2015

U. PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES | **2014/2015**

A Dissertação intitulada

"Task Scheduling for Multiples Robots in an Industrial Environment"

foi aprovada em provas realizadas em 16-10-2015

o júri

x [illegible]

Presidente Professor Doutor António Pedro Rodrigues Aguiar
Professor Associado do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

[illegible]

Professor Doutor José Alexandre Carvalho Gonçalves
Professor Adjunto do Departamento Eletrotecnia da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Bragança

[illegible]

Professor Doutor Paulo José Cerqueira Gomes da Costa
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extratos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.

x [illegible]

Autor - Vítor Emanuel dos Santos Lousas Alves da Mota

Faculdade de Engenharia da Universidade do Porto

Abstract

AGVs are an upcoming tool in the factories and warehouses substituting workers and the traditional conveyor belt lines, as they are more flexible and profitable solutions. As their number expand, the problem of optimizing the scheduling of their task becomes more and more relevant, as a good optimization translates in savings of both time and money.

This thesis will study the problem of Scheduling tasks for multiple robots in real time, using the Robot@Factory competition as a case study. It's aim is to provide a conflict free scheduling aiming to minimize the makespan time.

Before addressing the problem of the scheduling there's another problem that must be solved, how to deal with tasks that have indeterministic times. In order for the scheduling to have information of when a part is done being machined, or which machines are free, busy or malfunctioning a monitoring system was implemented. Besides providing the information for the scheduling, it has a visual interface which comes in handy for the competition, providing the needed information for the jury to evaluate the rounds of the factory competition.

Instead of allowing robots to move freely through the field, their movement was restricted to certain paths. Although this makes routing slightly less optimal, it decreases the complexity of the problem, making the routing faster. Using said design path, the routing was implemented using an A^* algorithm.

Reaching for the closest to optimal results, the schedule and routing were combined in a A^*, so that the algorithm can track the position of every robot minimizing the cases where two robots get to the same intersection at the same time, as one of them would have to wait.

Although it returns a good solution it may not completely eliminate the possibility of a deadlock, and so it is always a good idea to implement a solution to prevent it, as for example wheel spin can cause delays that accumulate so that two robots may get to the same intersection simultaneously. This is accomplish with zone control, and restricting the access to the intersections to one robot at a time, and only if the segment is travelling to is free.

Finally to navigate between the nodes of the path design a simple follow line was implemented, so the robot navigates in a straight line between two nodes connected nodes.

Agradecimentos

Ao Prof. Doutor Paulo Costa e ap Prof. Doutor José Lima pelo apoio durante a dissertação.

Ao Daniel Campos e Nuno Moreira, pelo companheirismos e amizade, bem como a todos os meus colegas que me acompanharam durante este meu percurso.

E em especial à Cristina Serra pelo apoio incondicional.

Vítor Mota

*"Engineers like to solve problems.
If there are no problems handily available, they will create their own problems."*

Scott Adams

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
AI	Artificial Intelligence
AGV	Automated Guided Vehicle
B&B	Branch and Bound
FMS	Flexible Manufacturing System
FCFS	First come first served
HMI	Human-Machine Interface
NN	Neural Network
P&D	Pick & Delivery
R@F	Robot@Factory

Chapter 1

Introduction

The present document describes all the work carried out in the Dissertation, final project of Master Degree in Electrical and Computer Engineering.

Throughout this chapter, the following topics will be discussed: Context and Motivation1.1 as well as the Document Overview 1.2.

1.1 Context and Motivation

AGVs were first introduced in the industry at Volvo's plant in Kalmar, Sweden in 1974, and since then the popularity of AGVs has grown considerably. More than 15,000 AGVs were in use within little more than a decade thereafter [1]. The number of AGVs keeps on rapidly increasing, as its usage spread through a variety of industries, ranging from manufacturing plants, warehouses, container terminals, and even hospitals, due to being flexible, highly scalable, good space utilization, improved factory floor safety, reduction in overall operating cost, and easier interface with other automated systems [2].

For example Kiva [3, 4] has redesigned the warehouses, substituting the conventional conveyor belt lines and workers picking the cargo themselves with AGVs that will carry the whole shelf of worker as we can see in the figure 1.1.

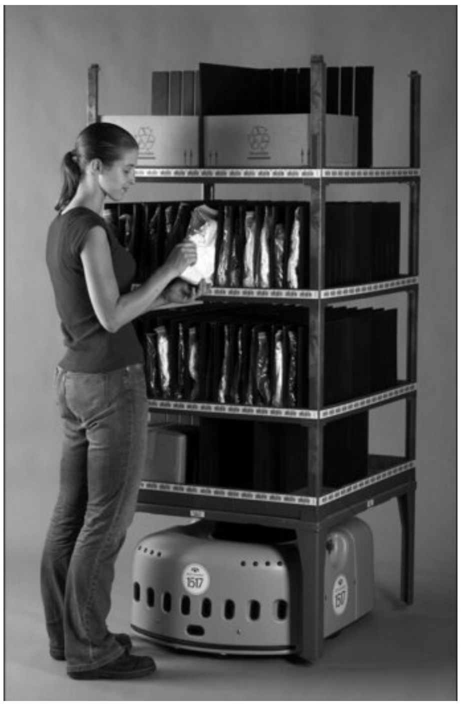


Figure 1.1: A Kiva Drive Unit and Storage Pod [4].

This method minimizes the time it takes getting an item from the shelf, and sending it out of the warehouse, as well as minimizing warehouse costs, since AGVs do not use up as many energy as big conveyor belts. Also as the workers only inhabit a small part of the warehouse, there is no need to climate control the whole space.

There are many benefits from having AGVs in industrial environments, so it can be expected to see their number grow, and as they do the problem of finding good scheduling solutions their tasks gets more and more relevant to real manufacturing systems considering that production rate and production costs are very dependent on the schedules used for controlling the work flow through the system [5].

1.2 Document Overview

Other than the Introduction, this dissertation contains 8 more chapters.

In chapter 2 the scheduling problem will be further explain and related to the Robot@Factory competition.

Chapter 3 is dedicated to the bibliographical review.

Chapter 4 features all the platforms used to develop this dissertation.

Chapter 5 shows the improvement done to the software of the field of the Robot@Factory competition, which is also used to retrieve information for the Scheduling.

Chapter 6 presents scheduling solution for both single and multiple AGV scheduling in an flexible manufacturing system.

Chapter 7 is dedicated to the results obtained from the implementations of the work described in the previous chapters.

At last, chapter 8 presents the conclusions and future work.

Chapter 2

Explaining the Problem

This chapter features a brief explanation of the robotics competition, Robot@Factory (R@F), as well as this thesis relationship to it.

AGVs are an upcoming tool in the industries and warehouses, specially when it comes to transporting material from point A to point B. They are replacing workers as well as the conventional conveyor belt, because they move inventory faster, more efficiently and in a more flexible way.

The R@F is a competition that presents the unique opportunity of testing out, in a small scale project, solutions that could be implemented in a industry. It encourages teams to try newer approaches by having them compete with teams from all around the world.

2.1 Robot@Factory

The R@F presents the challenge of building and programming a pick & deliver autonomously guided vehicle, that can navigate through the field of the competition (the factory, figure 2.1), pick the cargo and deliver it to the right location [6].

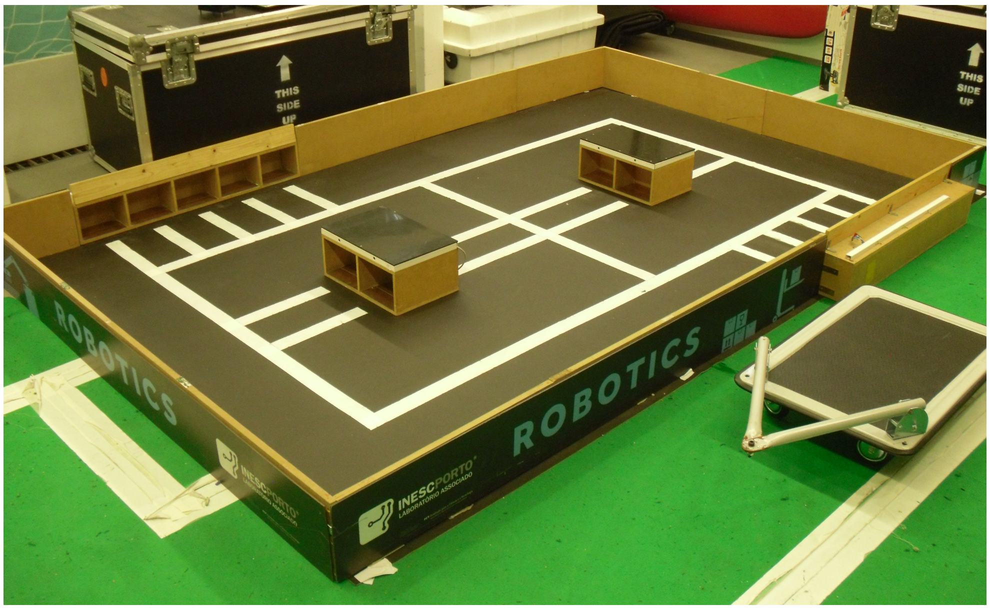


Figure 2.1: Robot@Factory's field

To accomplish this, robots must be able to self-localize, detect obstacles, navigate avoiding collisions with walls, obstacles or other robots, recognise the type of cargo, and decide the order in which they will transport the cargo.

2.2 Task Scheduling at Robot@Factory

There are 3 kinds of parts, identified by a LED. Depending on which type of part, it needs to be transported to a different place as we can see in table 2.1.

Table 2.1: Destination of the different parts and end result of the machining

Parts	Destinations	Colour after machining
Red	Machine A	Green
Green	Machine B	Blue
Blue	Outgoing Warehouse	-

The machining process takes different machining times depending on the machine and these times are unknown until the competition, so for a robot to execute the pick & deliver jobs in the best possible order, it needs an algorithm that can determine that order upon knowing which are the parts in the incoming warehouse, otherwise teams have to calculate beforehand the best case for each combination of 5 parts.

Robots should have a computer vision systems so that they can identify the color of the LED in each part. Knowing the color of each part's led, it has a list of jobs to schedule.

We can optimize the time it takes to execute all tasks, i.e. to move all 5 parts from the incoming warehouse to the outgoing warehouse. To accomplish this, the algorithm should minimize both the time it waits for a part to be ready, i.e. it should be executing other task while it waits for a part to get ready, as well as minimizing distance it travels without part, since for a certain pick and drop locations the shortest root is a constant, but the shortest root from the robot current position to the pick location depends where it finished its last tasks.

2.3 Summary

The objective of this thesis is to assign jobs to AGVs, minimizing the makespan, i.e. minimizing the duration between the beginning of the first job and the conclusion of the last job. The work done is going to developed on the following assumptions:

- Each machine has 4 stations and each operates only one product at a time;
- Each product can only be operated by one type of machine;
- An AGV carries only one product at a time;
- The machining and travel times are deterministic and known in advance;

Chapter 3

Bibliographical Review

The task scheduling is a complex problem which can be decomposed in the several steps discussed throughout this chapter.

3.1 Path Design

There are different types of robot navigation, with different levels of flexibility which affects how the scheduling algorithms are implemented.

Fixed path Guidance

The robot follows fixed tracks, for example rails systems. It's a robust system but lacks flexibility.

Wire/Magnetic Guidance

The AGV follows the electrical wire/magnetic strip, with the similar restrictions and advantages to the fixed path guidance.

Non-wire guidance

It's the more flexible option as the map is programmed into the robot's controlling system. It can still be aided by external maps as the case where a robot with an optical sensor follows a stripe painted on the floor, or it can, for example, focus on the map programmed and locate itself with a laser beam.

3.1.1 Flow Topology

Throughout this section will be shown different types of flow topology, giving a brief comment on them along with some diagrams.

Single Line

The simplest flow topology is the single line system as seen on figure 3.1. It's a very simple system, but it's not much more flexible than a conveyor belt.

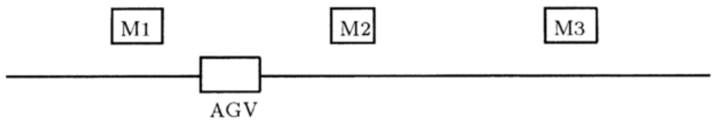


Figure 3.1: Single Line AGV System [7]

Single-Loop

The Single-loop is a slightly more flexible solution than the single line, having the advantage of being able to choose from two paths, to get from point A to point B, either clockwise or counter-clockwise which will facilitate having more than one AGV working simultaneously.

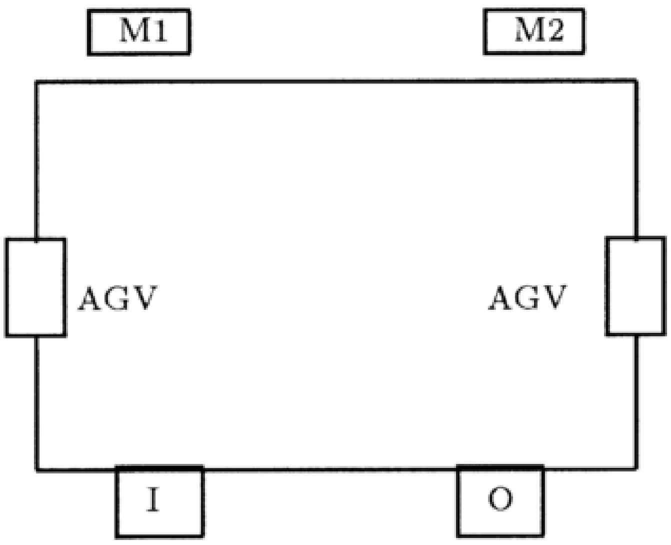


Figure 3.2: Single-loop AGV System [7]

Ladder

A ladder type system is significantly more complex than the single loop, but has also more flexibility. However, the deadlock becomes a greater worry as, unlike the previous system, it has paths that cross each other.

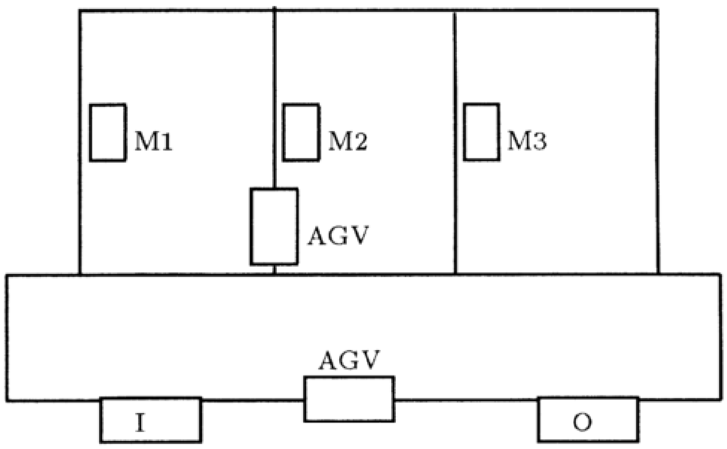


Figure 3.3: Ladder type AGV System [7]

Complex Network

When a network is considered, the problem becomes more complex because of the availability of many routes for AGV dispatching, and the possibility of conflicts between AGVs.

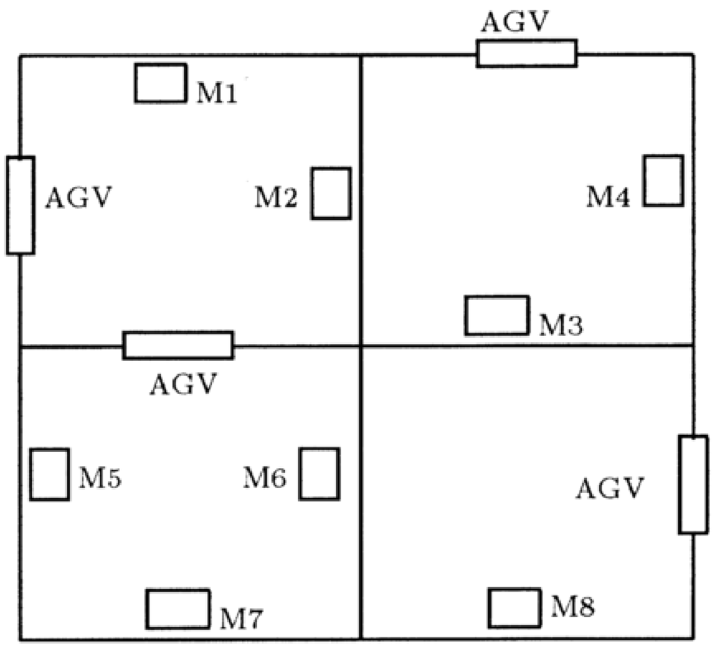


Figure 3.4: Complex AGV networking System [7]

Tandem

The tandem type flow design is less used due to its mechanical complexity and thus more expensive, as it has the added costs of building the transfer station. Even thought it's not as flexible as other solutions it offers simple control.

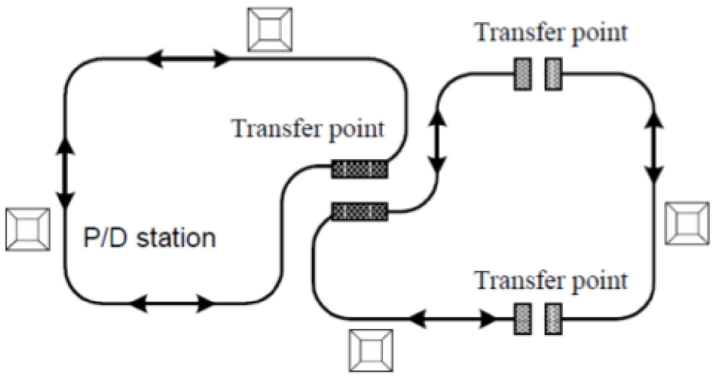


Figure 3.5: Tandem AGV System [8]

3.1.2 Flow Direction

The flow direction is the direction in which the AGV travels along a certain segment of the flow path. The flow direction can be unidirectional or bidirectional. An unidirectional mode is frequently used for easier control[7], but a bidirectional flow is more flexible.

3.1.3 Number of Parallel lanes

For any flow path design there's the option of having multiple parallel lines instead of a single line as shown in the examples above. Having multiple parallel lines enables to have more AGVs cruising the factory but also increases expenses, as there will be a need to have a larger facility. Another drawback is that the route between P&D locations will be slightly longer.

3.2 Scheduling System

Scheduling Systems involves both schedule of AGV's and machines, as well as the management of resources available [7].

3.2.1 Search Algorithms

In terms of computer science, a search algorithm refers to an algorithm design to find an item with specified properties among a collection of items. Items which may either be stored individually as records in a database or be elements of a search space defined by a mathematical formula or procedure, such as the roots of an equation with integer variables; or a combination of the two, such as the Hamiltonian circuits of a graph [9].

AGVs routing and scheduling can be performed either on-line or off-line. Off-line planning requires the knowledge of all tasks prior to execution of an algorithm, while on-line planning allows new tasks to appear although the planning has been already done, re-planing to calculate a better schedule, taking in consideration the new information. It also accommodates delays in multi AGV systems, and recalculates after each delivery [10].

3.2.1.1 Branch&Bound

Since there are a finite number of combinations that we can go through and compare until we reach the correct one, we could use this algorithm, just as seen on [11, 12], in order to get at least a close to optimal solution, in case we stop the algorithm at the very first solution. Otherwise, continuing to iterate will confirm if the found solution is indeed the optimal one, or will keep finding better ones until the optimal solution is reached, i.e. there are no more nodes with a lesser cost than the best solution found.

The algorithm starts from the root node and adds its children to the open nodes list. From this list it chooses the node with less cost and if it is not a final node, i.e. if a solution has not been reached, it adds this node's children to the open list, then proceeds to, once again, pick the node with less cost from the list and so on until a solution is reached. If we prefer a faster algorithm to an optimal one, we would stop it here and we would have gotten a solution pretty close to optimal. Otherwise, we would compare the solution with the open list and continue to iterate until our solution has a smaller cost than any of the nodes in the open list [13].

When scaling this thesis challenge to an industrial environment, there would be many more tasks to schedule as well as AGVs to assign the tasks to. When applying this algorithm to this situation, as the quantity of the variables grow, the number of nodes would grow exponentially with them, which implies a longer processing time for the algorithm.

3.2.1.2 A*

This algorithm is pretty close to the one previously presented, being its only difference the use of an heuristic as well as the current cost, to get an estimation better or equal to the best case from that node on. Using this, some backtracking is avoided, making the algorithms faster than the Branch&Bound.

An heuristic is the difference from a sub-estimated total cost minus the current cost. In the problem at hands, the heuristic would be an estimation of the time in which the robot is carrying a part, since that time is fixed with a small variation, unlike the time in which the robot is not carrying a part which is unknown and the objective to minimize. The A*'s heuristic is better when its value is closest to the real cost between the current node and the final node without overstepping it [13].

3.2.2 Neural Network

Neural Networks (NN) are computer models capable of recognising patterns, and find a mathematical model closest to the distribution of the results obtained so far, and using that model, to determine the result of the points to add next.

NNs are also a possibility, as we can see in figure 3.6 taken from [14], the NN is structured in a way that you enter the coordinates x and y of the points in which the task starts and ends, i.e. the point where the robot picks and delivers the parts. As it iterates through all the jobs, it recalculates the weights, for each node. The algorithm starts with one output node, and as you iterate it adds another node to the output layer. Once you go through all the jobs you will have the same number of nodes in the output layer as jobs. The winner selection is obtained by comparing, for each job, its coordinates with the weights w between the input and output layer. This will lead us to an organized list of tasks with a certain level of optimization.

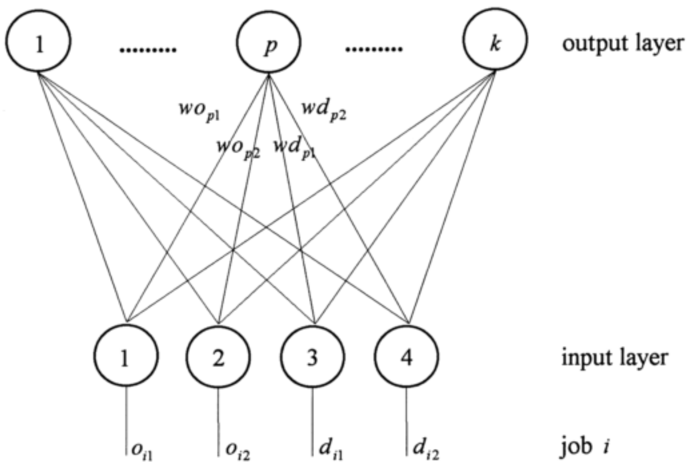


Figure 3.6: Structure of self-organizing map

3.2.3 Genetic Algorithms

Genetic Algorithms (GA) [15, 16] are a great alternative to Search Algorithms when one is trying to minimize the processing time. Instead of going through the nodes in the tree, it creates some solutions (chromosomes), either randomly or based on some characteristic that would benefit the cost of the solution, for example create several chromosomes where the first parts to be picked up are green or red, since they need to be machined, and while they are doing that the AGV would do other jobs. After having a reasonable amount of chromosomes we need to check if it is a valid solution as they were at least partially random. The algorithm calculates the total cost of each chromosome. Inspired in nature, those who are better adapted (i.e. have lower cost) have a better change to generate offspring, which in this algorithm means being picked to go through mutations and crossover. In hopes of achieving better solutions it uses crossover, this is done by splitting both chromosomes, and mixing them to generate two new solutions. These two new chromosomes then have a certain chance of suffering mutation, if this occurs, one task in this solution will be changing, which might lead to better or worse results. Lastly we check if new chromosomes are valid solutions and calculate. Using an elitist selection, we may discard the newer chromosomes and keep the old ones, in case the new ones are less fit. We continue to iterate until we have an optimized solution. The more we iterate, the better is the odd of having a closer to optimal solution, but obviously is going to take more time. So when implementing this algorithm one should study how much iterations are needed [13].

3.2.4 Dispatching Rules

In a flexible manufacturing system (FMS) it's typical to have free AGVs while machines are busy, so in this case it makes sense to use dispatching rules such as shortest time dispatching rule (SPT) or minimum slack time rule (SLACK). Although dispatching rules don't get an optimal result, the results are very satisfactory and are very easy to implement and scale.

3.2.5 Auction biding system

Auction biding system or Combinatorial Auctions are systems where each vehicle is represented by an intelligent agent that bids for task and plans its own schedule. The auctioneer has the objective of minimizing transportation tardiness. It has the advantages associated with a decentralized model, such as faster information processing, achieved sharing the computational burden by multiple entities, higher system reliability by eliminating the risk of a single-node system failure, and better system flexibility and scalability.

However, it is apparent that the current technical knowledge of distributed transportation system is still far from convincing in order for it to completely replace the centralized system, particularly from the performance point of view due to decisions being made on the basis of localized information [2].

A Combinatorial Auctions for multiple loading capacity AGV was proposed by M. Fauadi et al [2] to shorten the performance gap between decentralized and centralized scheduling systems. This was accomplish with a multilateral CA to enable multiple auctioneer and multiple bidder communication to take place. The bidders concurrently determine an appropriate combination of tasks that they should transport and submit multiple packages, each consisting of one or more distinct tasks.

3.3 Routing

Routing can be a complex problem, but in FMS there is a tendency to keep it simple and having the AGV's follow pre-planned paths, rather than calculating the best path using search algorithms such as A* or *Dijkstra*.

3.4 Deadlock Avoidance

3.4.1 Zone Control

The guide paths are divided in several zones representing workstations, intersections of several paths or simple parts of a straight lane, and the admittance of a vehicle to any zone must be previously authorized by that zone's controller, which only allows access after checking that collisions and deadlocks are avoided [17, 18].

3.4.2 Time Windows based Dynamic Routing

Using dynamic routing the problem is treated as a time-space problem instead of static routing which obtains solutions only in the space domain. It works by attributing weights in the arcs of the graph of the factory, by dividing the length of the arc by the speed of the robot, obtaining an approximation of the time it takes the robot to travel through that arc. This algorithm starts by calculating the shortest path, and then checking each arc to see if the paths calculated are disable. It does this with time windows insertions, for each arc constructs time windows representing the moments in time the arc is occupied by a robot. If they overlap, meaning there are two robots simultaneously in the same arc, the algorithm iteratively reinserts time windows until conflicts disappear. This algorithm has been successfully implemented in several European factories [18].

3.4.3 Banker's Algorithm

Banker's Algorithm [10], as the name suggests, can be used for money allocation at a bank. Any process that enters the system, must declare the maximum number of instances (both number of AGVs and number of each type of machines) needed. This maximum declared cannot exceed the total number of resources in the system. The process can only be executed if there are available resources, otherwise the process must wait, so for this algorithm to be able to be successfully implemented at any given time there must be an AGV that can compete its job without any movement from the other vehicles. For the Banker's algorithm to work, it needs to know three things:

1. The maximum number of each resource that each process can request
2. How much of each resource are the current processes holding
3. How much of each resource the system has available

3.5 Estimating the number of AGV's

Increasing the number of AGVs can decrease the overall transportation times, but overdoing it can cause excessive traffic and slow down operations. To evaluate the ideal number of vehicles one must resort to simulations of the workspace conditions. Having a good model of the motors used in the AGVs is key to getting a good simulation [**?**]. On the other hand if the factory/warehouse is aiming for a prespecified throughput rate, then this becomes a minimization problem, to discover the minimal amount of robots needed to assure that rate [7].

3.6 Battery Management

Although this is not a concert in the problem presented, this is a key factor in scheduling AGVs. There are two alternatives to battery management, either the robot gets to the charging station and replaces the battery, or it recharges the battery it is currently holding. The first solution is much more efficient as the AGV stops for a shorter period, but is also more mechanically complex compared to the later where the robot just needs to connect its charging terminals to the charging station.

There are several options regarding the scheduling of AGVs trips to the charging station, such as:

Opportunity Charging

The AGV will take advantage of breaks (moments when it has no orders assigned) to charge its battery.

Automatic Charging

The AGV keeps working until the battery level drops bellow a certain threshold, and at this point the AGV will, after completing the current job, head to the charging station.

Combination System

As the name indicates this is a combination of the Opportunity charging and the Automatic charging. It will take advantages of brakes to charge its battery, but if the battery level drops bellow the assigned threshold it will finish the current job and immediately head to the charging station.

Chapter 4

Implemented platforms

This chapter explains the work done, related to the factory competition, as part of FEUP Factory team consisted at the time of two other students and a professor.

4.1 3D Models

Both the robot used in the *Robot@Factory's* competition, as the field of the competition itself, were projected using the solid modelling CAD (Computer-aided design), *SolidWorks*.

4.1.1 The field

As the field of the competition needed to be restored, it was asked of the refereed team to estimate the cost of building a new one. To avoid miscalculating the wood cuts for the built, the field was first sketched as seen on 4.4. Having a 3D sketch up of the field also eases the job of other teams trying to replicate the field, in order to practice for the competition beforehand

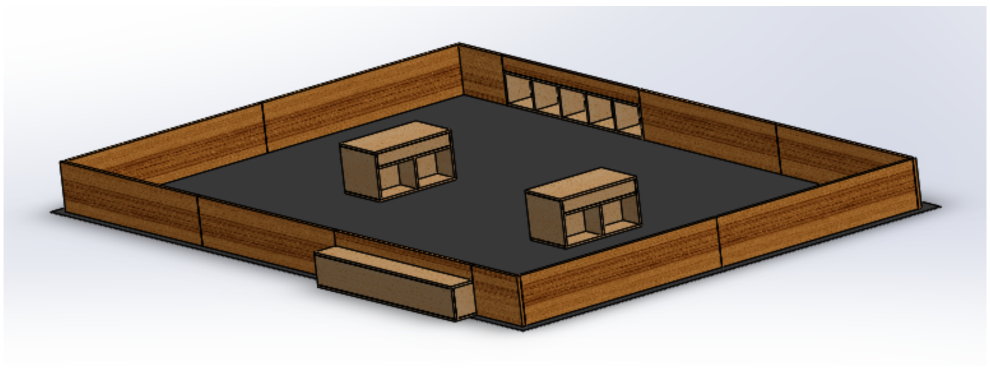


Figure 4.1: 3D Model of Robot@Factory's field

Along with the 3D sketch of field it was also produced a detailed budget, with a list of parts required for the build, from materials required for the structure of the field to all the electronic hardware required for the parts to interact with both the competing AGVs and the warehouses and machines.

Unfortunately as the budget was decline, efforts were focus on the restoration of the current field, this included strengthening the structure of the field but also the improvement of the software as explained on chapter 5.

4.1.2 The robot

In order to have a better understanding of where every component in our robot should go, and thus having clearer picture of how the end result would look like before construction start, it was decided to first sketched it in a 3D environment as shown on figure 4.2, using SolidWorks. Doing this prevents re-drilling holes in the chassis, due to miscalculations. Also having a 3D design of the robot, which is constituted by assembling every piece in a single sketch, there is the benefit that, in case of emergency, there is always the option of 3D printing a required part.

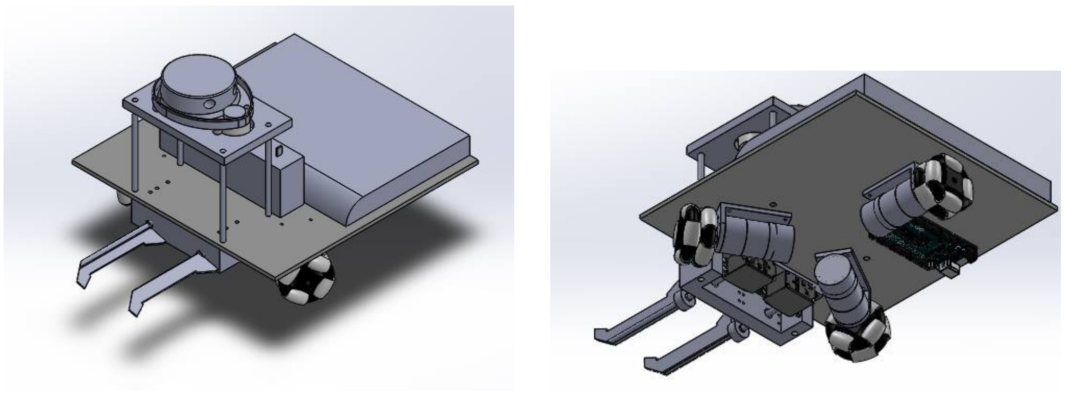
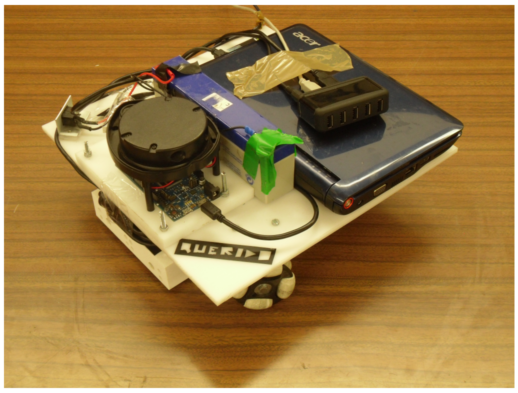

(a) Top view (b) Bottom view

Figure 4.2: 3D sketch of the robot

4.2 Construction

A three wheeled omnidirectional robot, was built. The choice of three wheels over a four wheeled solution was made to facilitate the construction, as a four wheeled robot is mechanically more challenging to build as it the needs suspension. The decision of having an omnidirectional vehicle was a bet in manoeuvrability. Although it would be more challenging to control, the extra manoeuvrability would facilitate the job of aligning the robot with the P&D locations, and this is really something worth considering as the pick and delivery is a critical part of the rounds having a high potential to fail.


(a) Top view

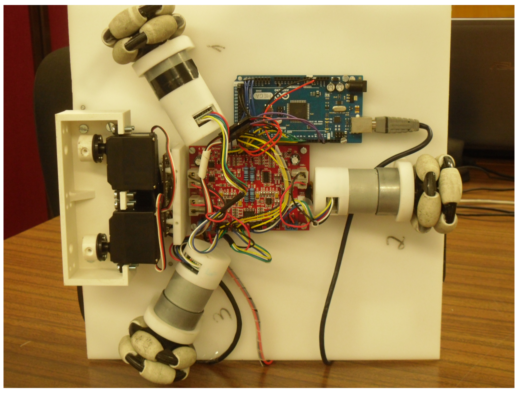

(b) Bottom view

Figure 4.3: Robot built to participate in the competition

The chassis of the robot was built with a 30cm by 30cm by 0.5cm board of polyacetal, which is a sturdy non-conductive material.

For the 3 wheels it was chosen the *omnidirectional wheel* from *Bot'n Roll* which are moved by 3 *30:1 Metal Gearmotor 37Dx52L mm with 64 CPR Encoder* that are in turn controlled by a *Rover 5 Motor Driver Board*, having an *Arduino Mega* serving as interface between it and the software in an *acer aspire one* computer. The *Arduino Mega* also controls the *Futaba S3003* servo motors which actuate the 3D printed claws designed to pick up parts in the Robot@Factory competition.

For the laser range finder was used the *Lidar* taken out of a *Neato xV-11* vacuum cleaning robot. An *Arduino Due* was used to control the speed at which the laser spins. This control needs to be precise because if the frequency of the rotation is off, the results obtained won't be accurate. Another use for the *Arduino* was to serve as an interface between the laser and the for-mention computer, which would be running the locating, routing, scheduling and part color identification algorithms.

To detect the color of the parts we used the *PlayStation Eye* camera, connected to the computer via *usb*.

4.3 Simulation Environment

In order to train for the competition without having access to the *Robot@Factory's* field, a simulation version of our robot was created in *simtwo* taking advantage of the already available simulation of the field.

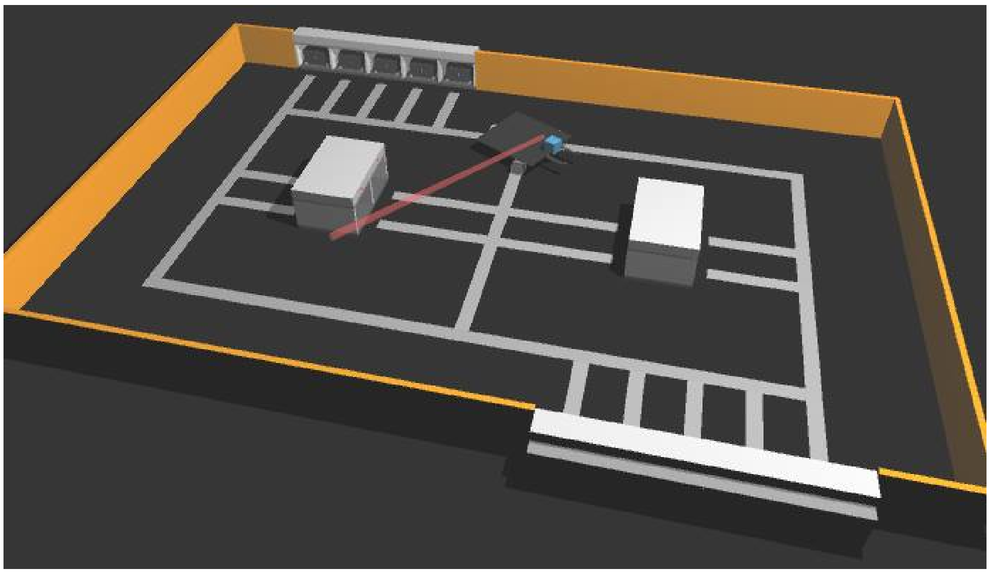


Figure 4.4: SimTwo

Chapter 5

Monitoring System

With the increase of automation in factories and warehouses, arises the need to collect more data from the machinery. This data was many porpoises, from giving workers a constant feedback of what's happening in the facility, what jobs have been completed and give warnings when something goes wrong, to create statistics like throughput rate and machine usage, but also, and this is its the main porpoise regarding the objectives of this thesis, update an eventual schedule program with the current of the status shop floor.

The team Feup@Factory was constituted by myself, as well as two other students, Daniel Campos and Nuno Moreira, who since then have finished their Master's thesis, and lead by Prof. Paulo Costa. As well as entering the Robot@Factory competition this team was responsible for the maintenance of all the hardware and software involved in the field of the competition. The team's responsibilities ranged from reinforcing the fragile wooden structure of the field to planning the construction of a new one, and were also proposed the task of updating the software used by the judges.

Since the information required for the jury aiding software and the monitoring system is the same, it was decided to create a program to satisfy both criteria, so a system was created to collect the needed information for the scheduling program having a visual interface that satisfies the needs of the juries if the R@F competition.

As the monitoring system as a judges software is more complex as it needs to read and also set data, this chapter will focus more on the point of view of the judging of the R@F competition.

5.1 Objectives

From the Scheduling point of view the purpose of this system is to retrieve the current location of parts, state of the machines and alert when a part is finished being machined. Furthermore it could be used to get the information of what type of part is in each of the slots in the Incoming Warehouse, but that would defeat of the proposes of the competition which is to test teams solutions to detect the color of the part's led.

To aid the judging of the competition this system allows to set up and reset for each stage of the competition. The only manual task is the retrieval of the parts to the incoming warehouse. The software will be critical to determine if the right part was taken to the right drop location and specially if it was there the right amount of time, which must be no less than the machining time previously determined for that part, applicable only to the machines.

In syntheses, the system will need to:

- Detect the presence of parts in each station of the Warehouse/Machine.
- Read the part's information(id, color and battery level).
- Change the part's information(id, color).
- Display field information in the jury's computer.
- Control the state of the machines.
- Control the state of the field via computer.

5.2 Architecture

The architecture of the Factory's Monitoring System is as follow (figure 5.1).

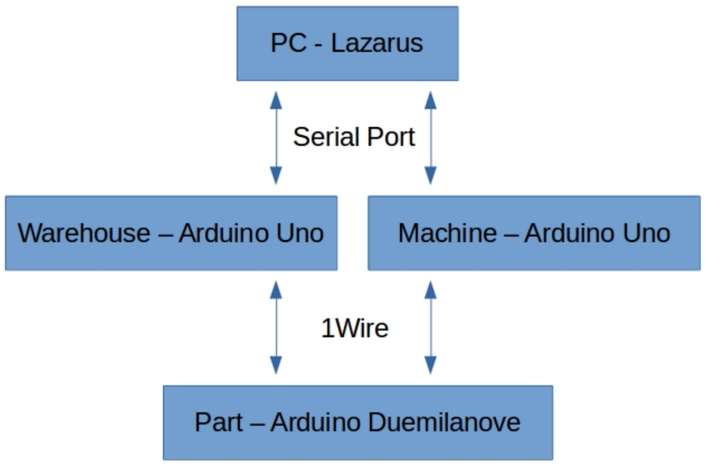
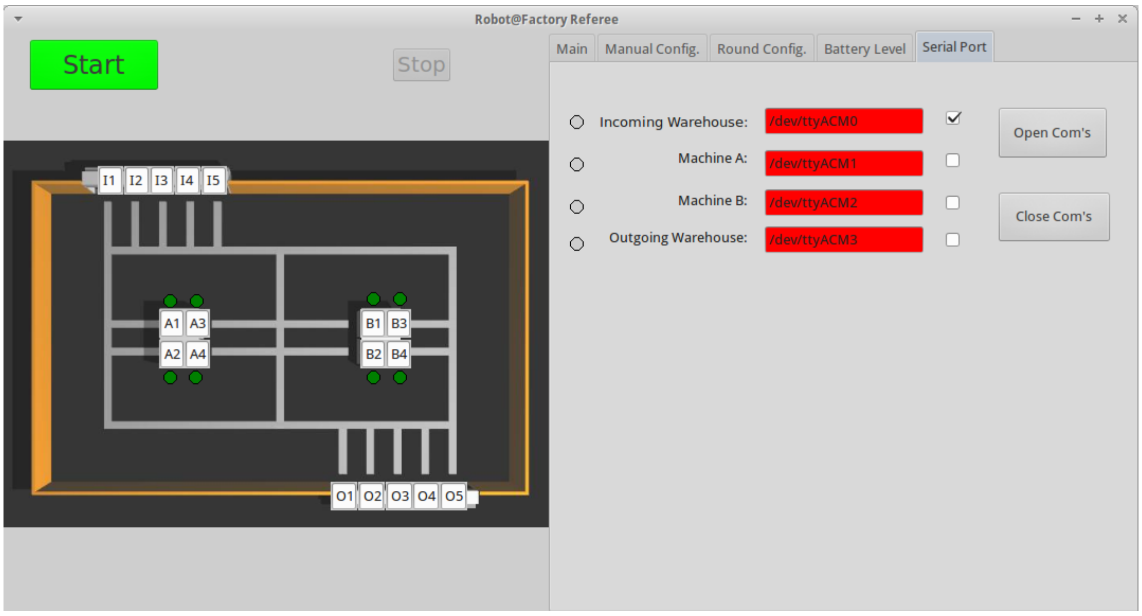


Figure 5.1: Architecture of the Factory Monitoring System

5.2.1 Computer

The Computer serves as a Human-Machine Interface(HMI) allowing the jury to track where the several parts are, view their id and color, and displays the voltage level of the batteries of each part. Displaying the battery level of each part allows the jury to know exactly when to substitute them, thus avoiding the problem of a part's led turning off while someone is competing. Furthermore this software regulates the machines processing time, and randomizes the colors of the part's led maintaining the correct ratio between blue, red and green parts each round.

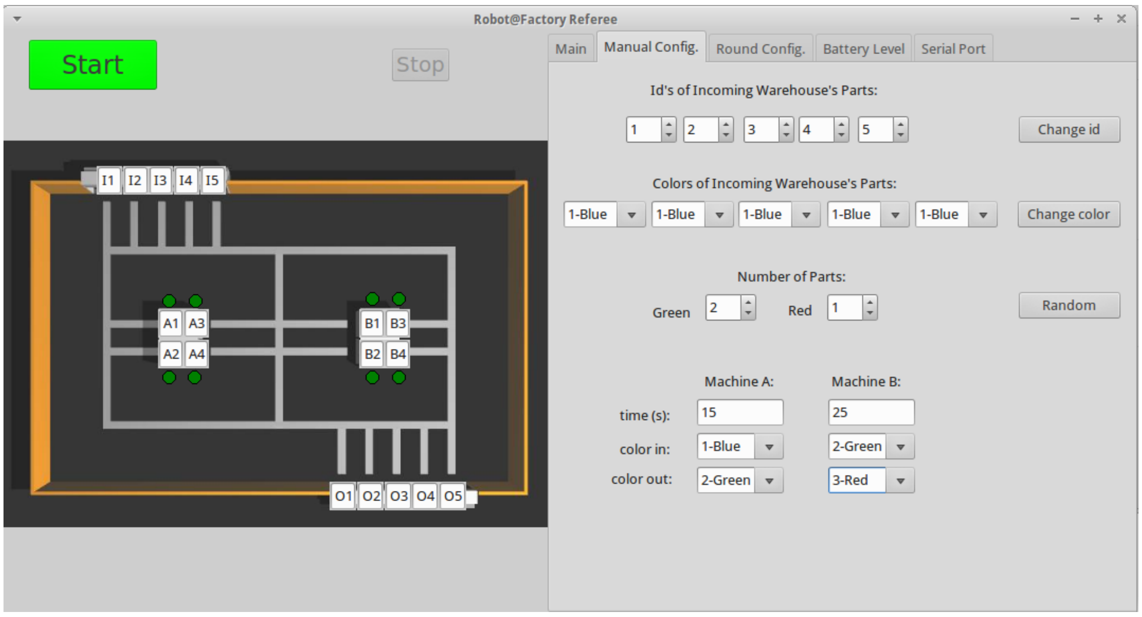
As seen in figure 5.1, the computer and machines/warehouses are connected over serial port, so when we open the interface, one of the first steps should be configuring the serial ports, as seen in the follow print screen (figure 5.2). The green soft-LED left to each Serial Port will toggle whenever the Serial Port receives a message, thus assuring the *Arduinos* and Computer are communicating.



Incoming Warehouse: /dev/ttyACM0 ✓
Machine A: /dev/ttyACM1
Machine B: /dev/ttyACM2
Outgoing Warehouse: /dev/ttyACM3

Figure 5.2: Print Screen of the Program - Serial Port tab

Next step would be defining the parameters of the competition's rounds. There are two options for this, either use the *Manual Config.* tab to configure a custom round as seen on the figure(5.3) or use the *Round Config.* tab to define the parameters for each round of the competition as shown on figure(5.4). In both cases we can set, for a specific round, the number of parts of each color, the time it takes for each Machine to transform a part, and the color transformation it occurs in each Machine. Regarding the *Manual Config.* tab (figure 5.3), in addition to the previously mentioned features it as a couple more. It allows to define the id of the parts in the Incoming Warehouse, which is useful no only to track a part during the competition but also to identify one unique part and check it's battery level in the corresponding tab (figure 5.5).



Id's of Incoming Warehouse's Parts:

1 2 3 4 5

Colors of Incoming Warehouse's Parts:

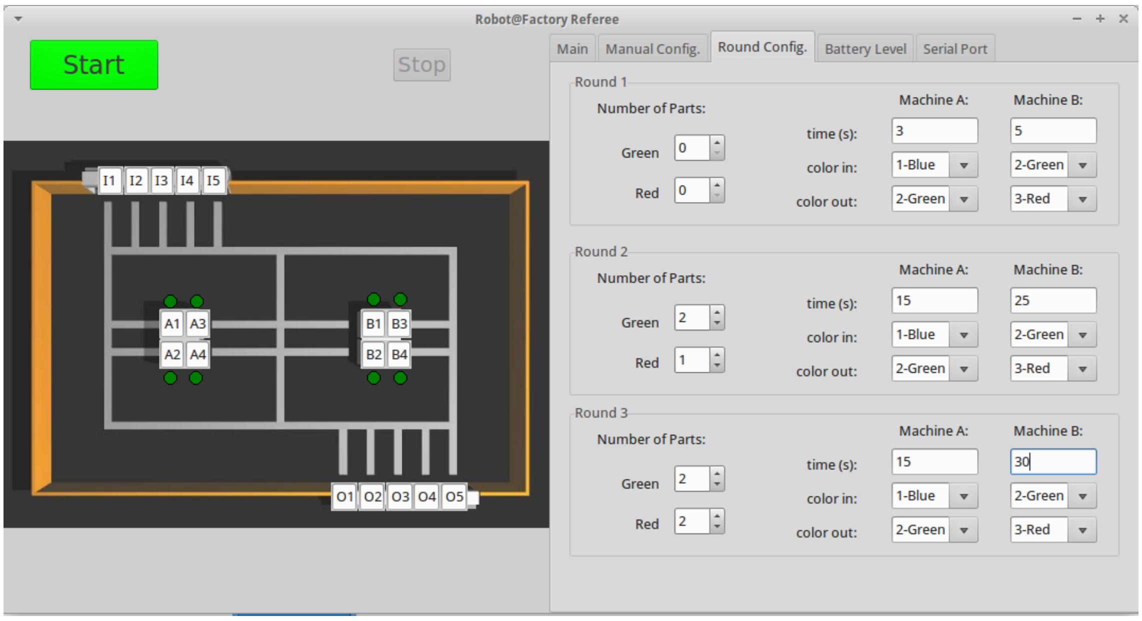
1-Blue 1-Blue 1-Blue 1-Blue 1-Blue

Number of Parts:

Green 2 Red 1

	Machine A:	Machine B:
time (s):	15	25
color in:	1-Blue	2-Green
color out:	2-Green	3-Red

Figure 5.3: Print Screen of the Program - Manual Config. tab



Round 1

Number of Parts: Green 0 Red 0

	Machine A:	Machine B:
time (s):	3	5
color in:	1-Blue	2-Green
color out:	2-Green	3-Red

Round 2

Number of Parts: Green 2 Red 1

	Machine A:	Machine B:
time (s):	15	25
color in:	1-Blue	2-Green
color out:	2-Green	3-Red

Round 3

Number of Parts: Green 2 Red 2

	Machine A:	Machine B:
time (s):	15	30
color in:	1-Blue	2-Green
color out:	2-Green	3-Red

Figure 5.4: Print Screen of the Program - Round Config. tab

As previously stated, it is of utmost importance, to know the battery level of each part, so that the part won't shut down during the competition, but also to avoid having to manually check with a multimeter and avoid constantly charge and replace the batteries to make sure they last through the round. Hence the need for the battery level tab as seen on the figure 5.5. It shows the last recorded voltage value of up to 15 parts, the equivalent of three sets of parts as there is the need for plenty of spare parts. To update the voltage value is simply a question of placing it on a machine or warehouse and making sure the serial communications are working.

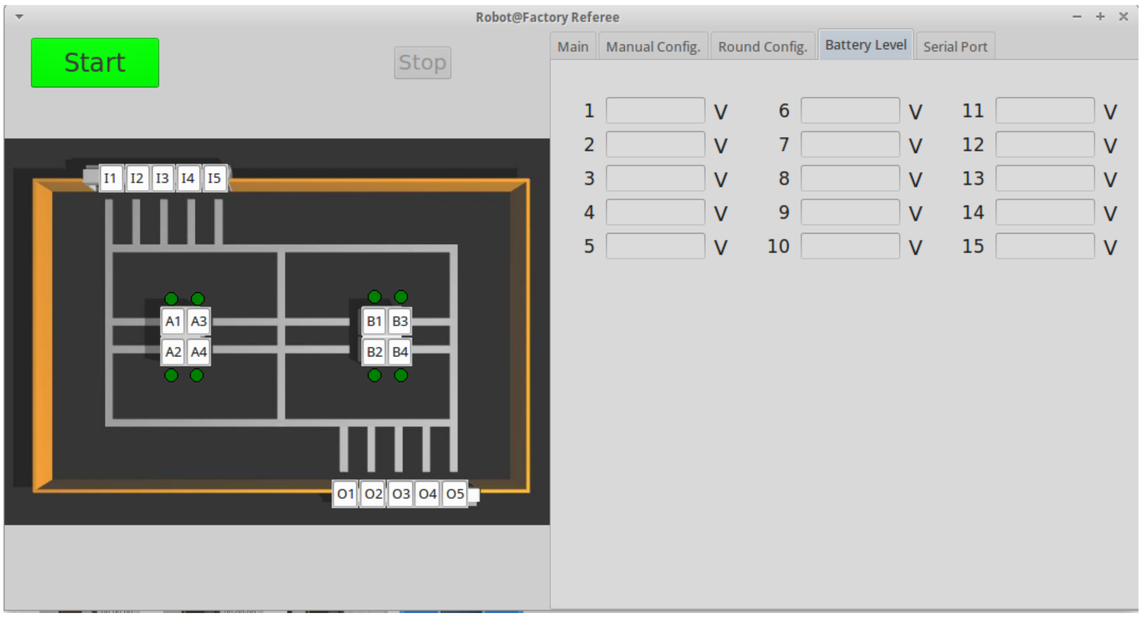


Figure 5.5: Print Screen of the Program - Battery Level tab

All preparations completed, all that is left is to choose the round of the competition desired and press start. When it is pressed, a timer will start and display the time both below "run time", and also in a separate window just for the audience. The timer will stop when the stop button is pressed to reset the trial, or when trial is complete, i.e. five blue parts are in the outgoing warehouse.

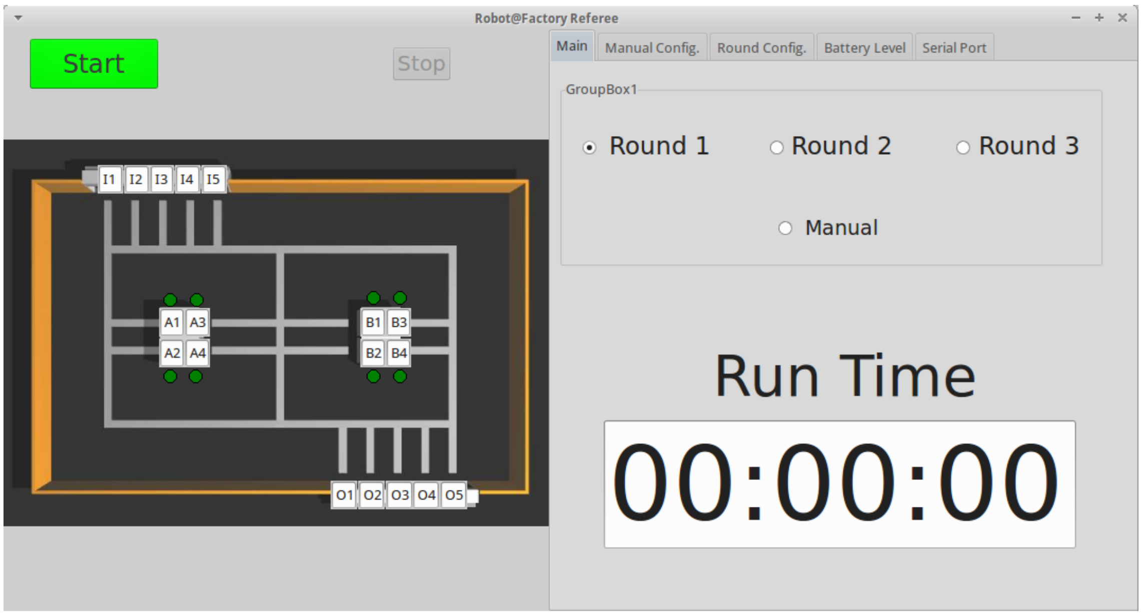


Figure 5.6: Print Screen of the Program - Main tab

As seen on the figure 5.7, the program will display the id and color of the part in corresponding box on the field diagram.

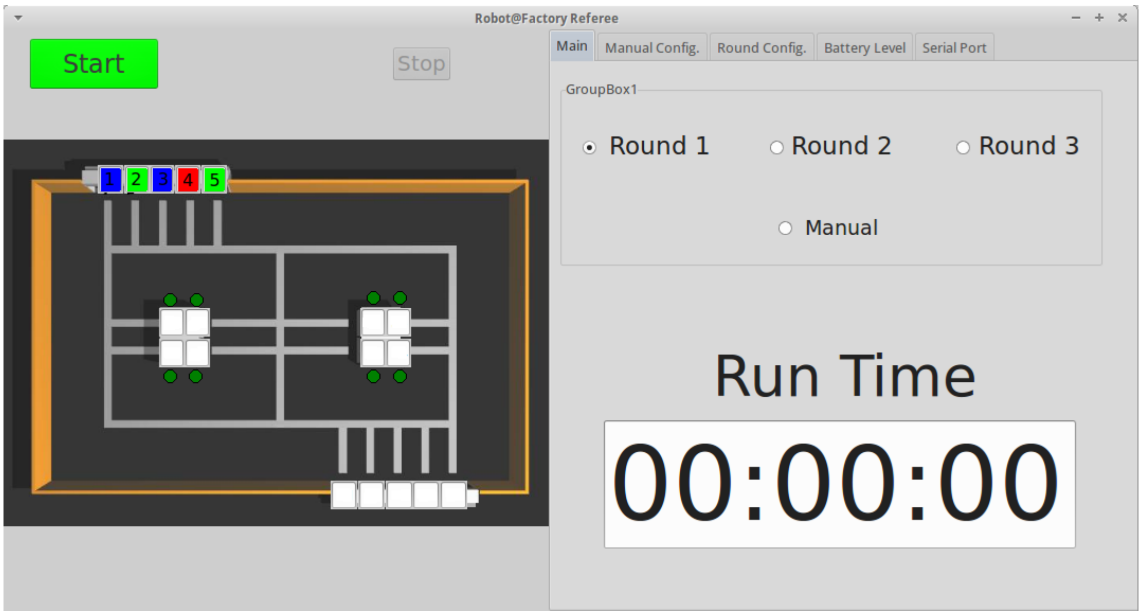


Figure 5.7: Print Screen of the Program - Main tab

5.2.2 Serial Port

On the Serial Communication between Computer and Machine/Warehouse was implemented a three character protocol. First character is a letter between G and Z. From higher level to lower level was used upper-case, on the other hand, lower-case was used from lower level to the higher level. The next two characters represent an hexadecimal number, which means that the characters were between 0 and F.

The first letter corresponds to a different function according to the following table (5.1).

Table 5.1: Function corresponding to each letter in the protocol implemented over Serial Port

Letter	Function
s	Sets *arduino* to behave either as a machine or warehouse
p	Indicates whether or not it has a par in it
i	Sends the id of a part
k	Sends the color of a part
t	Sends the high part of the value of battery voltage
v	Sends the lower part of the value of battery voltage
I	Sets part id
K	Sets part color
L	Sets machine state

The first hexadecimal character indicates the station of the machine/warehouse, and the second one the value. For example, i2A would be a message from a machine/warehouse to the pc indicating that in the station 2 there's a part with the id 10.

5.2.3 Machine/Warehouse

The firmware on the machines and warehouses was updated and was implemented a non-pre-emptive micro-kernel to manage the several tasks in execution. For practicality the code was structured so that the same firmware works for either machine or warehouse.

The *Arduino* in each of the aforementioned station has the following responsibilities:

- Detect whether or not there's a part in each station
- Read the parts' information (color and id)
- Send commands to change the id and/or color of each part
- Change the machine's state (the led on top of the stations)
- Communicate with the computer via serial port

5.2.3.1 Scheduler

Due to the potentially high number of tasks the machines/warehouse have to deal with at one time, a scheduler was implemented to better manage the processor's usage, prioritizing the read and write serial and when these tasks aren't utilizing the processor cycling thought the other tasks.

In order to come to a good solution for the architecture of the micro-kernel first we need to know what are the exact tasks that it must schedule. Also to predict the kind of delays we might have to deal with when implementing the scheduler it was measure the execution times of each task.

Read Serial Port (0,008 ms)
It's responsible for reading the packets in the incoming serial buffer, containing orders from the computer to the Machine/Warehouse, decoding it and add the corresponding task to a queue of tasks to be executed.

Write Serial Port (0,45 ms)
Based on the state it creates a packages with all that information (whether or not it has a part in each of the stations, the colors of those parts, their id's, and state of those parts' batteries) and sends it via Serial Port.

Establish 1-Wire Connection (1.1 ms)
The master (warehouse or machine) resets communications and searches for address to assert whether or not there is a slave connected.

Read id (7,5 ms)
Sends a command to the slave for it to return it's id.

Read color (7,5 ms)
Sends a command to the slave for it to return it's color.

Ping to 1-Wire slave (7,5 ms)
Verify that the connection wasn't broken.

Change part's color (7,5 ms)
Sends a command to change the slave's led color.

Change part's id (7,5 ms)
Sends a command to change the slave's id.

Machine's led color (0,004 ms)
Changes the color of the rgb led just above each station.

All 1-wire tasks start by resetting the communication to check whether the 1-wire connection is still active. If it is the rest of the task is executed, if not the task ends and the processor moves on to the next task, but not before using the ticks of the scheduler to calculated the time interval since the last successful communication. In case this interval is less than the time-out that was set, it could be just a bad connection so this task is added to the end of the One-Shot Tasks' list to be tried again later, otherwise, if the interval exceeds the time-out period it is assumed that the part was removed from that station so the Establish 1-Wire Connection task for that station is added to the end of the One-Shot Tasks' list.

The implemented solutions is a non-pre-emptive micro-kernel that splits the tasks into two lists, the periodic tasks list and the one shot tasks' list (table 5.2).

Table 5.2: List of task

Periodic Tasks' list	One-Shot Taks' list
Read serial port	4x Establish 1-wire communication
Write serial port	4x Read id of part
Read battery level	4x Read color of part
	4x Ping to part
	4x Change id of part
	4x Change color of part
	4x Change machine's led's color

The Periodic Tasks' list is constituted by tasks that ideally should be executed with a certain periodicity, although in this case some delays aren't critical. This list is set so the tasks in it have priority to the one-shot tasks' list. This list is constituted by the read and write to the serial port as well as the the read battery level. Obviously, the read and write serial port should be a periodic task with a high frequency, and little delay. Delay in the read serial causes information to accumulate in the buffer causing the cpu to be stuck in the decoding serial port for a while, causing delays in the other tasks. Since the output buffer is only created in the write serial function, excessive delay might cause events not to be reported via serial port, for example a part is quickly set and lifted. As the write serial is a task with high priority, high frequency and minimum delay as will be shown below, it's justified to build the output buffer then with the current state of the system. This way we minimize repeated information without worries of part of the information being lost as it will be updated in the next period. The battery level task wouldn't need to have priority, still it was put in this list in order to control the frequency at which it's executed. This tasks should have a low periodicity since the fluctuation between measured values is minimal.

The One-Shot Task's list is constituted by lower priority tasks that will run whenever the cpu is free and there isn't any Periodic Task waiting to be executed. This list is constituted by the 1-wire protocol functions as well as the task responsible to change the rgb led above each of the machines' stations. This list is FCFS (first come first served), which means the first task added will be the to be executed. In order to implement the 1-wire protocol, we need an state machine, so according to the state, one of the 1-wire tasks for each station will be always in this list awaiting to be executed, once it finishes, according to its result, it adds another task to the list.

5.2.4 1-Wire

1-Wire is a device communication bus system designed by Dallas Semiconductor Corp. that provides low-speed data, signalling, and power over a single signal.

As oppose to a 2 wires configuration where we have a wire designated for data, and clock signal, in 1-wire protocol we use just one wire and master and slaves take turns using it to send a message. As in all protocols another wire must be added for ground in case both system do not share the same power supply, and thus do not share the same ground.

5.2.5 Part

Each part has an Arduino Uno, responsible of dealing with the Slave's part of the 1-wire Protocol. It's responsible for on command updating the color of the RGB Led, updating the id of the part to the *eprom* memory, and measuring the voltage of the batteries.

Chapter 6

Schedulling for single AGV environment

6.1 for single AGV environment

The work done in this dissertation started by scheduling tasks for just one robot, and then escalating to multiple robots. Beyond being easier to start by implementing just for one robot and then progressing to multiples robots, the second round of the Robot@Factory competition requires the scheduling to just one robot. So to start this problem, it was chosen a complete search algorithm that schedules jobs for just one robot, and that would work with any routing algorithm, provided that the time it takes for the robot to get from one point A to a point B is fairly constant throughout multiple runs, as the scheduling algorithms generally focus on the time of travel instead of distances as is the case of static routing algorithms.

This chapter will describe only offline algorithms. A single AGV system in which the all jobs are known in advanced and without obstacles or machines breakdown, there's no need to implement online algorithms. The objective of this algorithms will be to return a list of pick & deliver commands.

6.1.1 Branch & Bound

As was previously stated, this algorithm is based on the travel times, so it is required to know the average time it takes for the robot to travel between the points in the figure(6.1).

$$M[c,d] = t \tag{6.1}$$

Then we compile that information into a matrix M, and as seen on equation 6.1, t is the time it takes the robot to get from the current position (c) to its destination (d). This matrix is going to be used to calculate the costs for the nodes in the B&B algorithm.

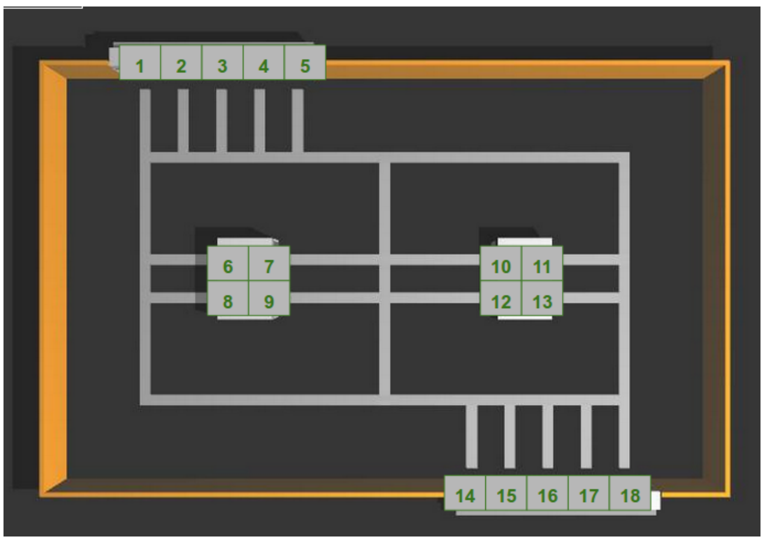


Figure 6.1: Numbers representing each Pick and Deliver locations

The algorithm starts by creating the initial node, which in the presented problem, will be setting the location of all boxes to the incoming warehouse, set the robot position to the starting position, and setting the cost 0. From this node it will generate children based on the possible drop destinations for each of the parts. Those nodes will be added to a binary heap. It will keep generating children from the branching node (the first position of the binary heap, which is the node with the lowest cost) until either there's no node with lower cost than the best position obtained so far or the binary list is empty. The latter option is unlikely, since there is a solution to this problem and there will always be solutions with higher costs then the best solution.

Algorithm *B&B*

Variables:

AN - List of all nodes generated

NVN - Index of non visited nodes in AN with *binary heap*

NVN[0]- Branching node, the non visited node with least cost

BV - best value, lowest cost out of all visited nodes

BI - best index, index of BV in AN

Pseudocode:

```
1:      Add Initial node a AN
2:      Generate Children from NVN[0]
3:      Remove NVN[0] from Heap
4:      For each generated child
4.1:        Add child to AN
4.2:        If child is a complete solution:
4.2.1:          If child's cost < BV: update BV and BI
4.3:        Else: add child's index to NVN
5:      If N not empty and N[0]<BV: jump to 2
6:      Else: best solution found.
```

6.1.1.1 Generate children function

The Generate Children function needs to calculate, for each part not delivered to the outgoing warehouse, the travelling time between the robot's current position and the pick location, and between said location and a valid drop location. For the parts in the Machines there is one extra step, which is to verify that when the robot reaches the machine, the part is ready to be retrieved, if not, the time the robot needs to wait has to be added to the cost.

A binary heap was implemented to keep the list of not visited nodes order by cost. In the above algorithm the branching node for generating children is the node in position 0 of the heap, as it is the node with the least cost.

6.1.1.2 Binary Heap

As it is common when implementing a search algorithm, there is a need to store information related to the node, such as its cost and the index the parent. Moreover in cases where the next branching node is decided by costs and/or heuristics of previously generated nodes, it is recommended to keep them organized, as it takes less processing time than to search through the nodes whenever choosing a branching node, since it has low temporal complexity for insertion and removal of nodes n(log n), being n the number of nodes.

The two key information needed for a binary key is the index and the cost of a node. However, since in the described algorithm the cost needs to be associated with a node of the branch & bound tree, the binary heap implemented only needs to store the index of each heap node and id of the B&B node. Children and parents index don't need to be stored as they can be easily calculated with the following equations where i represents the index of the current node.

Left son: $2i+1$

Right son: $2i+2$

Parent: $(i-1)/2$

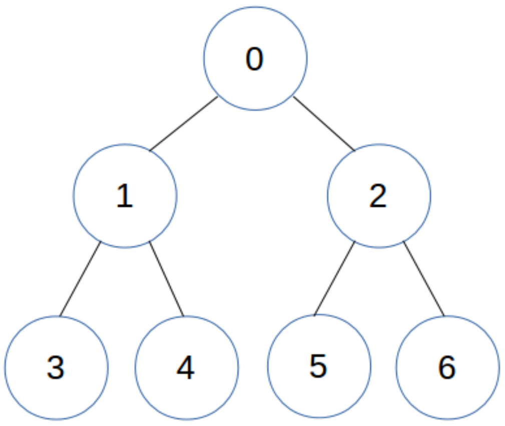
The follow equations are only true if the index starts with 0, if instead the index starts with 1, the calculations change to the following:

Left son: $2i$

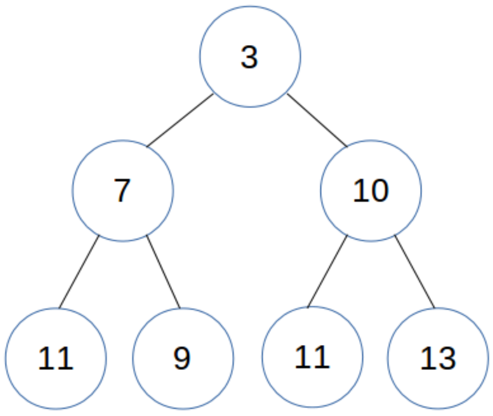
Right son: $2i+1$

Parent: $i/2$

Binary Heap is a cost organized binary tree that points to the nodes generated by the B&B. It can be organized either keeping the lowest on top, or the highest. As this is a minimization problem, the lowest value will be kept on top.



(a) Indexes



(b) Values

Figure 6.2: Binary Heap tree

When adding a node to a binary heap the node is always added to the last position, which on the example of the figure6.2a would be the node with index 6. Then to keep the heap organized, the added node will keep being compared with it's parent and replaced if it's smaller, until it gets to a position that it is no longer smaller than its parent, or reaches the initial node.

Function Add to Heap

Variables:

Heap - List of Nodes

i - Index of current node

i_f - Index of last position of the list

c - Cost associated with a node

Pseudocode:

```
1:      Add position to the Heap
2:      Add node to position $i_f$
3:      $i = i_f$
4:      While i > 0 and Heap[(i-1)/2].c > Heap[i].c
4.2:        Swap Heap[(i-1)/2] with Heap[i]
4.3:        i := (i-1)/2
```

In the described algorithm we need to remove de branching node from the list, which will always be the initial node from the binary heap tree. We do this by swamping the first and last nodes of the tree, and then removing what now will be the last node. Then to keep the three organized the first node is compared to its children. If the smaller of those children has a lowest cost than its parent, it is swapped. The node will be brought down this way until it is no longer bigger than any of its children.

Function Remove Node from Heap

Variables:

Heap - List of Nodes

i - Index of current node

i_f - Index of last position of the list

sc - Child with the lower cost

c - Cost associated with a node

Pseudocódigo:

```
2:          Heap[0]:=Heap[$I_f$]
3:          Remove last position from the list
4:          If Heap not empty then
4.1:            i:=0
4.2:            While i*2+1 < Size(Heap)
4.2.1:              sc := i*2+1
4.2.2:              If i*2+2 < Size(Heap) and (Heap[i*2+1].c > Heap[i*2+2].c)
4.2.2.5:                sc:= i*2+2
4.2.3:              If Heap[i].c > Heap[s].c
4.2.3.5:                swap Heap[i] with Heap[sc]
4.2.4:              else break
4.2.5:              i:=sc
```

The max number of iterations in either the removal or addiction of a node is the number of levels in the binary heap, making it faster than searching through a list.

6.2 Scheduling for Multiple Robots

This chapter goes through some of the necessary steps of solving a scheduling problem. As previously mentioned in chapter 2, some steps will be left behind, as they are not part of the problem at hands, such as the battery management.

The chapter starts by explaining the path designed used in the solution created. Then, the scheduling is addressed, as well as its two key elements, the dynamic routing and the deadlock avoidance. Finally, the chapter ends with path following for the AGV.

6.2.1 Path Design

Even thought the robot (described in chapter 4) has a laser, enabling it to locate itself anywhere in the map, its movements are going to be restricted to the paths represented by the green lines in figure(6.3). This reduces the complexity of the problem, meaning the computer can calculate solutions faster and the deadlock avoidance is going to be easier to implement. Although from the point of view of routing, it's not as optimal solution as dividing the map in very small squares and using a search algorithm to calculate the shortest path between two points, the advantages still outweigh the disadvantages.

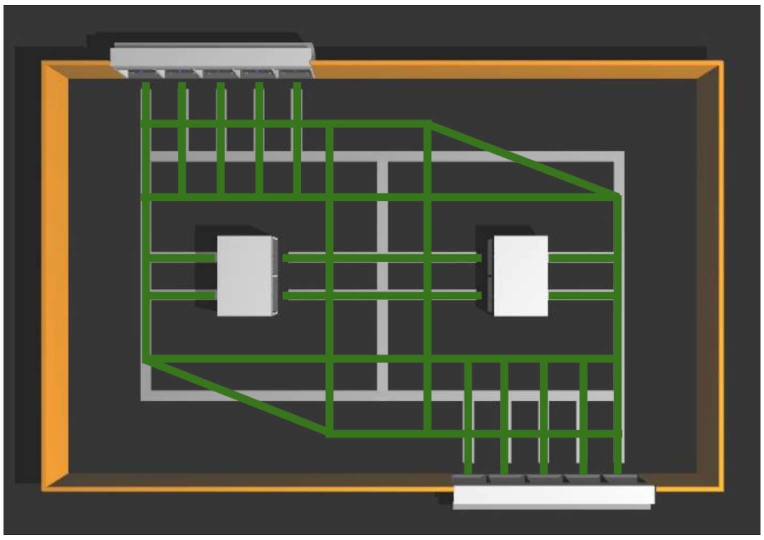


Figure 6.3: Designed ladder system with parallel lanes and multi-directional sections

As seen in the previous figure (6.3) the path has parallel lanes in the top, bottom, and middle areas of the field, to facilitate traffic and allowing solutions to solve deadlocks with small deviations to the shortest path.

Having parallel lanes in such a small space leaves little room for error, about $2,5cm$. Taking in account the maximum distance from the robot to the furthest wall at any given state, the maximum error obtained from the laser measurements, inside the field, was of around $1cm$ [19]. Since the margin for error is more than twice bigger than the maximum error in localization, collisions are avoided.

Still, as the focus of this thesis is on the scheduling and not so much on the location or obstacle avoidance, the work was carried on the assumption that the robots have a perfect locating system. To accomplish this, the system will be implemented in the *simtwo*, using the true position of the robot in the field, instead of calculating it from the data received from sensors.

6.2.2 Routing

Having decided upon a path design, before jumping to the scheduling, first was implemented a dynamic routing. There are two good reasons for this. First, the obvious divide and concur, by starting with just the routing, later will become easier to implement the scheduling. Also this program can be used to calculate, offline, a table with the heuristics for the Scheduling, as will be further explained in section 6.2.3.

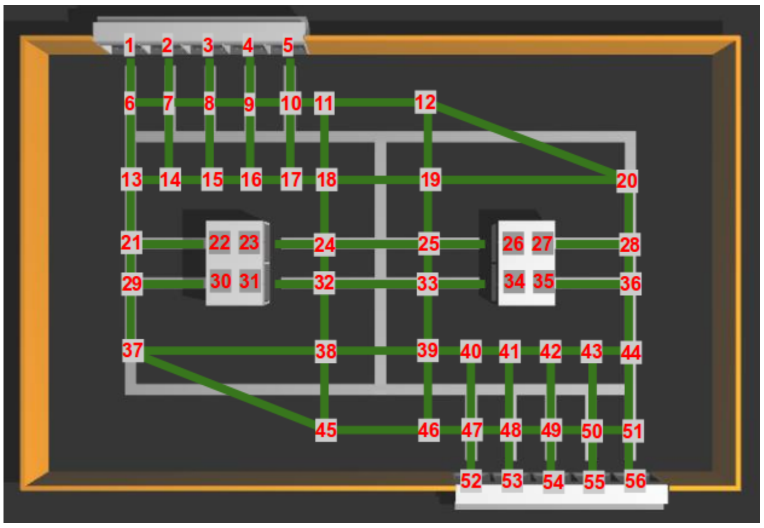


Figure 6.4: graph of the factory field

A graph was created with the nodes shown in figure 6.4, and also information on the arcs connecting those nodes was stored. Information that ranges from average travel time, by the AGV described in chapter 4, for that arc, orientation of the arc (horizontal, vertical, diagonal), and orientation of the robot when crossing each arc. The reason for this last one is the aforementioned small spacing between the arcs. Since the claws are the furthest part from the center of the robot, when two AGVs are travelling through certain parallel arcs, if the claws are facing each other, they would hit one another.

6.2.2.1 Algorithm

The routing was obtained implementing an A* algorithm, but as previously mentioned, using the costs as travel time. The heuristic was calculated measuring the distance in straight line from the current position to the destination, and dividing by the robot's speed.

Algorithm $A*$

Variables:

AN - List of all nodes generated

NVN - Index of non visited nodes in AN with *binary heap*

NVN[0]- Branching node, the non visited node with least cost plus heuristic

Pseudocode:

```
1:      Add Initial node a AN
2:      Generate Children from NVN[0]
3:      Remove NVN[0] from Heap
4:      For each generated child
4.1:        Add child to AN
4.2:        If child = destination:
4.2.1:          Print Routing Path
4.2.2:          No more iterations
4.3:        Else:
4.3.1:          Add child's index to NVN
4.3.1:          Jump to 2
```

Each node of the A* is associated to a node of the graph (figure 6.4), representing the location. The initial node is created. Children are generated from the branching node, which is the visited node with lower cost plus heuristic. The number of children generated equals the number of connections of the corresponding node in the graph. Just like the B&B in chapter 6, a binary heap was used to keep a organized list of visited nodes, and the position 0, corresponds to the branching node. The algorithm will continue to iterate until it reaches the destination node in the graph. Finally the routing is obtained by taking the final node and iteratively checking the parents until reaching the initial node.

6.2.3 Scheduling

The Robot@Factory problem isn't a dynamic problem, at least for the first two rounds, since right from the start of each round, a robot can know the complete list of the tasks for that round, by checking all the parts LEDs. However it is good idea to implement a dynamic algorithm because as more teams are capable of finishing the competition, the organization is planing in increasing the difficulty by adding obstacles mid round or having machines breakdown during the third round, making this a dynamic problem, at least in the last stage of the competition. Still event if the problem is static, using a dynamic scheduling algorithm has the advantages of recalculating after each time a part is picked or delivered, or even periodically if the algorithm is fast enough. This has the advantage that even if an AGV suffers a delay (from for example wheel spin, or getting to a intersection after another robot, and having to wait), it calculates the best order and route based on the current state of the system, this is specially useful for better managing potentially deadlock situations, which can happen where AGVs want to cross the same arch.

The algorithm implemented calculates the scheduling of tasks using a dynamic routing to more accurately calculate the costs. Both scheduling and routing were implemented in the same A* algorithm. The implemented A* can be divided in three levels as seen in figure 6.5. There's the assignment of tasks, picking and delivering.

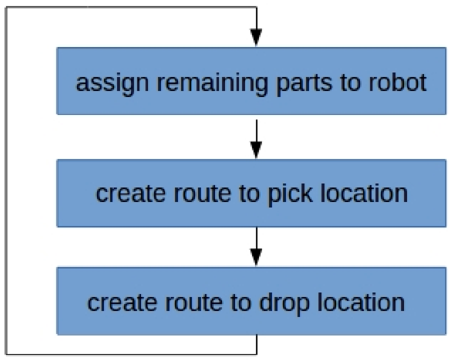

assign remaining parts to robot
create route to pick location
create route to drop location

Figure 6.5: Three possible states of a node in the implemented A*

The algorithm starts by assigning tasks to the AGVs, this is accomplish by permutations of the number of tasks by the number of robots. In equation 6.2, N represents the total number of nodes in level one for T number of tasks and R number of robots.

$$N = \frac{T!}{(T-R)!} \tag{6.2}$$

Once completed it goes to the picking, so from that moment on until it gets to the part location, this becomes purely a routing problem, just like in section 6.2.2. Once it has the part it goes into delivering mode, which is pretty similar to the picking, except instead of just one possible destinations it has multiple possible destinations, as it can be put in either free slot of a particular machine or warehouse. After delivering a part it goes back to assigning parts, generating one node for each of the parts remaining and then creates pick routes for said parts.

6.2.3.1 Algorithm

This algorithms is very similar to the one detailed in the Routing Section 6.2.2, the main difference, apart from including scheduling, is that the generating children algorithm varies according to the state of the node, as previously mentioned (figure 6.5).

Algorithm $A * Scheduling$

Variables:

AN - List of all nodes generated

NVN - Index of non visited nodes in AN with *binary heap*

NVN[0]- Branching node, the non visited node with least cost with heuristic

Pseudocode:

```
1:      Add Initial node a AN
2:      If NVM[0].state = Assign Part
2.1:        Generate Children from NVN[0] - Assign remaining parts to the current robot
3:      Else if NVM[0].state = Pick Part
3.1:        Generate Children from NVN[0] - Do next iteration on the routing to pick a part
4:      Else if NVM[0].state = Deliver Part
4.1:        Generate Children from NVN[0] - Do next iteration on the routing to deliver a part
5:      Remove NVN[0] from Heap
6:      Add generated children to AN and NVM
7:      If in any child all parts are in the outgoing warehouse
7.1:        Print Routing Paths
7.2:            End of algorithm
8:          Else:
8.1:            Add child's index to NVN
8.2:            Jump to 2
```

6.2.3.2 Recalculating

By updating the initial node with the current state of the system, the algorithm can be recalculated at any time. For that it needs the current position of the robots, their travel time, and the location of all parts.

6.2.3.3 Deadlock Avoidance

The Deadlock Avoidance is achieved in two ways, both in the scheduling as well as in path following, as there might be some slightly differences in the time that two AGVs get to an intersection from what was planed in the scheduling to what happened when the tasks are executed.

In the scheduling this is accomplished by only moving one AGV per node, and will be always the one with the lowest cost, this way for any given node, it is known the "current" positions of all AGVs. It's not exactly the current position as different AGVs' will have different costs in the same node, which means they are not in the same position in time. Since only one AGV will be moved per node, and it is always the one with the least cost, the difference in cost will be kept minimal. So there is enough data to plan a rout without deadlock. When generating children in the pick or deliver part of the algorithm, it needs to check if the new arc is being occupied by any of the other robots, if so the robot must wait. The node created is still valid, but to the cost it will be added the time it has to wait.

This will make for an accurate scheduling, since it is accounting even the time robots have to wait, and not just travel times.

6.2.4 Path Following

The scheduling will return an array of points in the map that make the routing. So for the path following there's only the need for a simple follow line [20, 21, **?**] to navigate between those points.

The figure 6.6 taken from shows a typical representation of the vectors at play in the control of the movement of a robot.

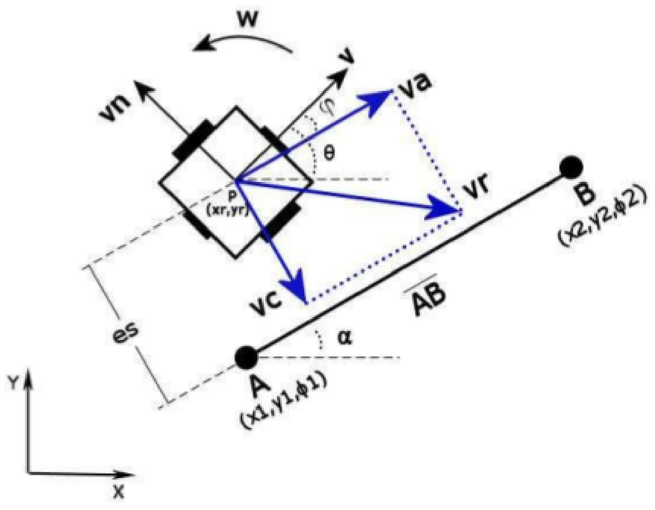


Figure 6.6: Representation of robot movement [21]

Applying a change of direction to this classical solution, using a rotation matrix and a translation so that the end of the segment is in $(0,0)$, the result is figure 6.7.

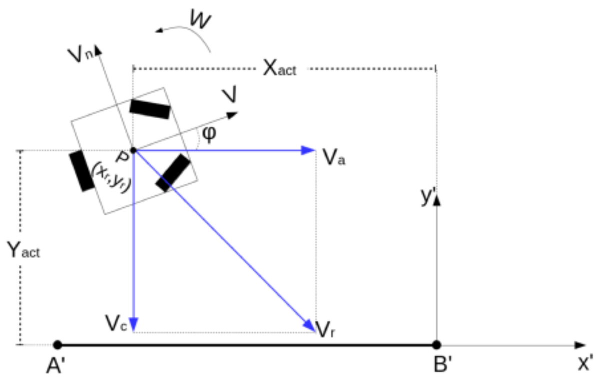


Figure 6.7: Representation of robot movement after change of coordinates [20].

This is a great advantage as it simplifies the control of the robot, because now the deviation to the path can be calculated using only the y coordinate, and the distance to the end of the segment is the absolute value of the x coordinate of the position of the robot.

6.2.4.1 Deadlock Avoidance

As in real life the average velocity of the AGV won't be exactly the same as the average velocity used in the scheduling, times from the scheduling may vary slightly from the execution of the tasks. This may be critical when two robots approach the same intersection as represented on figure 6.8. So it is implemented zone control during the path following. The field is divided into zones and a controller will ensure that no deadlocks occur inside that zone. There's no re-routing necessary. AGV wait in an arc until the next node and arc are free.

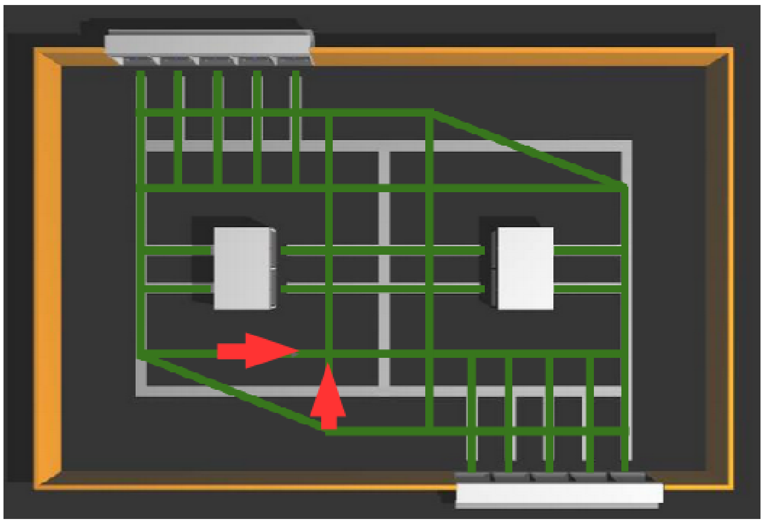


Figure 6.8: Two robots represented by red arrows approaching an intersection

Chapter 7

Results

This chapter shows the results obtained from the implemented programs.

7.1 Monitoring System

The tick of the micro-kernel was used to calculate the duration of each task, the results can be seen in table 7.1.

Table 7.1: Destination of the different parts and end result of the machining

Task	Processing time
Read Serial Port	0,008ms
Write Serial Port	0,45ms
Establish 1-Wire Connection	1,1ms
parts 1-Wire Read	7,5ms
1-Wire Write	7,5ms
1-Wire Ping	7,5ms
Change Machine State	0,004ms

Using the previous table the worst case for detecting a part can be estimated. This scenario would be dropping a part in the last available slot in a machine. The worst case scenario for this micro-kernel would be when a part is dropped as soon as the program finishes trying to establish a connection to the slot where it was dropped. So the program would have to do other three 1-Wire tasks before looping back to this slot. In the worst scenario the write serial tasks would be executed just before the 1-Wire communication to the slot with the part. So before detecting this part, the kernel would do four 1-wire communications, and wait one period of the write serial task which is $30ms$. So the worst scenario would be approximately:

$$3 \times 7,5ms + 30ms + 2 \times 0,45ms + 7,5ms = 60,9ms$$

Corresponding to doing three 1-Wire task $(1,1ms)$, one write serial $(0,45ms)$, doing the 1-Wire task for the correct slot (this time doesn't factor in the calculus because it occurs in the period of write serial task which is $30ms$), doing another 1-Wire task (which is the slowest) just as the write serial was about to be executed, and finally sending the information to the computer by serial port.

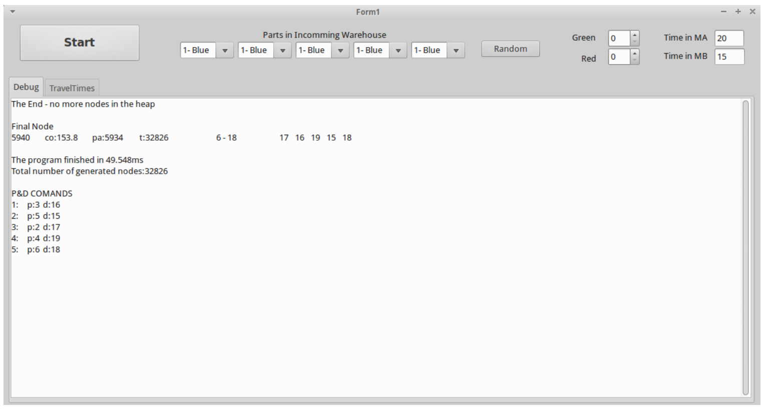
The calculus was simplified as before looping back to the correct 1-wire task the micro-kernel could execute a read serial$(0,008ms)$ and 3 change machine state $(0,004ms)$. So the absolute worst would be:

$$60,9ms + 0,008ms + 3 \times 0,004 = 60,92ms$$

This from just the point of the micro kernel, because if we want to know the worst time since the event of dropping a part to the computer receiving the information, it needs to be added the time of one period of the read serial function in the computer.

7.2 Scheduling single AGV

In figures 7.1 and 7.2 are print screens of the interface displaying the number of nodes generated, run time and the final list of pick and deliver tasks to be executed, for 2 different situations.



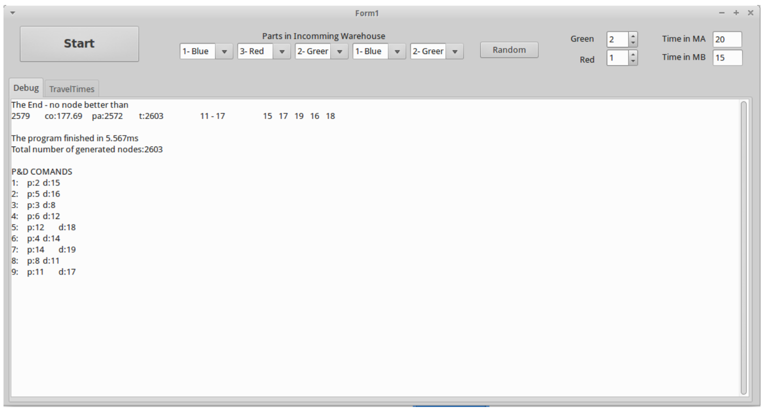
```
The End - no more nodes in the heap

Final Node
5940     co:153.8     pa:5934     t:32826          6 - 18          17  16  19  15  18

The program finished in 49.548ms
Total number of generated nodes:32826

P&D COMANDS
1:   p:3 d:16
2:   p:5 d:15
3:   p:2 d:17
4:   p:4 d:19
5:   p:6 d:18
```

Figure 7.1: An example of the scheduling for 5 blue parts



Selected settings: 1- Blue, 3- Red, 2- Greer, 1- Blue, 2- Greer; Green 2; Red 1; Time in MA 20; Time in MB 15

```
The End - no node better than
2579    co:177.69   pa:2572    t:2603        11 - 17        15  17  19  16  18

The program finished in 5.567ms
Total number of generated nodes:2603

P&D COMANDS
1:   p:2 d:15
2:   p:5 d:16
3:   p:3 d:8
4:   p:6 d:12
5:   p:12     d:18
6:   p:4 d:14
7:   p:14     d:19
8:   p:8 d:11
9:   p:11     d:17
```

Figure 7.2: An example of the scheduling for 2 blue, 2 green and 1 red parts

As it would be expected, the algorithm favours solutions that start by putting parts in the machines instead of taking parts directly to the outgoing warehouse. Moreover it favors the slots in the middle opposed to the slots in the borders of the field

7.3 Routing

The key advantage from having implemented a routing algorithm separated from the scheduling is that it allows to, on a previous stage, calculate the travelling times between all the pick and drop locations on the field, represented with number from 1 to 18 in figure 7.3.

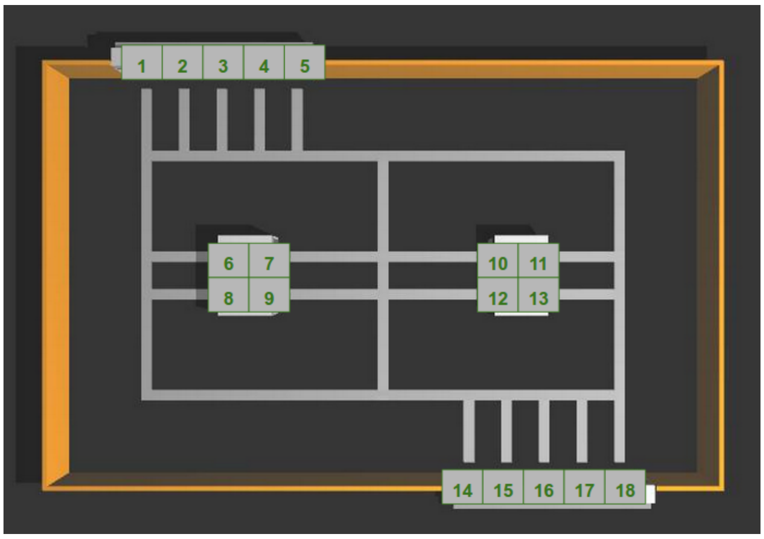
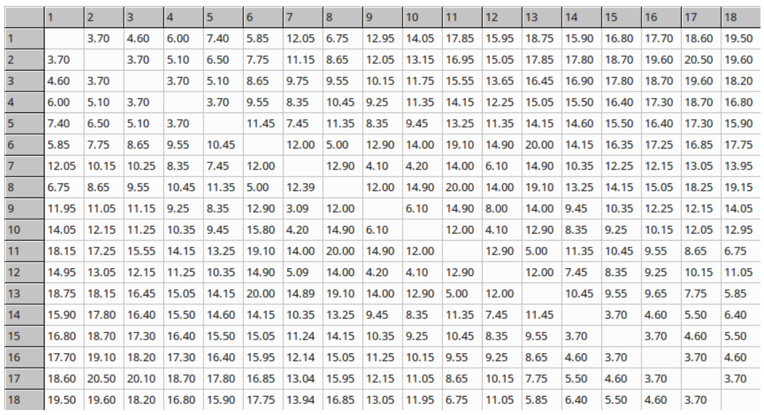


Figure 7.3: P&D locations

The result of this step can be seen in figure 7.4.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1		3.70	4.60	6.00	7.40	5.85	12.05	6.75	12.95	14.05	17.85	15.95	18.75	15.90	16.80	17.70	18.60	19.50
2	3.70		3.70	5.10	6.50	7.75	11.15	8.65	12.05	13.15	16.95	15.05	17.85	17.80	18.70	19.60	20.50	19.60
3	4.60	3.70		3.70	5.10	8.65	9.75	9.55	10.15	11.75	15.55	13.65	16.45	16.90	17.80	18.70	19.60	18.20
4	6.00	5.10	3.70		3.70	9.55	8.35	10.45	9.25	11.35	14.15	12.25	15.05	15.50	16.40	17.30	18.70	16.80
5	7.40	6.50	5.10	3.70		11.45	7.45	11.35	8.35	9.45	13.25	11.35	14.15	14.60	15.50	16.40	17.30	15.90
6	5.85	7.75	8.65	9.55	10.45		12.00	5.00	12.90	14.00	19.10	14.90	20.00	14.15	16.35	17.25	16.85	17.75
7	12.05	10.15	10.25	8.35	7.45	12.00		12.90	4.10	4.20	14.00	6.10	14.90	10.35	12.25	12.15	13.05	13.95
8	6.75	8.65	9.55	10.45	11.35	5.00	12.39		12.00	14.90	20.00	14.00	19.10	13.25	14.15	15.05	18.25	19.15
9	11.95	11.05	11.15	9.25	8.35	12.90	3.09	12.00		6.10	14.90	8.00	14.00	9.45	10.35	12.25	12.15	14.05
10	14.05	12.15	11.25	10.35	9.45	15.80	4.20	14.90	6.10		12.00	4.10	12.90	8.35	9.25	10.15	12.05	12.95
11	18.15	17.25	15.55	14.15	13.25	19.10	14.00	20.00	14.90	12.00		12.90	5.00	11.35	10.45	9.55	8.65	6.75
12	14.95	13.05	12.15	11.25	10.35	14.90	5.09	14.00	4.20	4.10	12.90		12.00	7.45	8.35	9.25	10.15	11.05
13	18.75	18.15	16.45	15.05	14.15	20.00	14.89	19.10	14.00	12.90	5.00	12.00		10.45	9.55	9.65	7.75	5.85
14	15.90	17.80	16.40	15.50	14.60	14.15	10.35	13.25	9.45	8.35	11.35	7.45	11.45		3.70	4.60	5.50	6.40
15	16.80	18.70	17.30	16.40	15.50	15.05	11.24	14.15	10.35	9.25	10.45	8.35	9.55	3.70		3.70	4.60	5.50
16	17.70	19.10	18.20	17.30	16.40	15.95	12.14	15.05	11.25	10.15	9.55	9.25	8.65	4.60	3.70		3.70	4.60
17	18.60	20.50	20.10	18.70	17.80	16.85	13.04	15.95	12.15	11.05	8.65	10.15	7.75	5.50	4.60	3.70		3.70
18	19.50	19.60	18.20	16.80	15.90	17.75	13.94	16.85	13.05	11.95	6.75	11.05	5.85	6.40	5.50	4.60	3.70	

Figure 7.4: Table of the travel times between P&D locations

This values from figure 7.4 were used in the heuristic of the Scheduling. Using this table instead of constantly calculating the heuristic helped reduce the Scheduling processing time.

7.4 Path Following

Figure 7.5 shows frames of the movement of the robot from the initial position, to the first slot of the Incoming Warehouse, and then going to the Machine B to deliver it.

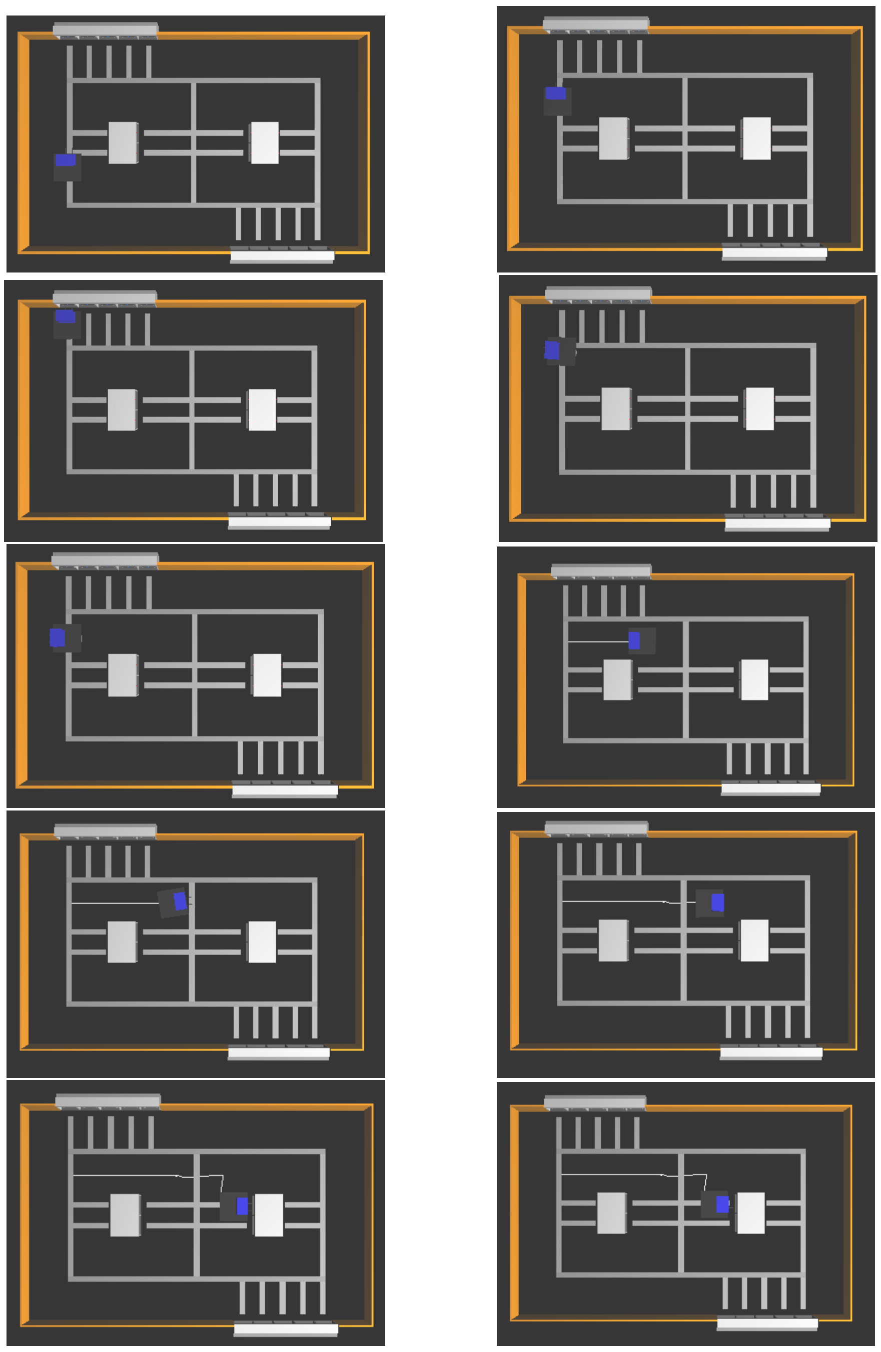


Figure 7.5: Green Part Picket from the Incoming Warehouse and delivered to Machine B

As it would be expected since the position of the robot in simtwo was obtained from the actual position and not the sensors, there's little deviations in the robots trajectory to the intended trajectory, which is a straight line between the nodes.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

Having implemented a micro-kernel in the *arduinos* has the advantage of being able to predict the worst case response of the system to a particular scenario. Having periodic task allows to easily change their frequency so test can be perform to find the optimal frequency. This also allows to easily set low frequency tasks such as reading battery level, which has the benefice of being one less task constantly wasting processing time.

The monitoring system connected to the Scheduling allows the scheduling of jobs with indeterministic times.

The scheduling for a single AGV fulfilled its purpose, its a fast algorithm that can be adapted to any kind of routing, or AGV.

The algorithms was implemented with the intent of being used on a low complexity problem. It was meant to obtain close to optimal results for this case. If tried to be adapted to a more complex problem the runtime and the memory needed would be enormous, and would be better to use other examples described in chapter 3, such as decentralized networks, or time windows. Aiming for the best possible solution wasn't the best solution for this problem. The algorithm implemented generated to many nodes, which not only made the debugging process slower and harder, it would also compromise the ability to execute the algorithm online, as the run times are higher.

8.2 Future Work

8.2.1 Monitoring System

To optimize the micro-kernel in the *arduinos*, the 1-Wire libraries should be rewritten in order to allow the implementation of a preemptive micro-kernel.

8.2.2 Optimizing the routing

Since the system has zone control to check deadlock avoidance, the system could be altered to check if there are no other robots between it and its destination. If not, the robot could follow a shorter rout not included in the path designed, such is the example in figure 8.1.

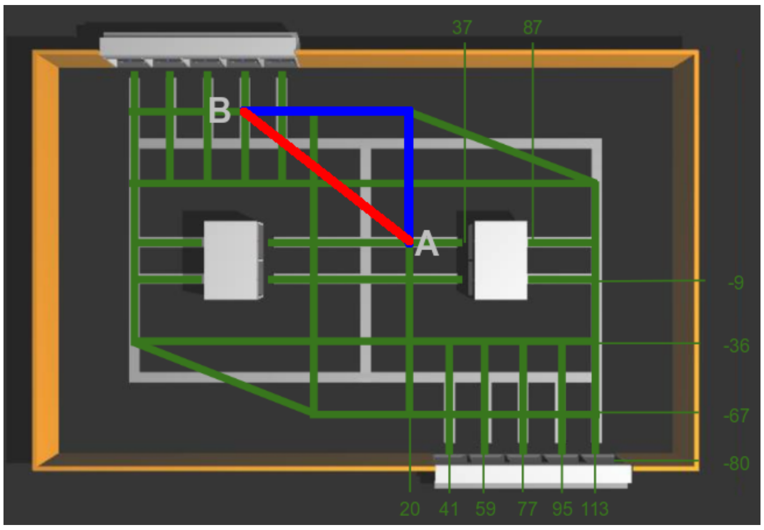


Figure 8.1: Using a short-cut

This could be accomplished by running a set of rules in the AGV in parallel to the path following, given the small dimension of the field, and taking in account that the AGVs know each others position and trajectory.

8.2.3 Optimizing the A*

As is described the A* implemented for multiple robot scheduling is generating to many nodes, some my be removed by not allowing the robot to go back to the previous node of the graph except when backing-up from picking or delivering a part. Using gains in the heuristics may also get the final solution faster. Making for a faster algorithm.

8.2.4 Scheduling

Looking over the work produced, a better solution for the scheduling problem would be a combination of centralized an decentralized solutions. Inspiring in the commonly used time windows solution [18], that uses the shortest route rule to generate a initial solution and then using time windows adapts the solution to where no two robots are in the same arc of the graph at the same time.

An A* is created for the Scheduling, to generate the task to be executed. When an node is generated in the Scheduling A*, a Routing A* is used to calculate the path and cost to that Scheduling node, as seen in diagram in figure 8.2.

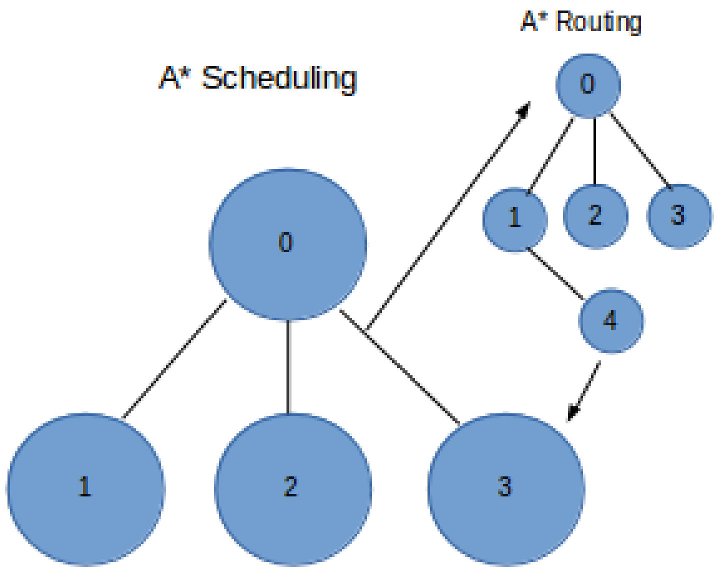


Figure 8.2: A* combining Scheduling and Routing

This solution would create very few nodes compared to the implemented solution. The result might not be as optimal, but still would be a close to optimal solution. As the robot would always take the shortest path, and the Scheduling part would take in consideration the machining time for each part.

Instead of using the time windows to manage situations where two robots were schedule to the same node or arc of the graph at the same time, a decentralized online solution would be implemented. Several zone control algorithm would be used to manage each of the critical area.

This solution could be used for scheduling of complex problems.

8.2.5 Battery Management

Although the rules do not account for the need to recharge or replace the AGVs' batteries, were this a real FMS, the implementation of such a system would be a must.

References

[1] J. Zygmont. Guided vehicles set manufacturing in motion.

[2] Muhammad Hafidz Fazli Bin Md Fauadi, Saifudin Hafiz Yahaya, and Tomohiro Murata. Intelligent combinatorial auctions of decentralized task assignment for AGV with multiple loading capacity. *IEEJ Transactions on Electrical and Electronic Engineering*, 8(4):371–379, 2013. doi:10.1002/tee.21868.

[3] DB Poudel. Coordinating Hundreds of Cooperative, Autonomous Robots in a Warehouse. pages 1–13, 2013. URL: http://www.academia.edu/download/30491528/seminarpaper.pdf.

[4] Peter R Wurman, Raffaello D Andrea, and Mick Mountz. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. 29(1):9–20, 2008.

[5] A. Madureira, I. Pereira, P. Pereira, and A. Abraham. Negotiation mechanism for self-organized scheduling system with collective intelligence. *Neurocomputing*, 132:97–110, May 2014. URL: http://www.sciencedirect.com/science/article/pii/S0925231213010928, doi:10.1016/j.neucom.2013.10.032.

[6] Robot@Factory - Competition rules and technical specifications.

[7] Tharma Ganesharajah, Nicholas G Hall, and Chelliah Sriskandarajah. Design and operational issues in AGV-served manufacturing systems. *Annals of Operations Research*, 76:109–154, 1998. doi:10.1023/A:1018936219150.

[8] A Paulo Moreira. AGV's in Flexible Production Systems.

[9] Donald E. Knuth. *The art of computer programming*. Addison-wesley Publishing Company, second edition, 1981. URL: http://profs.scienze.univr.it/~manca/storia-informatica/mmix.pdf.

[10] Vedran Bobanac and Stjepan Bogdan. Routing and scheduling in multi-AGV systems based on dynamic banker algorithm. *2008 Mediterranean Conference on Control and Automation - Conference Proceedings, MED'08*, pages 1168–1173, 2008. doi:10.1109/MED.2008.4602057.

[11] PATRICK J.M. MEERSMANS and ALBERT P.M. WAGELMANS. EFFECTIVE ALGORITHMS FOR INTEGRATED SCHEDULING OF HANDLING EQUIPMENT AT AUTOMATED CONTAINER TERMINALS. 2001.

[12] Anis Gharbi and Mohamed Haouari. Minimizing makespan on parallel machines subject to release dates and delivery times. *Journal of Scheduling*, 5(4):329–355, July 2002. URL: http://doi.wiley.com/10.1002/jos.103, doi:10.1002/jos.103.

[13] Eugénio Oliveira. Slides de apoio á unidade Curricular Inteligência Artificial.

[14] Mustafa Soylu, Nur E Özdemirel, and Sinan Kayaligil. A self-organizing neural network approach for the single AGV routing problem. *European Journal of Operational Research*, 121(1):124–137, February 2000. URL: http://www.sciencedirect.com/science/article/pii/S0377221799000326, doi:http://dx.doi.org/10.1016/S0377-2217(99)00032-6.

[15] Xiaokun Chang, Ming Dong, and Dong Yang. Multi-objective real-time dispatching for integrated delivery in a Fab using GA based simulation optimization. *Journal of Manufacturing Systems*, 32(4):741–751, October 2013. URL: http://www.sciencedirect.com/science/article/pii/S0278612513000873, doi:http://dx.doi.org/10.1016/j.jmsy.2013.07.001.

[16] Hamed Fazlollahtabar and Mohammad Saidi-Mehrabad. Methodologies to Optimize Automated Guided Vehicle Scheduling and Routing Problems: A Review Study. *Journal of Intelligent & Robotic Systems*, December 2013. URL: http://link.springer.com/10.1007/s10846-013-0003-8, doi:10.1007/s10846-013-0003-8.

[17] B. Turchianod M. P. Fanti. Deadlock Avoidance in Automated Guided Vehicle Systems. *Proc of 2001 IEEE/ASME Int'l Conf on Advanced Intelligent Mechatronics*, pages 1017–1022, 2001.

[18] Nenad Smolic-Rocak, Stjepan Bogdan, Zdenko Kovacic, and Tamara Petrovic. Time windows based dynamic routing in multi-AGV systems. *IEEE Transactions on Automation Science and Engineering*, 7(1):151–155, 2010. doi:10.1109/TASE.2009.2016350.

[19] Nuno Moreira. Localização de Robôs Autónomos em Ambiente Industrial. 2014.

[20] Paulo José Costa, Nuno Moreira, Daniel Campos, José Gonçalves, José Lima, and Pedro Luís Costa. Localizacao e Navegacao de um Robô Móvel Omnidirecional: Caso de Estudo da Competicao¸ ~ Robot@Factory. *VAEP-RITA*, 3, 2015.

[21] André S. Conceição, António P. Moreira, and Paulo Costa. Controller optimization and modelling of an omni-directional mobile robot. *IFAC*, 2006. URL: http://medcontent.metapress.com/index/A65RM03P4874243N.pdf$\delimiter"026E30F$nhttp://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:CONTROLLER+OPTIMIZATION+AND+MODELLING+OF+AN+OMNI-DIRECTIONAL+MOBILE+ROBOT#2.