Development and Operation of Autonomous Guided Vehicle

A Project Report submitted by

Milán Balázs
Meenal Malik
Jordi Gonzalez
Flemming Juel Jensen
Rasmus Nielsen

under the supervision of

Matteo Fumagalli

Aalborg University
Faculty of Engineering and Science
The Study Board of Industry and Global Business Development
A.C. Meyers Vænge 15
DK-2450 Copenhagen

Department of Materials and Production
A.C. Meyers Vænge 15
DK-2450 Copenhagen
http://mp.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Development and Operation of an Autonomous Guided Vehicle

Theme:
Semester Project

Project Period:
Autumn Semester 2018

Project Group:
ASGR5

Participant(s):
Milán Balázs
Meenal Malik
Jordi Gonzalez
Flemming Juel Jensen
Rasmus Nielsen

Supervisor(s):
Matteo Fumagalli

Page Numbers: 63

Date of Completion:
January 25, 2019

Abstract:

In this project, an Autonomous Guided Vehicle (AGV) is developed with the functionality to collect and transport components from multiple workstations to a warehouse for assembly. It is achieved using a differential drive module. The AGV is composed of a Raspberry Pi 3B, calculating the path and an Arduino Mega 2560, controlling the motion. Location is provided by VICON optical motion capture system, while the AGV is equipped with encoders to track two wheel velocities, further assisting the positioning and orientation of the AGV. The routing strategy used is built on a logic of routing and Dijkstra's algorithm, satiating the objective of minimising the time required to transport the components in the warehouse. The built components undergo a quality check step after assembly. In case of a defective product, the AGV recalculates its path based on the rescheduled pick-up order including the components required to replace the defective product.

Contents

Preface

Development and Operation of an Autonomous Guided Vehicle

Aalborg University, January 25, 2019

Milán Balázs
<mbalaz15@student.aau.dk>

Jordi Gonzalez
<jgonza18@student.aau.dk>

Flemming Juel Jensen
<fjense18@student.aau.dk>

Meenal Malik
<mmalik18@student.aau.dk>

Rasmus Nielsen
<reni15@student.aau.dk>

Chapter 1

Introduction

The revolution of industrial sector has brought a lot of technologies with it, especially within the manufacturing and operation of products. But, the thing that it lacked at the very beginning was a proper management of these products that needed to be transported from the warehouse to different stations. Previously, manual labour was employed to do stuff like gathering the manufactured products and further placing them for delivery to different depots. Then, the invention of special conveyor belts for collecting the components and integrating them at the assembly station came up. The problem with these machineries were:

- It was difficult to relocate the conveyor systems or replace them, in case they were damaged, making them intransigent.
- These systems also proved to be inaccurate & inefficient in a real industry world. Also, manual labour can get affected by the external environment, for example, extreme temperature, pressure variations, harmful gases and sharp objects which might harm them in a serious way.
- They were unsuitable for various environments, thus needing frequent repair demands, making it difficult to manage on daily basis.
- They got exhausted after a while because of the non-uniform pace.
- Also, manual labour proves to be much more expensive in the long run.

That's when Autonomous Guided Vehicles(AGV) came into existence.

Autonomous Guided Vehicles are the mobile robots which can manoeuvre over a desired path that it is instructed to follow autonomously, meaning, once it has been instructed, there is no further need of human intervention as it can perform its task independently with excellent efficiency and accuracy.

AGVs proved to be a much better solution to the problems that were faced previously in any industry as they are quite flexible, have remarkable efficiency & accuracy, are safe for almost all sorts of environment and are very productive.

Automated Guided Vehicles (AGV) have been applied for the flexible manufacturing system. Many factories adopted it into assembly line or production line such as automobile, food processing, wood working, and other processes. Many researchers developed and designed different sorts of AGVs in order to suit their applications which are related to the main problem of factory. Automatic Guided Vehicle (AGV) was firstly developed by [1] and [2] in the attempt of using at Jumbo Truck Manufacturing in Thailand. On the past of developed AGVs, we surveyed several papers concerning the design and control aspects as following [3] proposed the architecture of AGV with two wheels driven by differential gear drive and parallel linkage steering. For entire Flexible Manufacturing Systems (FMS), [4] proposed the operation control method by using two AGVs system. They solved the problem of scheduling method of AGVs model based on Petri nets. The objective of optimization model is the minimization of the total distance travelled by vehicles to transport the components to warehouse for product assembly. The route planning of AGVs in FMS was proposed by [5], [6] and [7]. [5] presented the new approach for dynamics route planning and scheduling problem of AGVs. They applied the search algorithm and some heuristic rules to solve the route assignment in dynamic situations. [6] also proposed the path planning strategy of AGV for navigation, collision avoidance and docking to the target. The AGV control approach was the important part for controlling the AGV actions. [8] applied the variable structure system techniques. The AGV was modelled by using kinematics and dynamic system. The adaptive control of AGV is also proposed by [9]. The non-linearity of dynamic model was developed for motion generation. The intelligence of AGV was also worked on several methods. The integrate sensor and vision was applied for control AGV. [10] studied the intelligent path following and control for vision-based automated guided vehicle. They presented the control path following of AGV by vision control system, and multi-sensors were also applied in real time steering control. AGV is one of the significance of the present research trend. In industrial application, manufacturing factory brought the mobile vehicle to incorporate working with other machine in order to bring the automated manufacturing system. Many applications adopted the AGV in different tasks such as material handling system, AS/RS system, transportation system, etc.

In this project, we have worked on the development and operation of an AGV which is being localised with the help of VICON optical motion capture system. We have employed Dijkstra's shortest path and different routing algorithms to make it collect the components to assemble products at the warehouse in an optimal way. Then, we did trajectory planning to define the path that it should follow, and the control was provided by employing the encoders with the motors. It also has a feedback mechanism where it is fed by a false signal, from a quality checking terminal, if the product assembled is defective.

Chapter 2

Methodology

2.1 System Engineering Approach

The approach for developing the system is to establish requirements based on the project description. Hereby specifying these top level requirements and break them down to an item level requirements. This create the fundamental for choosing the components in which the system are to operate. These components are brought together in an assembly for which test can be run corresponding to the initial set requirements. This is an ongoing process to ensure that requirements are met during the development. The approach followed in the development of this system can be seen by figure 2.1, also known as the V-model.

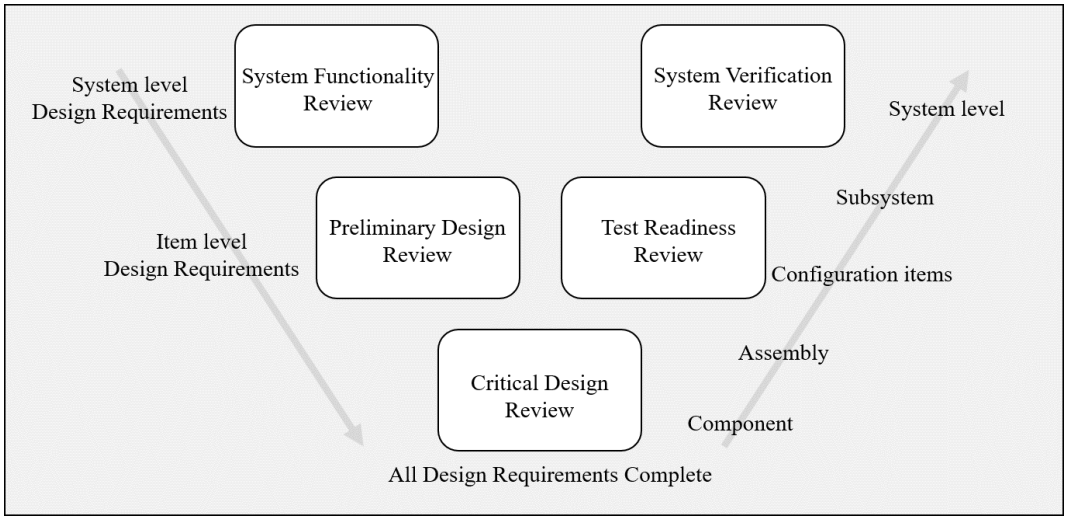


Figure 2.1: Developing a system, corresponding to a set of requirements are to follow the V-model for system design.

To further elaborate for the use of the V-model, the following list of activities are listed for each stage of the V-model to ensure a fulfilling development of the system:

System Functionality Review: The foundation for the design process is obtained through a requirement analysis. This analysis is based on the project description in which a set of requirements are listed for the top level system to perform. The requirements are divided in two segments; design and performance for which the system's ability to meet said requirements is evaluated. It's worth noting that since the project is a simulation of a real world application, this may prove to have some constraints. The system is then broken down into functionalities using the tool "IDEF0" (see appendix A) to determine the necessary processes to achieve set chosen requirements.

Preliminary Design Review: During the functionality process, a preliminary design is put together from the necessary mechanisms of each function. This proves as the initial stage of finding components which are capable of performing the required operations. The design of the framework allowing these components to connect physically are also taking place, meaning an evaluation of the necessary subcomponents. Furthermore the structure of the software elements are being developed.

Critical Design Review: From the initial list of components, each component is selected and documented in terms of its functionality. Arguments are made based on the capability to perform the task for each component, while the connections and communication in between are mapped. This creates the framework of the physical system and the capability of the system as a whole. As so, the software structure is realised by making the integration of scripts and middleware.

Test Readiness Review: Before deploying the system to verify it meets the requirements, each subsystem is tested to the corresponding functionalities set in the design phase. It's evaluated how both software and hardware respond to the tasks and adjusting measurements are done to increase the performance of the system.

System Verification Review: From testing the subsystems, the complete system is now tested corresponding to the top level requirements set in the system functionality review. This is to verify that the system performs according to the set requirements and therefore fulfil the project in terms of the framework presented in the description.

These steps create the approach for the completion of the system. Again, it's important to emphasise that there are loops between the design and testing stage. Due to the scale of this project, some limitations are set for the tests, meaning that each individual item won't be tested, but values provided by the supplier are taken as valid.

Chapter 3

Problem Analysis

The requirement analysis aims to provide a basis for decision making in the design process of the system's development. The output of the analysis should make the task of identifying functionalities and components clear and transparent.

Based on a set of inputs, high-level functionalities needed to complete the task are identified and performance measures are defined, in a manner that makes them verifiable. Subsequently, a functionality requirement analysis is performed, breaking the high-level functionalities lower levels of abstraction and linking the performance requirements to these low-level functionalities.

The design of the system will then be based on identifying specific technologies and methods, that are able to provide the required functionalities, capable of meeting the performance requirements.

3.1 Project inputs

Task (Customer) Requirements

The objective of the assignment is the development of an AGV that can operate in a simulated warehouse scenario. A production schedule is provided along with the Bill of Material (BoM) of each product. There exist four products comprised of six components. The task of the system is to move between the storage area and the manufacturing facility, delivering the required components to the assembly station (see figure 3.1). After assembly, the finished product undergoes a quality inspection. If the product does not comply with the certain specification, a new product of the same type has to be assembled immediately.

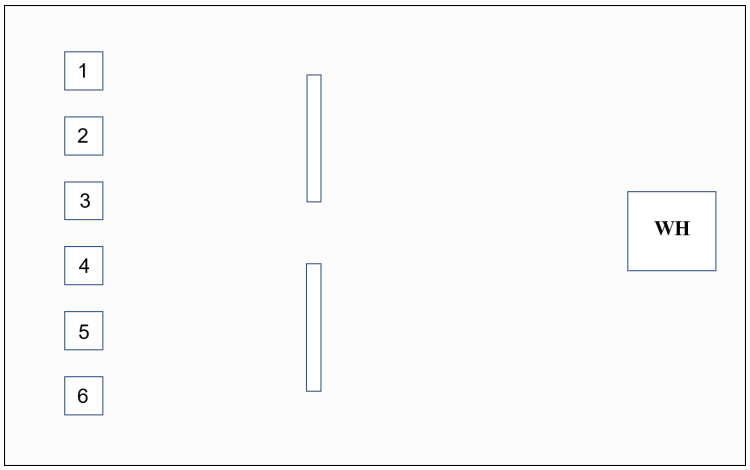


Figure 3.1: The warehouse that the AGV is to operate in.

The production time of a product is 15 s/product and quality inspection takes 5 s/product. Since both operations are carried out by the same operator, the effective cycle time is 20 s/product. The assembly station has room for a stock of components, at most three of each kind. The carrying capacity of the AGV is limited to two components. The respective position of each component stock and the assembly station is assumed to be known in advance.

The system will be evaluated based on its ability to complete the production schedule correctly and meet the cycle time.

Project constraints

The project is set to last 3 months. Funding of the project is limited, meaning pricey industrial grade components will not be an option. A number of components are already available. Some of these component might be used for the AGV over better components, to avoid delays to the project caused by delivery times. The following components are already available:

1 x Raspberry Pi 3B+
1 x Arduino Mega 2560
1 x H-bridge (TB6612FNG) motor driver
1 x 1000mAh 2-cell LiPo battery pack
2 x 6V brushed DC gearmotors
2 x Magnetic encoders
2 x Wheels

De-limitations

The focus in this project is to develop an AGV capable of planning and executing an optimal path for component delivery to the assembly station. Due to the duration of the project some functionalities will be occluded:

- Pick-and-place operations are outside the scope of this project and will not be considered further in this project.
- The development of self-localisation capability is not the interest of this project and will be replaced with VICON, a Motion Capture system covering the full operation environment.

The focus of the researchers is to identify and implement funtionalities capable of carrying out the task at hand. Analysis of hardware with regards to designing a system that is actually capable of meeting all requirements will therefore be superficial, since excessive testing is considered to time consuming.

3.2 Requirement Analysis

Based on the project inputs, specific requirements of the system, with respect to performance and design should be identified. Also, a set of functionalities that enables the system to carry out the operations needed in order to deliver all the components to the assembly stations should be defined.

The design and performance requirements is presented below followed by the functional breakdown:

Design Requirements

- The cross-section of the AGV must be smaller than 20cm, such that the robot can pass through the middle passage of the wall.
- 3D markers for VICON to identify an object requires a separation distance of 100mm.
- The battery must have the capacity to complete the entire of the production schedule.

Performance Requirements

- The length of the path for picking all components for a product should be minimized, such that the AGV bottleneck effect is diminished.
- The position error of the robot has to be less than or equal to 1) 5cm such that its center is within the component zone, 2) 20cm - cross section in order to pass through the middle passage of the wall.

Constraints

- Carrying capacity is 2
- Capacity for each component at assembly station 3

3.2.1 Functional requirements

In order to retrieve components, in a specific order, from specified locations and delivering them at the assembly station, the system requires the following 3 high-level functionalities:

1. Scheduling
2. Navigation
3. Pick-and-place

Scheduling is the process of planning the order of execution. The scheduling should pick the optimal order in which to pick up components, with the purpose of minimizing traveling distance and meeting the cycle time. This includes considering whether to include components of the next products in the current run, re-planning in case of failed QC and utilizing the buffer at the assembly station.

Navigation is the process of self-localization in the operational environment, planning a path to a desired destination and following that path. Navigation allows the AGV to execute the plan made by scheduling.

Pick-and-place is the ability of the robot to change the position of an external object, such as a component. As mentioned in the constraint considerations, this functionality will not be considered further in this analysis.

3.3 Functional Analysis

The purpose of the functional analysis is to break down the high-level functionalities identified in the previous section, into more specific sub-functionalities, that can be considered as a direct input to the design process. After the sub-functionalities has been established, the performance requirements will be allocated to the parts that directly impacts the system's ability to meet these requirements. Figure 3.2 visualizes the break-down of the high-level functionalities, which will be described in the section below.

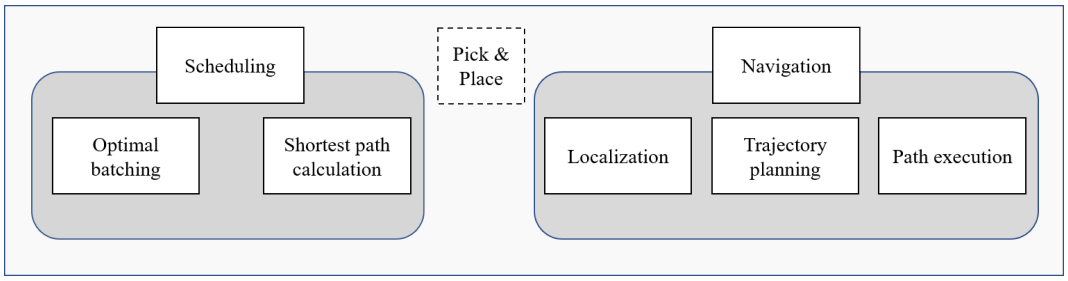


Figure 3.2: Overview of the top-level functionalities and their underlying functionalities.

3.3.1 Scheduling

Optimal batching

Since the carrying capacity of the AGV is limited to 2 components and the warehouse is in another area, the route must be broken down into runs. Thus the AGV goes to the storage area, picks up 2 components and deliver them to the assembly station. Therefore, considering the optimal combination of components is vital to the performance of the system.

The algorithm requires the result of the quality control, such that it, based on the production list and the Bill of Material (BoM) can derive a set of components that comprises the next products to be build.

The current warehouse inventory and robot inventory provides the algorithm with knowledge of which products are already on at the assembly station and on the robot. Knowledge of the warehouse inventory also allows the system place new items in the buffer. Furthermore, the global map allows the algorithm to compute the distances between the components' storage locations, such that it will not combine two components with great distance between them.

Optimal batching should provide an order picking or a batch list, containing the optimized combination of components to be picked up in a run.

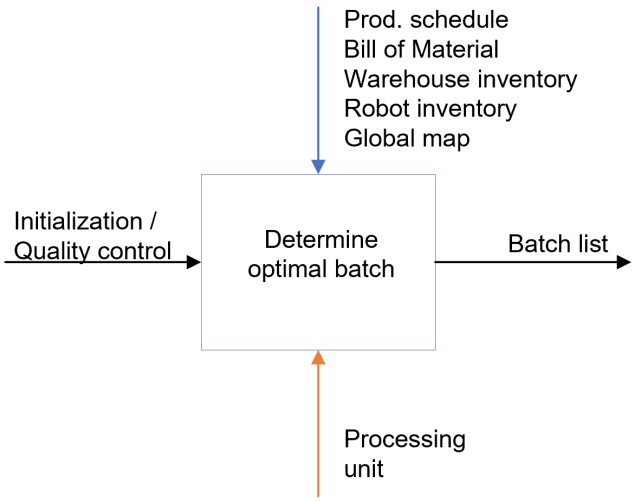


Figure 3.3: The functionality for determining the optimal batch

Physical components: Processing Unit

Interfaces			
Inputs	Source	Outputs	Sink
Initialization / Quality control	Operator / Production system	Order picking / batch list	Shortest path calculation

Figure 3.4: Table of interfaces for determining the batch

Shortest Path Calculation

In order to be time efficient, it is necessary to consider the shortest path from A to B in the warehouse environment.

Shortest route computation requires the target locations, which is acquired from the batch list. Based on the global map and the current position of the system, it can calculate the shortest path that passes through all of the location for a single run. Shortest route computation should provide a list of position that comprises the shortest path through the environment, arriving at the required component location and ending at the assembly station.

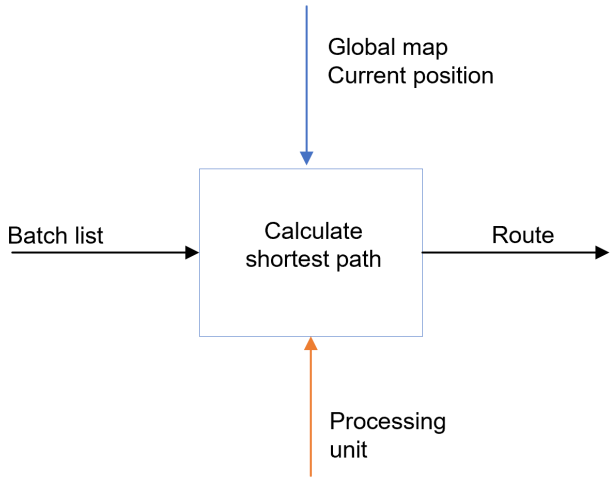


Figure 3.5: The functionality for calculating shortest path

Physical components: Processing Unit

Interfaces			
Inputs	Source	Outputs	Sink
Batch list	Optimal Batching	Optimal Route	Trajectory planning

Figure 3.6: Table of interfaces for shortest path calculation

3.3.2 Navigation

Localization

Localization is a key functionality in virtually all automated navigation systems and it is, more often than not, the performance of the localization that has the largest effect of the overall performance of the system. However, as mentioned in the project constraints, an external sensor system, VICON, will be used for localization. Thus, this section will not go into much more detail regarding the localization, since as mentioned, it is an external system providing millimeter accuracy. VICON is considered as an internal functionality, however the only action taking place is the communication with an external server to retrieve the data provided by VICON.

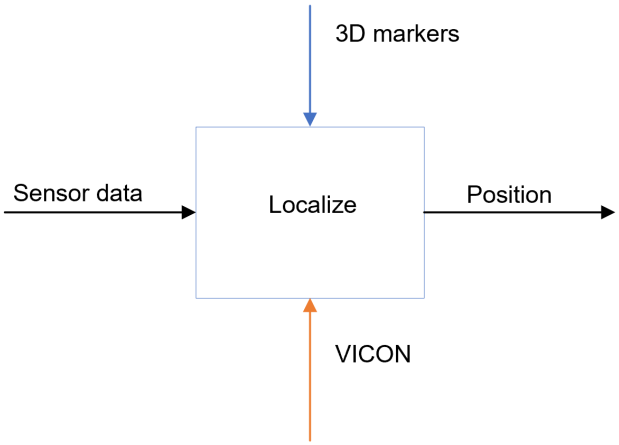


Figure 3.7: The functionality for localization.

Physical components: External Sensor(s)

Trajectory planning

Simply knowing the current and desired position of a system is not on its own enough to actually get there. The configuration of actuators in the system, result in a number of movement constraint. The way a system is able to move is referred to as the steering behavior of the system. In order to make an AGV move from point

A to point B, the path has to be broken down into a set of discrete movements, that, when executed consequently will translate the system to the desired end destination.

Given the route and current position, the trajectory planner should create a set of motion commands, based on the steering behavior of the system.

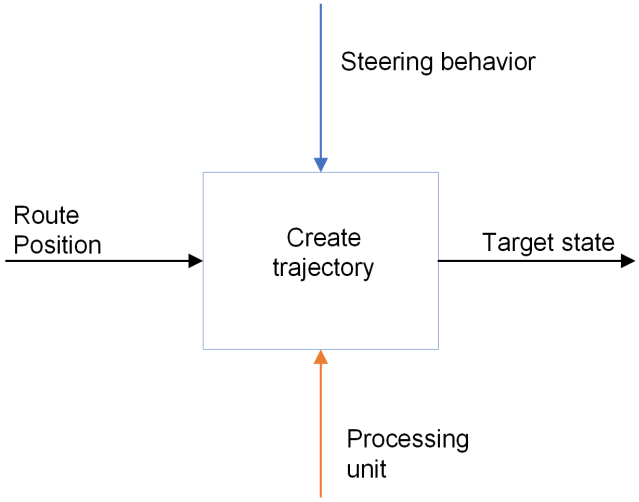


Figure 3.8: The functionality for creating trajectory

Physical components: Processing Unit

Interfaces			
Inputs	Source	Outputs	Sink
Position	Localization	Target state(s)	Path Execution
Route	Shortest path calculation		

Figure 3.9: Table of interfaces for creating the trajectory

Path execution

Movement is achieved through manipulation of actuators. Given a target state, the robot should be able to transform this input into a physical output. To carry out the behavior depicted by the trajectory planner, a controller is required to ensure that the actuators creates the desired change of the system. Commonly in complex systems, this requires a closed-loop controller, since too many variables impact the system to accurately predict the exact change caused by an actuation signal.

The input to the path execution is the target state, which based on a mathematical model of the system is converted to a control signal.

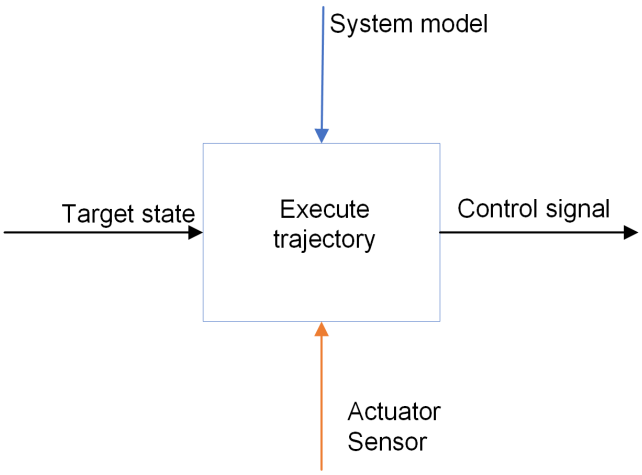


Figure 3.10: The functionality for executing the trajectory

Physical components: Processing unit, Actuator(s), Sensor(s).

Interfaces			
Inputs	Source	Outputs	Sink
Target state	Create trajectory	Control signal	-

Figure 3.11: Table of interfaces for executing the trajectory

3.3.3 System diagram

The functional architecture gives an overview of which processes are to be completed by the system, which input they receive, output they deliver and under which parameters that control the process. The physical components of the system will consist of a processing unit taking care of calculation, transmitting and receiving data. Furthermore, sensors are to be used to obtain necessary input and actuators to execute the output.

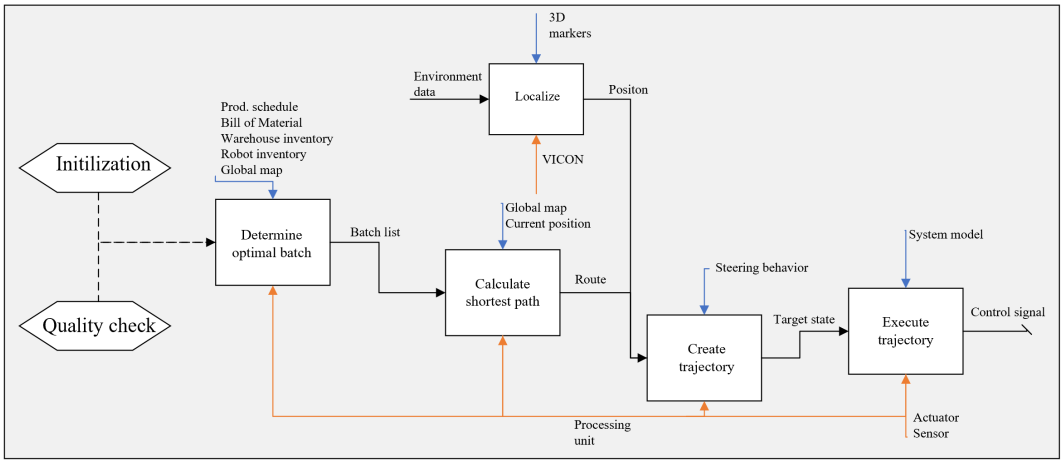


Figure 3.12: Overview of the interconnections of all functionalities for the system.

3.3.4 Physical View

Processing Unit

From a functional view it is clear that multiple calculations should be made during runtime, meaning that the system would need an on-board processing unit or a micro controller capable of wireless communication. Replanning in case of QC fail, path planning and execution is all real-time application, that can only be accomplished through a sense-plan-act cycle.

Actuators

The system will need at least two motors to move around in the warehouse environment. The motors should be controlled by the path execution functionality.

Sensors

The localization and path execution functionalities both requires real-time information of the system state through sensor readings. They can potentially rely on the same sensor, however, in order to meet the performance requirements, multiple sensors might have to be used. The precision of the systems movement is dependent on the information provided. The sensors used for path execution should be precise enough that it is possible to correct the actuation signals to decrease positional error in order to comply with performance requirements.

Power Source

Both actuators, sensors and a processing unit requires a power source. Since the system is mobile, so should the power source be. Due to the limit in size, the power source should also has a cross-section that is smaller than 20cm and a weight that

doesn't block the motors. Furthermore, the power source should have a capacity that can provide power to the system in the duration that is has to operate.

Structural Elements

The structural elements of the system should be large enough to carry all hardware components, but the cross-section should be less than 20cm.

Operator Interface

The interface should allow the operator to provide the production schedule as well as report the results of the quality control. Since the robot is moving at all times doing production, this interface should not be located on the robot. This requires an external interface with the robot, which should be wireless. Wires would limit the robots ability to move freely, and make the navigation process unnecessarily complicated compared to the relatively low cost of setting up a wireless connection.

Chapter 4

Design

The design of the AGV consist of hardware and software integration to make up the system corresponding to the requirements set. The different functionalities to achieve this can be seen by figure 4.1, to which this chapter contains sections. The *physical design* cover the hardware in order for the AGV to interact with the world, while the *scheduling* is to decide which components to collect to optimise the assembly of products. The trajectory for doing so, is compiled in the *trajectory planning* leading to the execution by the *control*. Lastly these functionalities come together in the *integration* using the middleware ROS.

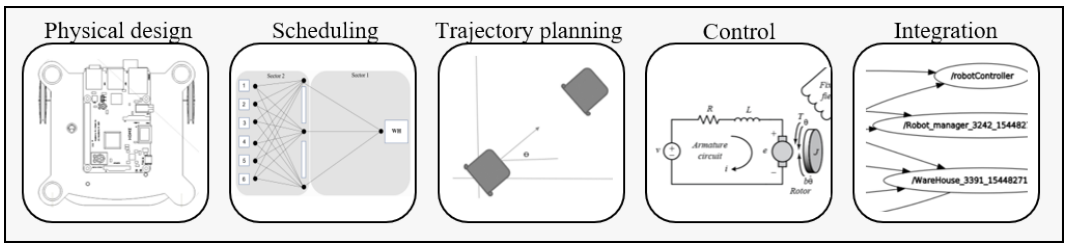


Figure 4.1: The different design aspects of the AGV.

4.1 Physical Design

To perform the functionalities while meeting the requirements defined in the analysis, the following four mechanisms have been addressed; Processing units, VICON, actuators and sensors. According to the list of components mentioned in the project constraints, all these functionalities are covered, but also allow for the option to choose between one or another. Hence, in this section it'll be discussed which component to use, why they are used and how they are integrated with one another.

4.1.1 Processing unit

The processing unit mentioned in the functionality analysis makes it ideal to use either the Raspberry Pi, the Arduino or both. Simplified, a Raspberry Pi is a microprocessor working as a computer whereas the Arduino is microcontroller, only capable of performing a single task at a time. This implies there are some limitations of using an Arduino as driving an autonomous robot wouldn't be able to utilize middleware such as ROS and the communication from VICON would require additional components.

With this in mind a Raspberry Pi is selected for a number of reasons:

1. It has wireless communication capabilities, which enable data transmission from VICON.
2. It can run multiple processes simultaneously, which is required to both receive data from VICON while outputting signals to the motors.
3. It allows for the utilization of middleware

However this does not exclude the use of an Arduino entirely.

As the Raspberry Pi is running middleware, filesystem, network, and other drivers, it's latency builds up and thereby delay response. This is due to all the queued processes, which goes thread by thread as per the priority scheduler for the Raspberry Pis CPU. In comparison the Arduino runs its sketch with its CPU all to itself, because no context switching can take place until, and unless, an ISR is called. Moreover, the interrupt latency of Raspberry Pi is higher than that of Arduino. Upon an interrupt reception via ISR, Arduino reacts to it faster as it has a pointer pointing at the interrupt look-up table, whereas the Raspberry Pi must run the application to see the interrupt. Raspberry Pi might sometimes just miss the interrupts if the application is not run properly. This is very unfavourable while using system interface to connect with GPIOs. Thus, the Arduino will be much more responsive to interrupts than the Raspberry Pi with less timing jitter [11]. Thereby, combining both Arduino and Raspberry Pi is the best solution for complex implementation, thus leveraging both their capabilities. The connection between them can be made by using either I2C or SPI link, which implies that the timing-sensitive pulse counting, and motor control can be done at the Arduino, and Raspberry Pi simply gives the instructions to the Arduino where to go next, making all the high-level decisions.

4.1.2 Actuator

The motors used are two 6V brushed DC gearmotors with the following specifications:

Gear ratio:	150.58:1
No-load speed @ 6V:	220 rpm
No-load current @ 6V:	0.10 A
Stall current @ 6V:	1.5 A
Stall torque @ 6V:	2.0 kg·cm
Max output power @ 6V:	1.1 W
Extended motor shaft?:	N
Long-life carbon brushes?:	Y
Motor type:	1.5A stall @ 6V (HPCB 6V - carbon brush)

Figure 4.2: Specifications as per the supplier (https://www.pololu.com/product/3066/specs).

Brushed DC motors provide inexpensive, lightweight actuation with a reasonably efficiency. Additionally these motors offers an extended back shaft to optionalize the addition of a magnetic encoder to the gearmotor in order to measure the rotational output, hence provide motor speed and position feedback.

4.1.3 Encoders

Two rotary magnetic encoders are used with a count per revolution of 12 to provide motor speed and position feedback in addition to VICON. Using the 3D markers VICON provide translation and orientation of the AGV with a system error of less than 2 mm for a dynamic object [12]. Hence it is optimal for localization. However in terms of control the encoders provide feedback at 12 count per revolution, assuming the motor is doing 170RPM at the gear ratio of 150.58 that is 5,12kHz compared to VICONs 100 Hz.

This allows for precise wheel velocities and a lower response time. The advantage of using encoders are high resolution, reliability and accuracy. It's important to note that though encoders would prove more precise for velocity measurements, they do not account for slippage or inconsistencies in the wheels to why VICON is utilized to verify when a given position is reached.

4.1.4 Battery

To power the AGV a 1000mAh 2-cell LiPo battery pack is used. Given the initial requirement of 120% x no. of products x 15 seconds, the battery must hold capacity to power the AGV during this time period, while powering both the motors, Arduino and Raspberry Pi. The dimensions of 72x34x14mm makes it an ideal power source for an AGV, considering it's mobility.

4.1.5 Assembly of components

The previous discussed component are as followed:

1. Raspberry Pi 3B+
2. Arduino Mega 2560
3. H-bridge (TB6612FNG) motor driver
4. 2 x 6V brushed DC gearmotors
5. 2 x Magnetic encoders
6. 1000mAh 2-cell LiPo battery pack

In addition to the above components, the following subcomponents are used: 1 x caster wheel, 2 x 40mm diameter wheels, 1 x switch and 3 x 3D VICON markers. These components form the AGV corresponding to the functional architecture while meeting the design requirements set in the analysis. The following three design requirements were set accordingly:

1. Cross-section size of the AGV is to be lower than 200 millimeters.
2. The 3D markers placed on the AGV needs a separation distance of 100 millimeters.
3. Battery capacity must be at least equivalent to 30 minutes of runtime.

It is already theoretically proven that the battery have the necessary capacity to meet the requirement, the assembly will take the first two requirements into account. To meet these design requirements the framework of the AGV is to be two layered to store all components. This is to achieve the two design requirements, as the size would otherwise exceed 200 millimeters cross section and the other component may cause a disturbance for the 3D markers.

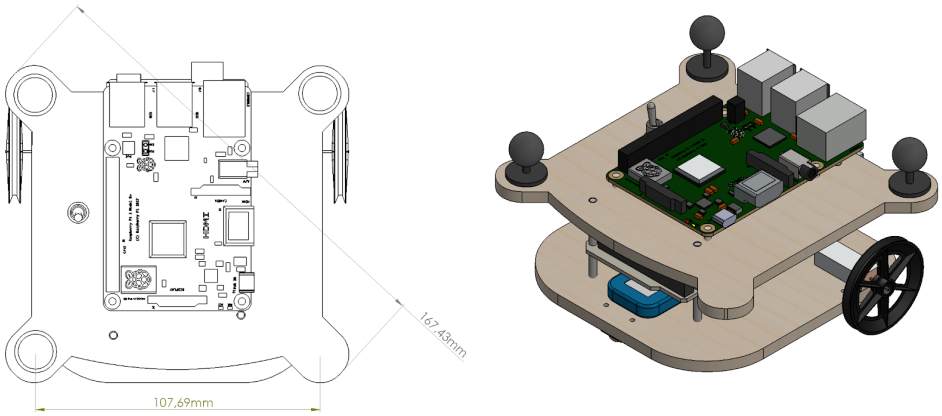


Figure 4.3: Top view of the AGV giving both distance separating 3D markers and the cross section.

As seen by figure 4.3 the cross section of the AGV is 167 millimeters and thereby within the allowed size. Furthermore is the distance in between the 3D markers 107 millimeters, also fulfilling the second requirement. It is to be noted that the three markers are placed asymmetric with varying altitude for VICON to establish a stable connection with the AGV as an object. Mentioned earlier the Raspberry Pi is to do the computing while the Arduino does the controlling, thus the wiring will primary go to the Arduino. As shown by figure 4.4, it's therefore located in between the frames, creating a spacious environment for this.

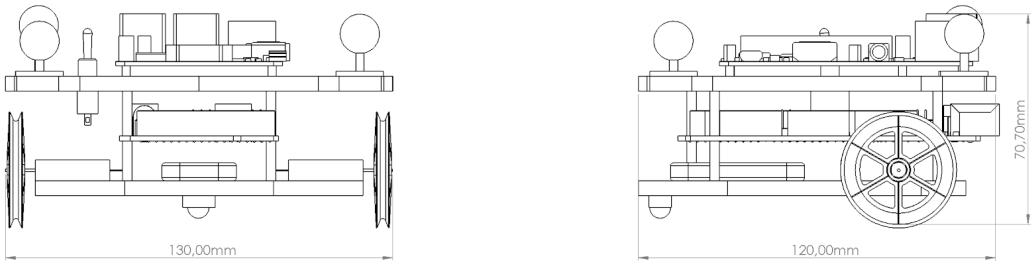


Figure 4.4: Width and length of the AGV physical assembly.

In order to keep the 3D markers free of obstacles, only the Raspberry Pi is located on the top frame of the AGV. This ensures that the signal between the VICON system and the AGV is free of disturbance. The lower frame of the AGV holds the motors, H-bridge and battery, which are all connected by the breadboard. Furthermore is the AGV equipped with a caster wheel and two 40mm wheels to enable motion.

4.1.6 Connections

The powering of the Raspberry Pi and Arduino is completed in serial using the LiPo battery. To allow for a direct control of the motors a switch is connected in between the LiPo battery and the H-bridge. The connection between Raspberry Pi and Arduino is obtained through a USB to USB-A cable, whereas all other connections are done on the breadboard. Wiring can be seen by figure 4.5.

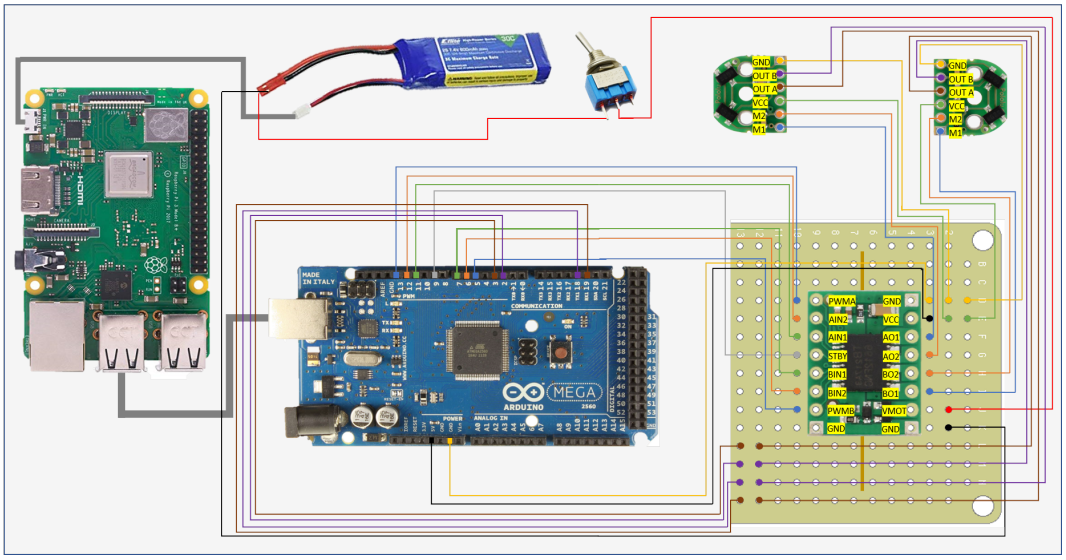


Figure 4.5: Diagram showing the connections between the components of the AGV.

This concludes the physical design of the AGV. Further sections cover the functionalities in terms of the software, thereby calculating, planning and execution of the code forming the *autonomous* part of the AGV.

4.2 Scheduling algorithm

4.2.1 Context of the problem

The Capacitated Vehicle Routing Problem, CVRP, consists on collecting/delivering a certain number of components from the customers without exceeding the vehicle and warehouse capacity. In the problem, the vehicle has to collect and deliver six different types of components (figure 4.6) placed in different places to the warehouse. The robot has a maximum capacity of two components and the warehouse has a maximum capacity of three components of each kind. Between the components and the warehouse there are two walls and the robot has to go in between these walls.

Product 1	Product 2	Product 3	Product 4
1 x C1 1 x C3 2 x C4	1 x C1 1 x C2 1 x C5 1 x C6	2 x C3 1 x C5	1 x C2 1 x C3 2 x C4

Figure 4.6: Combination of components to each product.

4.2.2 Mapping of the world

The representation of the map is done by using a graph. In this graph, the nodes represent the components, the warehouse and the middle points. The edges represent the path which the robot can follow between the two connected nodes. In Figure 4.7, there is a representation of the map with the graph on top.

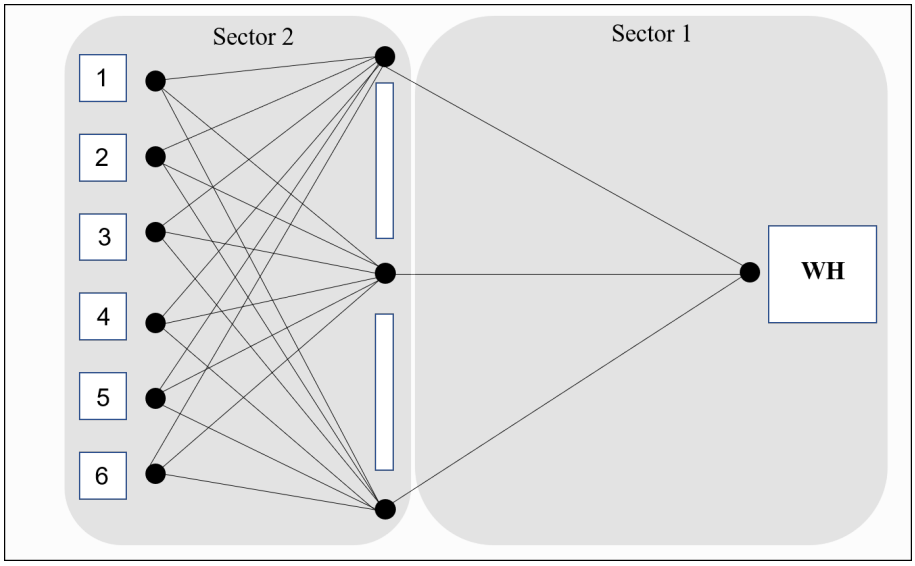


Figure 4.7: Mapping of the world illustration

In order to implement the graph, multiple classes have been set up with information stored corresponding to the use case of each task. In Figure 4.8, the class diagram of the routing system for the robot is shown. A class named product has the information to control and check if a component is from a corresponding product and to know the position of that component. This class is composed of components that are used to create the path that the robot has to follow.
To get all the information about the composition of the products, the position of the components along with the position of the middle points and the warehouse, an XML file is used. With these files, it is possible to change the information easily without changing anything from the source code.

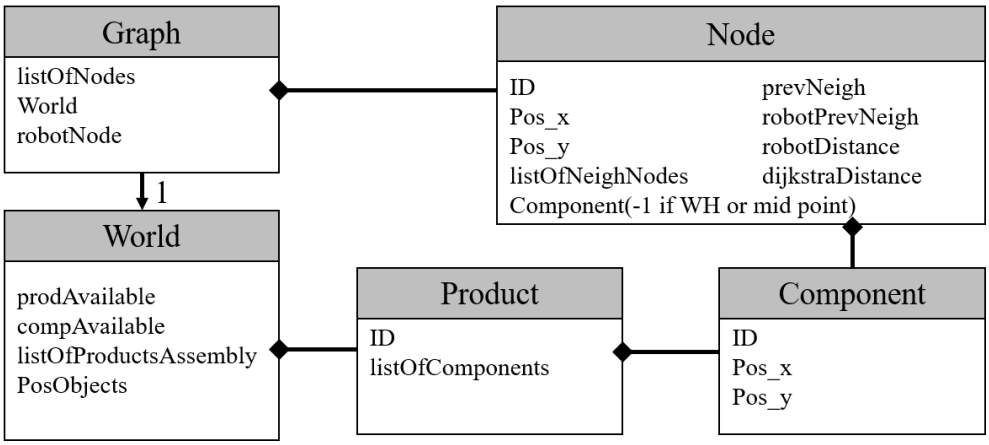


Figure 4.8: Class diagram routing

4.2.3 Optimal distances

To be able to get the optimal path, the optimal distances have to be calculated between the nodes. In order to calculate these distances, Dijkstra's algorithm have been used. This algorithm returns the shortest distance to reach a node and it also indicates which is the next closest neighbour.

There are two types of distances:

- **Static distances**: These distances never change as they are using the warehouse as reference.
- **Dynamic distances**: These distances always keep changing as they are using the robot position as reference.

This information is used while creating the path to go to a node either from the warehouse or from the robot, where the path is created by tracking the values of previous neighbour, which is the closest one, stored in the node class, as seen in Figure 4.8.
To compute the Dijkstra's algorithm, we have taken two different scenarios in order to compare if there is any difference in time:

- The first version calculates the distances from a node to all it's possible neighbours and once it's finished, it searches the closest node in the list of nodes, to calculate the next distance. In order to find the closest node, it needs to go through the whole list of nodes in the graph. And this is repeated until all the nodes have been explored.
- The second version, instead of searching the closest neighbour in the list, inserts all the neighbours from the actual node into a sorted list, sorted by the calculated distance. This is repeated until the sorted list is empty, which will mean that all the nodes have been visited.

4.2.4 Path calculation

Once the information is in the graph and the optimal distances are calculated, it is possible to calculate the path that the robot will need to follow in order to get all the components for the products demanded through a CSV file.
In order to calculate the path, two approximations have been developed:

- **Greedy algorithm**: This algorithm calculates all the paths that the robot has to follow from the first to the last product in the list.
- **Optimal solution with a window size of two**: In order to take into account the future products, a window size of two is used and the path is calculated for the next product, taking the following product into account.

Greedy algorithm

This algorithm finds a path by choosing the best local solution. In this problem, it always takes the closest, non-visited component to create the path. This is a way to get a feasible solution but it does not mean that it will always be the best solution to collect the components of a product. The advantage of using this algorithm is that it is very simple to develop, and fast in order to find a solution. This means that the path for all the demanded products can be created at time zero by making the assumption that there is no defective product, the path just needs to be calculated once, and if a product is defective, this algorithm is fast enough to recalculate a new path.

Optimal solution in a window size of two

In the previous solution, one of the problems that could appear is if the robot has to complete a journey in less than 15 seconds, it might have more than 3 components of the same type in the warehouse, which is not possible. In order to solve this problem, the algorithm uses a window size of two where it takes into consideration the following product and the components that are in the warehouse at every iteration, so that a new path can be calculated. To calculate the path with this algorithm, all the permutations of collecting the actual product and the following product are created and it takes the permutation of the actual product that minimises the distance to the following product. This is repeated until there are no more products in the list of products.

Path creation

When creating a path for a product, the starting position is not always the warehouse, as it can also be the last component of a product. In the second algorithm, it could be easily solved just by setting the starting point to the robot position, but this would mean that the path cannot be calculated until the robot reaches the last component of a product. Instead, when calculating the path for a product, it

is checked if the last component collected will fill the robot capacity, which is two components, or not. If not, then this means that the robot will start collecting the components for the following product from one of the components. This allows the calculation of all the paths by predicting where the robot will be in that moment.

4.2.5 Path Recalculation

When all the components of a product are delivered to the warehouse, the assembly of that product starts, which takes 15 seconds. Then, there is a quality check (QC) option, that can be positive implying that the product is assembled correctly, or negative, implying that the product has to be assembled again. If the QC is negative, all the components for that product have to be delivered to the warehouse again. While the assembly process and the QC, which lasts for 20 seconds, the robot is collecting the components for the following product. This means that, at the moment of the negative QC, the robot might need some components and some not or depending in which sector the robot is present of the map, shown in Figure 4.7. Furthermore where different decisions might have to be taken or that the warehouse already have some products that are needed for the defective product, which are not needed to be collected again.

All the possibilities that can be found are the followings:

- Robot has no components
- Robot has components
  - Robot is in sector 1
  - Robot is in sector 2
    - The robot has 1 component needed
    - The robot has 2 components needed
    - The robot has 0 components needed

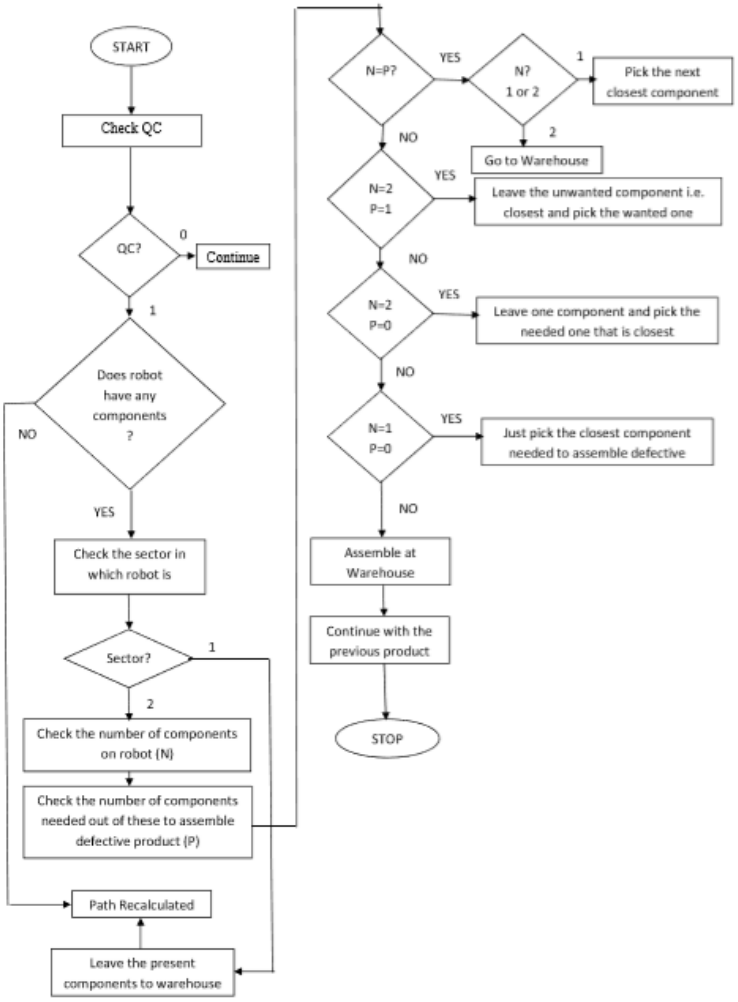


Figure 4.9: Flow of Quality Check Algorithm

After bringing all the components of the defective product, it will continue with the components of the product that was already in progress at the moment the QC was declared negative. In this case, it has to be checked which components are in the warehouse that will not be used by the defective product and then create the path for the remaining components.

The robot has no components

In this case, the robot might have collected some components for the product in progress and at the time of the negative QC, the robot was going to take more components to the product in progress. However, as the goal is to assemble the

first product as soon as possible, and now the first product is the defected one, the path is recalculated in order to collect all the components needed that are not in the warehouse.

The robot has components and is in section 1

If this situation happens, it will mean that the robot has two components that are needed for the product in progress and that is close to the warehouse, in sector 1. As the robot is close to the warehouse, this will leave the components to the warehouse and recalculate the path adding the defective product and taking into account the products that are in the warehouse to start the assembly of the defective product as soon as possible.

The robot is in section 2 and it has 1 component needed

When the robot is in this statement means that the robot can either has two components, one needed and the other one that has to be returned back to its place or the robot only has one component.

- Two components: In this case, the robot will leave the component that does not need and will collect the closest component, from its current position, that needs to assemble the defective product.
- One component: Here, the robot will collect the closest component needed from its current position.

The path will be recalculated adding the defective product again to the products list.

The robot is in section 2 and has 2 components needed

In this situation, the path is recalculated by adding the components needed to assemble the product and making sure that the product in progress might have some components in the warehouse that will not be used by the defective product. This path will start by bringing the components to the warehouse.

The robot is in section 2 and has 0 components needed

In this case, the robot might have either one component not needed or two components not needed in order to assemble the defective product.

- 1 component not needed: Here, the robot only has one component and is not needed, but instead of returning this component back to its position, which will take an amount of time, the robot will collect the closest component needed to assemble the defective product. The path will be calculated starting on the robot's position and taking into account the components that are already in the warehouse and the one in the robot.

- 2 components not needed: In this situation, the robot has no components needed to assemble the defective product and its capacity is full, so it can not collect any other component. So, it has been decided that the robot will leave the closest component to the current robot position and it will collect the closest component needed from the component that has been left.

4.3 Trajectory planning and control

Trajectories, made by concatenating straight motion and in place turning primitives, are the one that can be easily followed by a differential drive robot. This project presents trajectory tracking and control of differential drive robots along a predetermined route (the one calculated using routing algorithm) that the robot will follow in order to assemble components at the warehouse. A control algorithm was developed to control the robot along different trajectories.

The control algorithm of the robot is based on a multi-layer model. The bottom layer controls the mechanics to follow the robot's desired trajectory. The next higher layer calculates the trajectory, which is generally known as path planning or trajectory planning. The calculation of a trajectory is based on intersection points, which are the result of solving the generated tasks. Solving the tasks is done by the next layer, which is in itself a sub layer of the decision layer.

The path planning process could be predetermined or run time. The predetermined method builds the path before the next action is decided for the robot. In the run-time method, the path is built simultaneously along with the performance of any of the actions decided. For the outdoor environments, the run time method proves to be more precise considering the impromptu conditions. Even some minimal amount of noise in path planning can cause a considerably chaotic situation. The probabilistic approaches in mapping takes care for such noises and their counter effects on the robot.

4.3.1 Kinematics

The kinematic model of the robot allows the study of robot movement from one configuration to another without the consideration of forces acting upon the body. The robot configuration in the case of a differential drive robot can be characterised in a given instance with its position and its heading ($\mathrm{C} = \mathbb{R}^2 \times \mathbb{S}^1$). The robot position can be given by the x and y coordinates in an inertial frame of reference. The heading of the robot can be given as the angle (θ) between the x axis of the robot frame and the inertial frame.

$$q = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{4.1}$$

When considering the robot in the inertial frame, the reference point is the mid point of the common wheel axis (A). Different wheel velocities result in different robot motion. Rotation occurs when the two wheel velocities are not equal. In case of differential drive robots, beside the variable wheel velocities, the robot motion solely depends on the distance between the two wheels and the radius of the two wheels which are constant in most applications. The rate of change of the robot configuration in the robot frame (r) in point A is

$$\dot{x}_{\mathrm{A}}^{\mathrm{r}} = R\frac{(\dot{\phi}_{\mathrm{r}} + \dot{\phi}_{\mathrm{l}})}{2} \tag{4.2}$$

$$\dot{y}_{\mathrm{A}}^{\mathrm{r}} = 0 \tag{4.3}$$

$$\dot{\theta}_{\mathrm{A}}^{\mathrm{r}} = \omega_{\mathrm{A}}^{\mathrm{r}} = R\frac{(\dot{\phi}_{\mathrm{r}} + \dot{\phi}_{\mathrm{l}})}{L} \tag{4.4}$$

and in the inertial frame (I) in matrix form:

$$\dot{q}^{\mathrm{I}} = \begin{bmatrix} \dot{x}_{\mathrm{A}}^{\mathrm{I}} \\ \dot{y}_{\mathrm{A}}^{\mathrm{I}} \\ \dot{\theta}_{\mathrm{A}}^{\mathrm{I}} \end{bmatrix} = \begin{bmatrix} \frac{\mathrm{R}}{2}\cos\theta & \frac{\mathrm{R}}{2}\cos\theta \\ \frac{\mathrm{R}}{2}\sin\theta & \frac{\mathrm{R}}{2}\sin\theta \\ \frac{\mathrm{R}}{\mathrm{L}} & -\frac{\mathrm{R}}{\mathrm{L}} \end{bmatrix} \begin{bmatrix} \dot{\phi}_r \\ \dot{\phi}_l \end{bmatrix} \tag{4.5}$$

,where R is the wheel radius, $\dot{\phi}_l$ and $\dot{\phi}_r$ are the left and right wheel turning rates respectively. Due to the dynamic constraints of the robot, the robot only has linear velocity in one dimension perpendicular to the wheel axis, but when translated to the inertial frame, it can be decomposed into x and y components based on the robot heading.[13]

Collision free path

Working with a known static global map of the environment greatly helps with navigation. The information content of the map is used to measure distances between goal positions as well as to determine safe routes for the robot to take. In order to simplify the trajectory planning process, the robot is reduced to a point like object, and the map is changed according to a procedure called the Minkowski addition. In this process the obstacle space is dilated by the largest dimensions of the robot measured from the wheel axis centre point in all directions. In the resulting free space, way-points have been placed in such a manner that from all points on the map, the closest way-point can be connected with a straight line part of the free configuration space. This can be shown by connecting all intersections of each polygon to the way-point, thus dividing the polygon into triangles. Due to the

convexity of triangles, any point within the triangle can be connected to all of its vertices with a straight line. When navigating, convexity guarantees that from every configuration in the free space the shortest path to the nearest way point is collision free. The map with this way-point configuration is topologically identical to that of used in the scheduling algorithm.

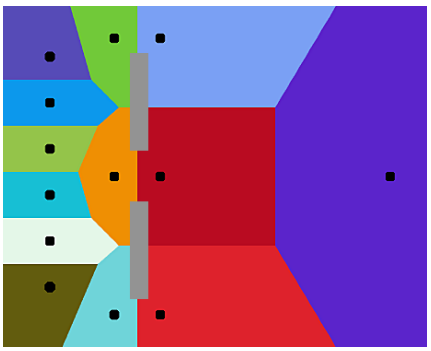


Figure 4.10: Voronoi diagram showing how way-points (marked with black) divide the configuration space into polygons. All points within a coloured polygon can be connected to the way-point within that specific cell with a straight line. This straight line is part of the free configuration space, therefore if the robot takes such path, it is guaranteed to be collision free.

4.3.2 Control Strategy

Based on the principles explained in 4.3.1, different control strategies for the system are presented. A strengths and weaknesses of a stop-and-go approach and a go-to-goal with regards to the warehouse scenario is discussed, after which one of them is selected as the position controller of the system.

The path planning should achieve the following two things: Avoid collisions with any obstacles in the warehouse and move between the assembly station and the storage area as fast as possible.

Based on 4.3.1, it can be assumed that only moving in straight lines between static way points would ensure a collision free path. The simplest control method to achieve this kind of behaviour is the turn-and-go approach, where the robot is turned in the direction of the destination and then goes forward until the target location is reached. The system is constrained to only one type of movement at a time: either translational or rotational. In theory, this control behaviour is the fastest way to move between two points, while being subjected to the aforementioned constraints, since it seeks to saturate the respective velocities (linear and angular).

However, in reality, the system is subjected to external disturbances such as ground friction. If such a disturbance causes the system to change its orientation doing linear movement, the constraints of the system would cause it to either stop and correct its orientation with respect to the desired track or simply accept that

its final position is going to be off-track. This can ultimately cause the system to be slow to perform certain movements, since it will stop very frequently to correct its position all the time or it might have some uncertainties regarding its final position.

An alternative to the turn-and-go approach is to control each velocity independently and simultaneously, which is possible by deploying go-to-goal approach. This approach controls the angular velocity based on the error in orientation, similar to the turn-and-go, combined with a linear velocity controller. The method for controlling linear velocity will vary based on the desired behaviour in a given application.

One way is simply to keep a constant linear velocity. This will, however, require linear velocity to be rather low, since otherwise the system would likely overshoot or in best case follow unnecessarily long paths. Consider the example of the robot having to make a large turn (> 90 °), in order to face towards the target position. If the linear velocity is high, while the robot is turning, this would cause the robot to drive on an overly long arc. Actually, having non-zero linear velocity when the system has to turn more than 90 ° would be unfeasible, since it would cause the robot to move further away from the goal position. This approach is not suitable for this system.

Another method is to control the linear velocity in the same way in which the angular velocity is controlled, the error being based on translational error rather than rotational error. However, this method is subject to the same limitations as having a constant linear velocity at large distances to the target position. .

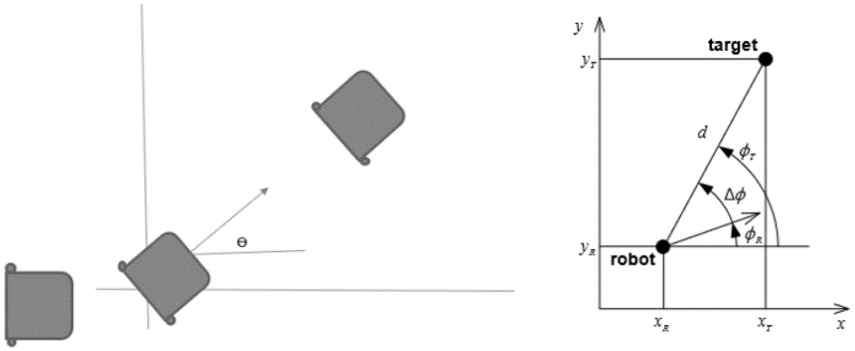
A method for controlling the linear velocity, that overcomes the aforementioned issues, is to make it dependent on the angular velocity. Since the angle is what will move the robot in the desired direction, the angular velocity should determine the magnitude of the linear velocity. An analogy can be made to a person driving a car. The "steering" consist of turning the steering wheel, not pressing the gas pedal. The relationship between the two velocity components should be inversely proportional, such that when the angular velocity is saturated, the linear velocity should be zero. The relationship is shown in figure 4.6.

$$\dot{x} = \dot{x}_{max} - \frac{\dot{\theta}}{\dot{\theta}_{max}} \dot{x}_{max} \tag{4.6}$$

If the gain of the controller is set such that the angular velocity is saturated except at small angles, this controller behaviour would resemble the one from the turn-and-go approach but will not be forced to stop to perform small corrections. In some sense, the inversely dependent linear velocity control can be seen as "the best of both worlds" since it attempts to optimise the velocities of the system, while being able to perform corrections while driving straight.This latter approach will be implemented as the position controller of the AGV.

4.3.3 Trajectory Defining Mechanism

The localisation of the AGV is done by the VICON system consecutively, providing the current coordinates or position of the robot. This system is not only accurate and precise but also has a commendable speed which has no lag with the real time operations, thus, providing excellent sync with the routing algorithm and the robot motion as can be seen in real time plot of the trajectory in figure 5.22. The position and orientation inputs to the trajectory planning algorithm assists in navigating the path from the current position to the desired destination. The axis of the object defined in VICON must coincide with the axis of the world to get correct angle calculations. The angle output from the trajectory planning algorithm, as is shown in figure 4.11a and 4.11b gives instructions to the controllers and help them calculating the speeds of the left and the right wheels.



(a) Robot Manoeuvring towards the desired goal.

(b) Logic of Trajectory Planning.

Figure 4.11: Depiction of angle calculation for defining robot trajectory

The logic used to make the robot follow a defined trajectory in the project is to consider the current position of the robot $(x_{\mathrm{R}}, y_{\mathrm{R}})$, that is received from the VICON system and the next goal position $(x_{\mathrm{T}}, y_{\mathrm{T}})$, that is received from the routing algorithm. Using these two coordinates, the calculation of error angle is done by using the formula 4.7 and 4.8:

$$b = \frac{y_{\mathrm{T}} - y_{\mathrm{R}}}{x_{\mathrm{T}} - x_{\mathrm{R}}} \tag{4.7}$$

$$\epsilon_{angle} = atan(b) \tag{4.8}$$

Upon checking the relative positions of the robot and the next goal, the following steps are followed in order to make the robot follow a defined trajectory :

1. Determine the angle with respect to the VICON orientation by comparing the positions of the robot and the target.
2. Determine the angular error by comparing the orientations of the robot and the VICON.
3. Then, on observing the value of the angular error, the calculation of angular and linear velocities can be done.
4. Finally, different values of Kp are tested in order to modulate the angular and linear velocities simultaneously.

Calculation of angle needed to be turned w.r.t.VICON coordinates (vtheta)

First, the value of the angle is needed that gives the measure of turn required to reach the target in VICON's coordinate system. This angle is being called **vtheta** here. It is calculated considering the separation distance between the robot position and the target position and then, based on that, the different aspects of the respective positions of the robot are used.

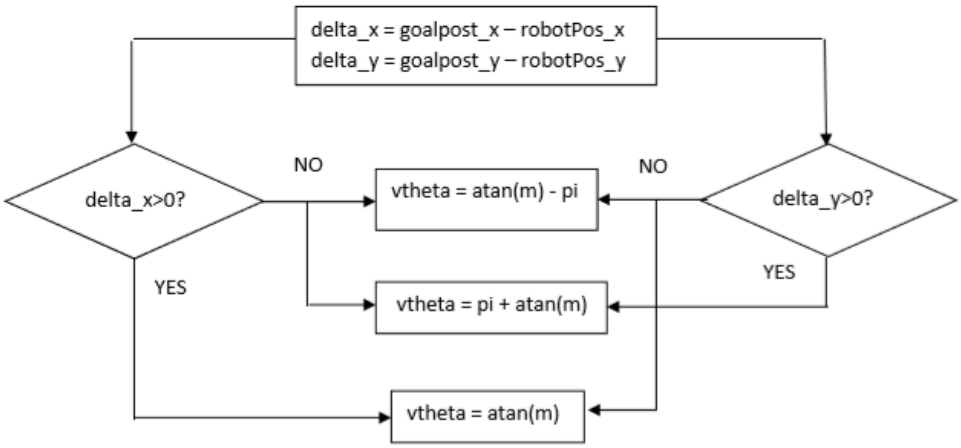


Figure 4.12: VICON angle calculation.

Calculation of angular error by which robot has to turn (ang_{err})

After the calculation of vtheta and ctheta (angular orientation of robot w.r.t. VICON), the next step is to calculate the angular error which will determine the gain, the angular velocity and the linear velocity of the robot. This parameter works as the error input for the PID controller.

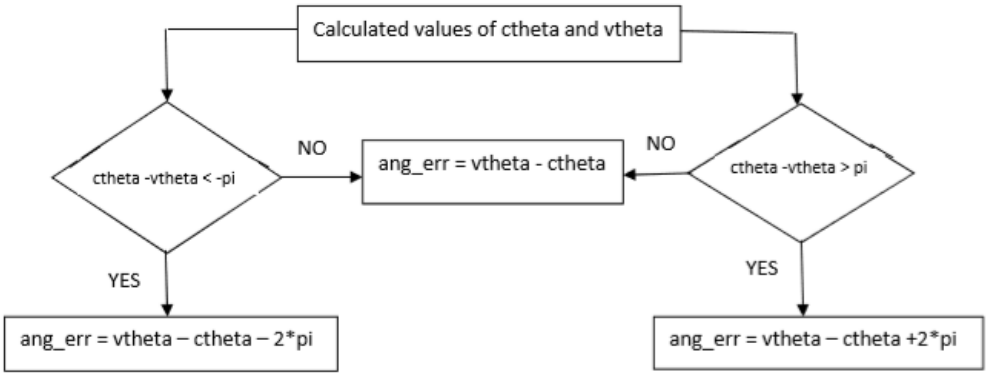


Figure 4.13: Angular Error Calculation

Below is the entire flow of the trajectory algorithm that has been deployed in this project, is depicted with the help of a flow diagram.

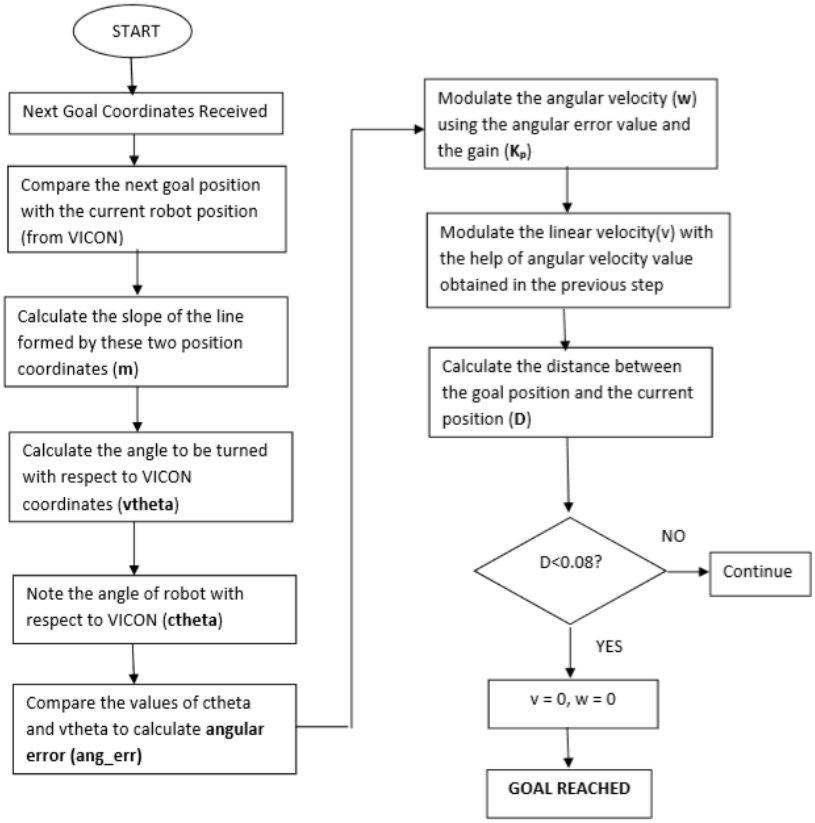


Figure 4.14: Flow of Trajectory control algorithm used

Control of DC motors

Accurate control of the DC-motors is an important aspect of achieving precision in system translations. The motors spins with an angular velocity that is proportional to the current flowing through the motor windings. From Ohm's law, if a constant electrical resistance is assumed, it can be seen that this also means that the rotational speed of the motor is proportional to the voltage, since voltage is proportional to the current.

The Arduino Mega does not support an analog output. It do, however, have the ability to provide PWM-outputs, which behaves as an analog output, if the electromechanical response time of the motor is slower than the frequency of the PWM signal. If the motor is in fact slower, energy is stored in the windings of the motor, effectively smoothing out the power bursts. [14]

The voltage experienced by the DC-motor is shown in figure 4.9, where V_b is the *voltage output* of the battery and DC is the *duty-cycle*, ranging from 0 - 100

$$V_{eff} = V_b * DC \tag{4.9}$$

Since the voltage output of the battery is dependent on the current charge of the battery, it is not possible to control in the current setup. Instead, the duty-cycle can easily be controlled with respect to achieving specific speeds of the motor. However, consistently controlling the motor velocity is made difficulty by the decreasing battery charge, since it changes the response of the motor over time. If the relationship between motor voltage and motor speed is in fact completely linear this would mean that the DC-gain of the response will also decrease linearly.

4.4 Integration

After having all the functionalities implemented, we put all of them together in order to send the positions of the robot, calculate the trajectory with the position information and to control the robot to make it follow the trajectory with the help of the VICON system. To build the communications between all the parts of the system, we have used Robot Operating System (ROS), where everything is calculated in different nodes and it communicates with the other nodes through the topics or parameters. In this section, we will explain what is done in each node and which topics are read or published in order to make the robot work.

4.4.1 Robot position module

To get the position of the robot, the Raspberry Pi is set up as the master and the computers as the slaves. With this configuration, the computer can read the VICON data and publish it, and the Raspberry Pi is able to see it. The angles published with the VICON are in quaternions and what is needed is the yaw, rotation in Z, in the Euler coordinates. In order to make this transformation, a node is created and

publishes the message type *robot*, containing x, y and yaw, into the topic *robotPos*. This is seen by figure 4.15.

Figure 4.15: Publishing the robot position in Euler coordinates diagram

4.4.2 Scheduling module

In this module, the path is calculated, taking into account the position of the robot, the warehouse and robot components, which comprises of two parameters. Each time the robot is at 7 cm from the last published *goalPos*, it publishes a new one. This message contains the position of the next target that the robot has to go to, the component that belongs to that position or "−1" if it is not a component position along with the action that has to be taken.
The following actions are implemented:

- Action param equal to 0: It means that it is a way point and the robot does not have to stop.
- Action param equal to 1: This will make the robot stop on reaching the assigned goal and take the corresponding component.
- Action param equal to 2: This will also make the robot stop on reaching the assigned goal, but this time it will leave the corresponding component.

When the robot takes all the components for a product, this node publishes a Boolean equal to True to the topic *productDone* to let the other nodes know that an assembly can start as soon the components reach to the warehouse.

In order to know whether a product is defected or not, this node is subscribed to the topic *Alarm* which if True means that the product assembled is defected, so it has to recalculate the path. In Figure 4.16, it is possible to see the connections that interact in the scheduling module.

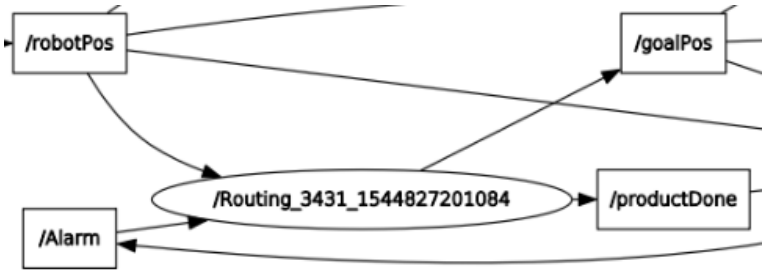


Figure 4.16: Schedule module diagram

4.4.3 Robot Components Manager module

This module has been implemented in order to keep track of the components that the robot is collecting and leaving every time it reaches to a component's pick & place area. By reading the *goalPos* topic, it is possible to know when the robot will pick or leave a component, and which it will be. In Figure 4.17, a diagram of this module is represented.

Figure 4.17: Robot Components Manager module diagram

4.4.4 Warehouse manager module

To know when to assemble the products and to give the feedback of the Quality Check (QC), this module has been implemented. When the robot reaches the warehouse and if it has components, this module takes all the components of the robot and puts them to the warehouse. If at this point, the warehouse receives a message through the topic *productDone*, it starts the assembly of the product. Once the product has been assembled, it gives the results of the QC through the topic *Alarm* which is read by the scheduling module. In Figure 4.18, the diagram of this module is shown.

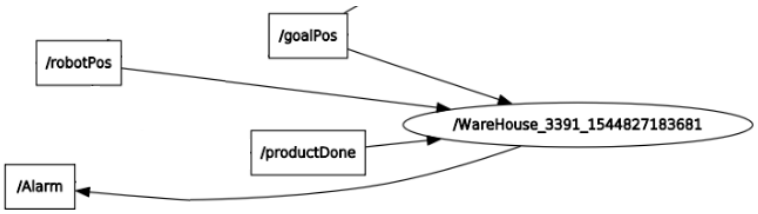


Figure 4.18: Warehouse manager diagram

4.4.5 Robot trajectory controller module

With the input from the topic *goalPos* and the position of the robot, the linear and angular velocities are calculated in this module in order to follow a path to get to the desired position. *goalPos* only sends the next position once, when the AGV is within a certain distance from the goal. The new goal position is then saved by the trajectory controller, and updated when the AGV passes the point.
The linear and angular positions are published to the topic */some/topic* which is read by the Arduino and makes the robot move.

In Figure 4.19 the diagram of this module is illustrated.

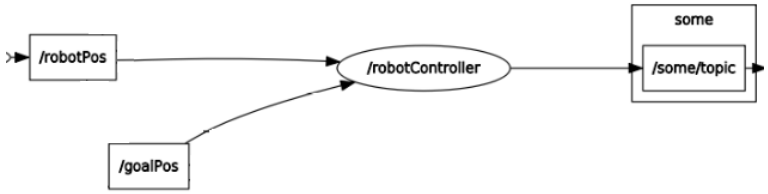


Figure 4.19: Robot trajectory controller module diagram

4.4.6 Robot wheels controller module

While receiving the desired linear and angular velocities, it performs an inverse kinematics to transform these velocities into an input for the DC motors and with a controller, keeping the desired velocity in the robot. The diagram of this module can be seen in Figure 4.20.

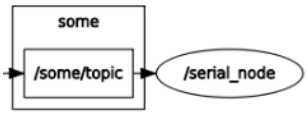


Figure 4.20: Robot wheels controller module diagram

4.4.7 Overall diagram

After having all the modules implemented and integrated, the overall diagram with all the connections are shown by figure 4.21. The result is the robot moving to the set points according to the scheduling module to follow the trajectory.

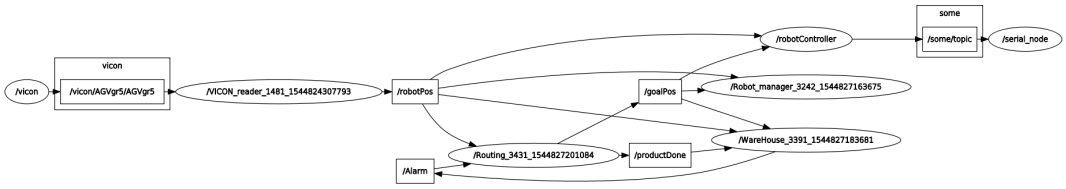


Figure 4.21: Overall diagram

Chapter 5

Testing

5.1 Scheduling tests

In the following tests, it will be shown that the path can be calculated when a list of products is sent to the algorithm. There will be a computation comparison between two implementations of the Dijkstra algorithm and it will also be checked the difference between the two routing algorithms implemented in terms of time and distance.

5.1.1 Path calculation

In order to show that the path is calculated in a proper way, a list with 3 products, which are product 1, product 3 and product 2, is sent to the greedy algorithm and by modifying the position of the robot different results are obtained.
In figure 5.1, it is illustrated the path that the robot will follow if it is placed in sector 1. It is possible to see that the path will make the robot go through the central middle point (nID:2). However, If we put the robot close to one of the lateral middle points it will go through them, but if not, as the distance to the components is bigger than going to through the central middle point.

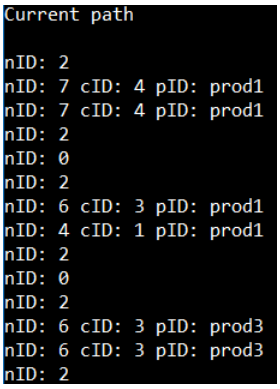


Figure 5.1: Path calculated with the greedy algorithm when the robot is in sector 1

In figure 5.2, it is possible to see that if we put the robot in sector 2, the path will be different, because the robot was placed closer to component 3 and that's why it takes component 3 (cID:3) rather than component 4 (cID:4).

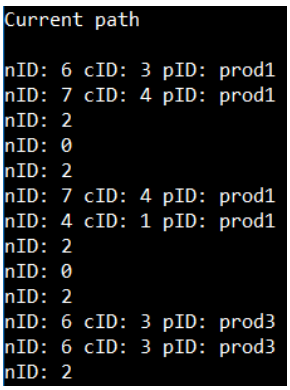


Figure 5.2: Path calculated with the greedy algorithm when the robot is in sector 2

The following test is with the algorithm that searches for the optimal path in a window size of two products using the same combination of products as in the previous tests. The test in figure 5.3, is when the component was in sector 2 and it can be seen that the path is different than the path generated in figure 5.2. Where, instead of collecting the closest components (cID:4 + cID:3), it takes a component (cID:1) further away and then for the next path combine two components in the same position (2 x cID:4).

```
Current path

nID: 6 cID: 3 pID: prod1
nID: 4 cID: 1 pID: prod1
nID: 2
nID: 0
nID: 2
nID: 7 cID: 4 pID: prod1
nID: 7 cID: 4 pID: prod1
nID: 2
nID: 0
nID: 2
nID: 6 cID: 3 pID: prod3
nID: 6 cID: 3 pID: prod3
nID: 2
```

Figure 5.3: Path using optimal path algorithm with a window size of 2 with the robot at sector 2

After seeing the previous result, it has been decided to plot the distances and compare them with the once obtained from the calculated path using the greedy algorithm. In figure 5.4, it is possible to see that the optimal path algorithm sacrifices the first journey, in order to improve the distance in the second journey and be able to collect all the needed components with a shortest distance.

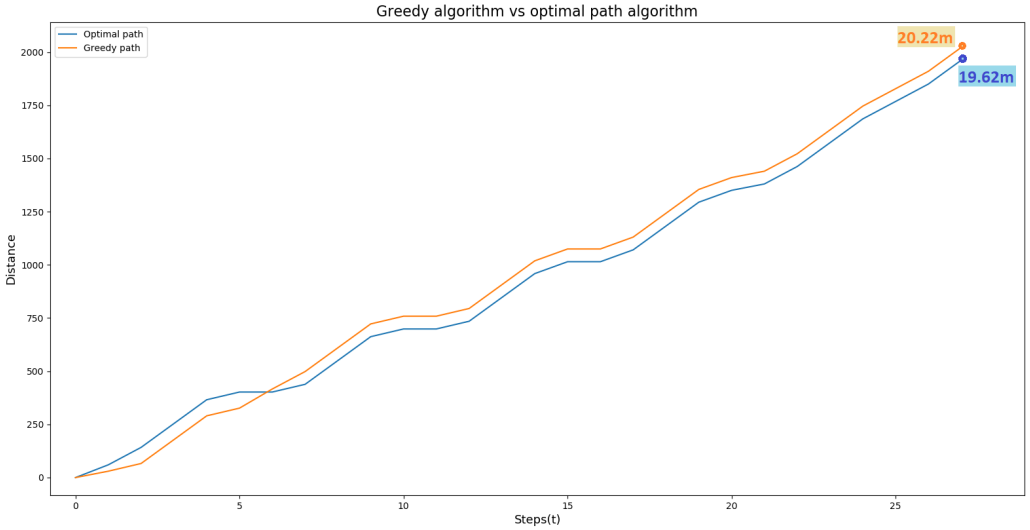


Figure 5.4: Distances along the path of the two scheduling algorithms implemented when collecting 3 products.

As illustrated in the plot, the optimal path can collect all the needed components with a distance of 19.62m and the greedy algorithm with a distance of 20.22m. In this plot the difference is not huge, but taking into account that the robot might collect a large number of products, these differences may become higher differences. This is why, a test with 30 random products has been made and see if the difference in distance changes along the path. The test in figure 5.5, shows the result of this, taking that the starting point of the robot is in sector 1 and without components.

It is possible to see that the distances between both algorithms gets bigger and bigger until reaching into a difference of 5m between both paths after collecting all the needed component to assemble 30 products.

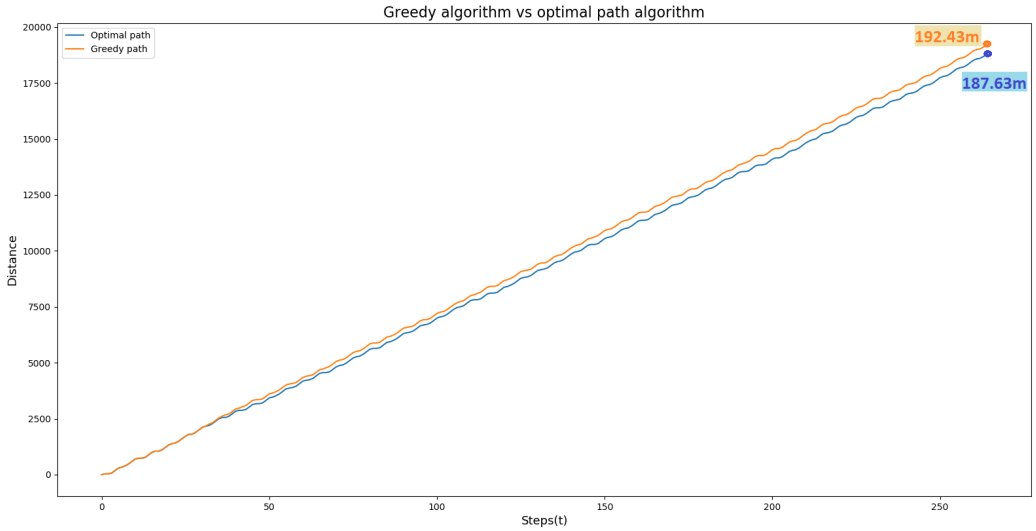


Figure 5.5: Distances along the path of the two scheduling algorithms implemented when collecting 30 products.

5.1.2 Multiple components into the warehouse

In this test, it is compared how both algorithms treat if there is a high probability to have more than 3 components of one type in the warehouse. TO make this situation possible, it has been placed 2 components of the same type, cID = 4, into the warehouse and the robot position has been set close to component 4 (cID:4) in order to make the distance to that component the shortest.
It is possible to see in Figure 5.6 that both algorithms treat this situation in a different way.

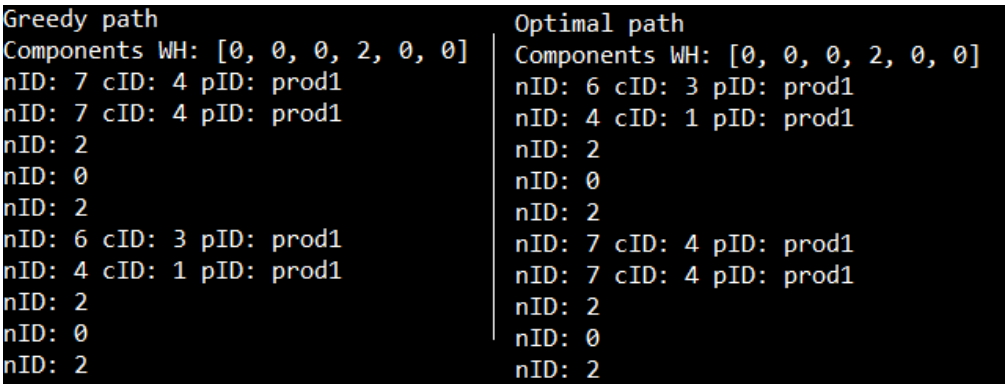


Figure 5.6: Comparison of both scheduling algorithms when there are 2 components of the same type in the warehouse.

In one hand, the greedy algorithm calculates the path for all the products to be assembled at the beginning, so it does not know which components will have the warehouse in a future. So in this situation, the greedy algorithm will bring 2 more components 4 (cID:4) into the warehouse, producing a situation that is not possible, to have more than 3 components of a type in the warehouse.

In the other hand, the optimal path algorithm calculates the path for each product taking into account the components that are in the warehouse, to avoid having more than 3 components of a type. In order to avoid that situation, what the optimal path algorithm does, by knowing that the robot takes more than 10 second to do one round and that the warehouse takes 15 seconds to assembly one product, is to leave the component that will overcome the warehouse capacity to the next round. However, as this algorithm searches for the best path, in Figure 5.6 it is possible to see that will take the 2 components 4 (cID:4) to the next round which is the best option to solve this situation

5.2 Quality Check

In the following tests, we have proved that our scheduling algorithm can recalculate the path in every situation that the robot can possibly be in. These situations depend on the number of components that the robot can have at the moment the QC goes negative, and also on the position of the robot. In these plots, it lists the path that the robot has to follow along with the relevant information of the class node at each test. All the tests are made with no components in the warehouse (WH).
In the following test, an optimal way to calculate the path is not found. In order to find it, more tests should have been done, but with lack of time it was not possible.

5.2.1 The robot has no components

This test was done when the robot had no components and was in sector 1 or in sector 2, as explained in figure 4.7 which means that it was going to the workstations to collect components. To show how the path changed, figure 5.7 shows the path that the robot was following before the result of the QC and was in sector 1, named as current path to the left. The current product being assembled and QC'ed at the WH is product 4 (pID: prod4), which proves as defect on the right image.

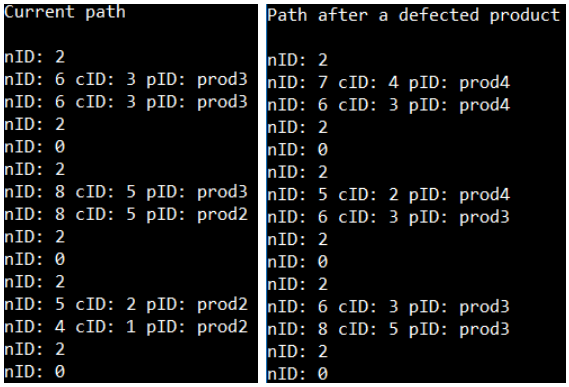


Figure 5.7: Test 1 results: No component in the AGVs inventory - located in sector 1.

After the path was recalculated, figure 5.7 shows that instead of picking the components for product 3, the robot changed its path and the components of the defected product (pID: prod4) were taken. If the robot is in sector 2, the results are the same as showed in figure 5.8.

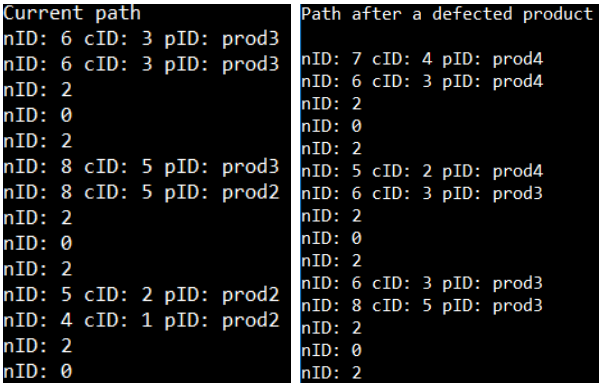


Figure 5.8: Test 2 results: No component in the AGVs inventory - located in sector 2.

5.2.2 Robot has components and is in sector 1

In this case, the robot already had two components in the inventory and therefore the next goal node was the WH, in order to leave the components, as can be seen

in figure 5.9 under the current path.
After recalculating the path due to the negative QC, it is seen that two components are included in the path from the defective product. This is because the 3rd component (cID: 3), which is the one missing in the path, is in the AGVs inventory. It can also be seen for the product in progress (pID: prod3), as one component is in the AGVs inventory, thus only components 3 & 5 (cID: 3 + cID: 5) are added to the path.

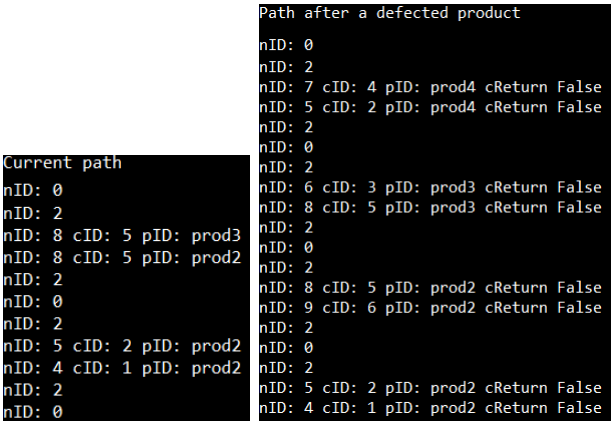


Figure 5.9: Test 3 results: The robot has 2 components (1 needed) and is in sector 1

5.2.3 The robot has 2 components needed and it is in sector 2

In this situation, the robot had two components: 3 and 4, to assemble the product 1 at the time the QC was published negative and it was in sector 2. In figure 5.10, it can be seen that the component (cID: 2) needed to assemble the defective product (pID: prod4) was included in the path. However first the robot needs to deposit its inventory at the WH, as they are also a part of product 4.

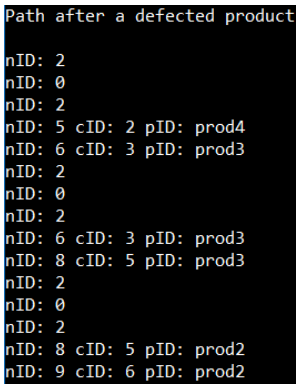


Figure 5.10: Test 4 results: The robot has 2 components needed in sector 2.

5.2.4 Robot has 1 component and is needed

In this test, the robot picked one component (cID: 3) to assemble the product 3 at the time the QC went negative. As this component was needed to assemble the defect product (pID: prod4), it is prioritised for re-assembly, the closest needed component (cID: 4) was added to the path in order to take it to the WH as seen in figure 5.11.

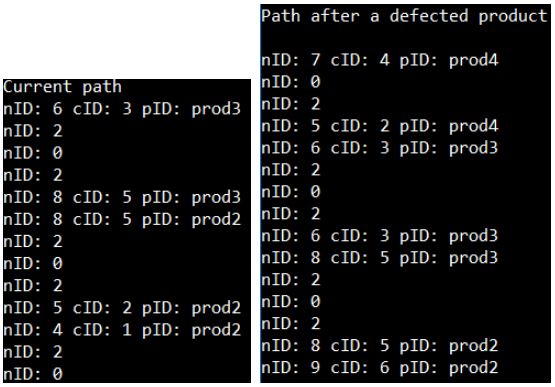


Figure 5.11: Test 5 results: The robot has 1 component, which is needed.

5.2.5 Robot has two components and only one needed in sector 2

In this test, the robot had two components (cID:5 & cID:3) and only one was needed (cID:3) to assemble the defected product (pID:prod4). As the goal was to assemble the first product of the list as soon as possible, the path recalculated left the component not needed (cID:5) and took the closest needed component (cID:4) as shown in figure 5.12.

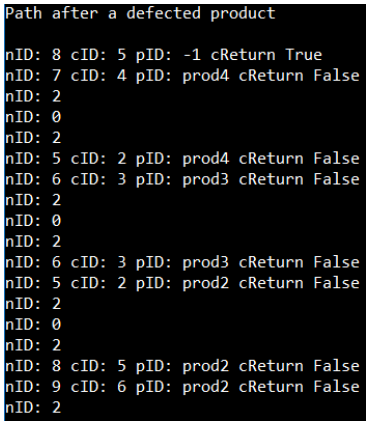


Figure 5.12: Test 6 results: The robot has 2 components (1 needed) and is in sector 2

5.2.6 Robot has 1 component and is not needed

In this test, the robot collected a component that was not needed for the defected product (pID: prod4). However, it did not return it back as it might be used in the future for the product (pID: prod3) that was previously in progress and it collected the needed component (cID:4) for the defected product. In figure 5.13, it is shown that it goes for the closest component (cID:4) from the current position of the robot and then, it assembled the defected product (pID: prod4) completely and thereafter assemble product 3, as it previously had one component in the inventory and the last two (cID:3 & cID:3) added to the path.

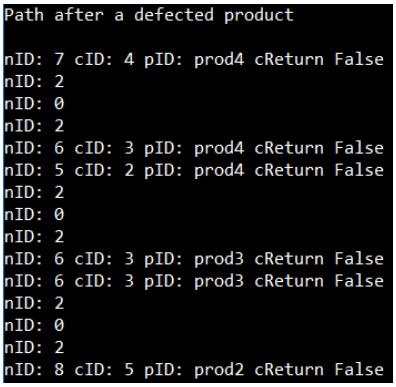


Figure 5.13: Test 7 results: The robot has 1 components that is not needed

5.2.7 Robot has two not needed components and it is in sector 2

In this test, the robot collected two component (cID5 & cID5) to the WH, when QC became negative. The defected product (pID: prod4) did not need these components, so in order to bring needed components as soon as possible, it planned to drop off one component (cID:5) and pick the closest component (cID:4) as seen in figure5.14. The component 5, that is brought to the WH, is used to assemble the product 3.

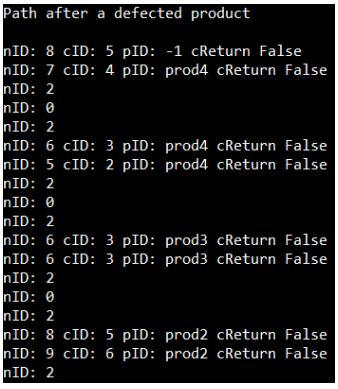
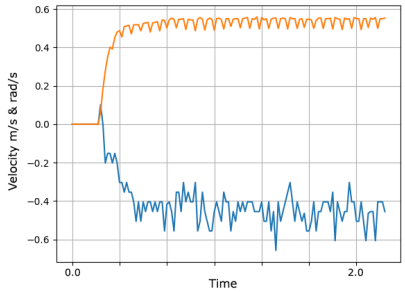


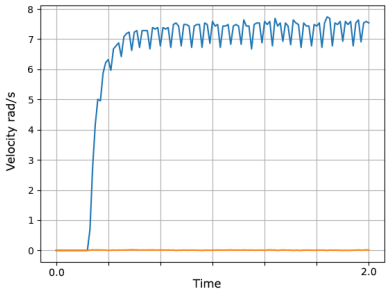
Figure 5.14: Test 8 results: The robot has 2 components that are not needed - in sector 2.

5.3 Baseline test

A preliminary test was performed with the robot to establish how the robot behaves without controllers. The test was carried out by applying a step control signal to the system and collecting data for 2 seconds from different sensors, providing the perfect opportunity to compare performance of multiple sensors simultaneously. This test was used to approximate the dynamic behaviour of the robot that can be used for trajectory planning and control. The data was used to determine the maximum linear and angular velocity of the robot as well as the acceleration profile. For testing the maximum linear velocity, 100% duty cycle was applied to both motors moving them in the same direction. For testing maximum angular rate, the two wheels were rotated in opposite directions.



(a) Response to step control signal - linear velocity test



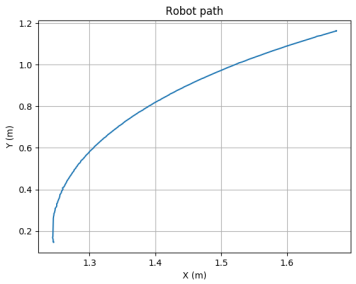
(b) Response to step control signal - angular velocity test

Figure 5.15: Linear(orange) and angular(blue) velocity response to step control signal (encoder data - 100Hz)

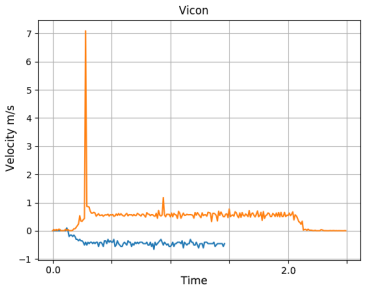
Figure 5.15 shows the response of the system to different linear and angular step control signals. Without the implementation of feedback control in the system, linear step signal yields both translation and rotation as the two wheel velocities are different given the same control signal. This could be the result of different amount of wear and tear, manufacturing tolerances etc. It can be concluded that a controller is required to achieve desired output values based on feedback from the wheels.

The maximum linear velocity and an approximation of the acceleration can still be obtained by averaging multiple runs. The curve can be approximated with two linear segments. A conservative approximation of the maximum velocity of 0.5m/s is assumed given the data. Acceleration between 0 and the maximum velocity can also be linearly approximated resulting in value of 3.3m/s^2. In the case of the angular velocity, the similar linear approximation is sufficient yielding $\omega_{\max} = 7\text{rad}/s$ and it is reached with the acceleration of $\dot{\omega} = 46\text{rad/s}^2$.

Although the VICON system is primarily used to determine the position and orientation of the robot, it can also be used to calculate the velocity of the robot by differentiating its position and orientation. Figure 5.16b shows the linear and angular velocity of the robot calculated from the robot position. Despite the excellent accuracy of the VICON system when it comes to positioning, encoders provide more accurate velocity information for the two wheels individually making it a primary candidate for velocity control feedback.



(a) Robot path produced by the velocity data shown in Figure 5.15a. Starting position $x = 1.24, y = 0.17$



(b) Linear(orange) and angular(blue) velocity values calculated from robot position over time

Figure 5.16: VICON system data collected from test run. (position/orientation data - 100Hz)

5.3.1 Motor Control Tuning

Due to the results in section 5.3, it was concluded that a controller for wheel velocities was needed. PID control was chosen, since it is easy to implement and often provides reliable results. Different variants of the PID controller was tested and compared.

The parameters of the different controllers presented in this section were derived through a trial-and-error process. All velocity calculations was made at a frequency of 50 Hz and all tests was performed at free-spin, i.e. no load on the motor. The input is a step, with a reference value of 15 rad/s which is applied at 1 second.

P-control

Figure 5.17 shows the response of a P-controller. The response settles at the reference value, with a settling time of 0.16 s. The response has an almost unnoticeable overshoot of 5.4 %.

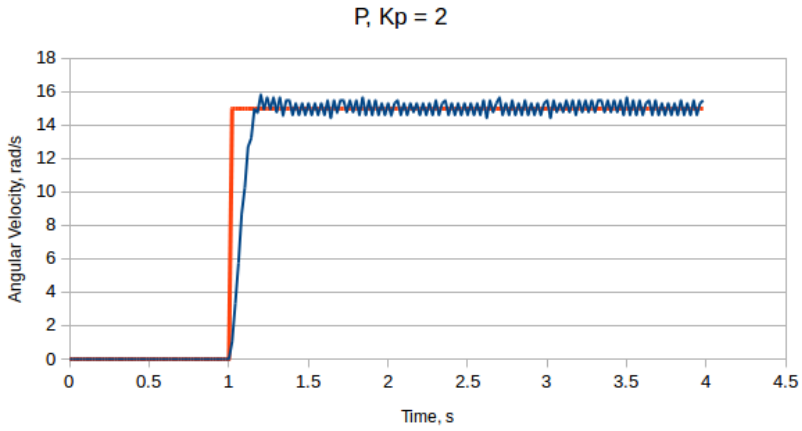


Figure 5.17: The response of a P-controller with $K_P = 2$

PI-control

Figure 5.18 shows the response of a PI-controller. The result shows a steady state error of approx. 0.5 rad/s. The overshoot is 1.3 %, however, the fact that the final value is lower than the target value conceals the actual overshoot. If the overshoot is calculated based on the final value, it is 4.6 % which is still acceptable since it is within 5 % of the target value. The rise time of the system is 0.08 s.

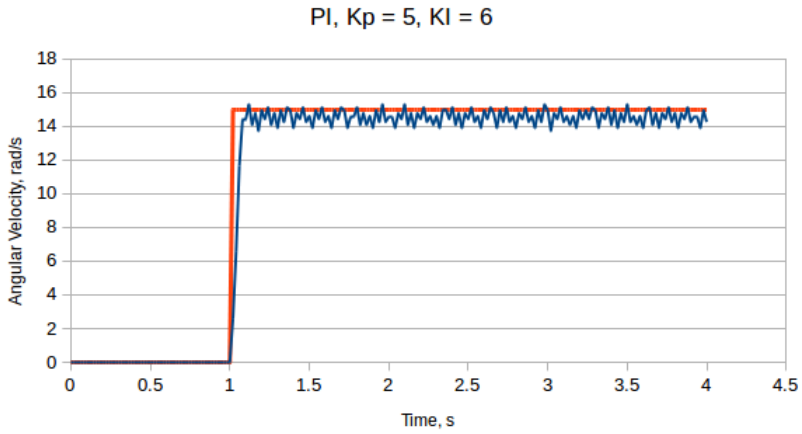


Figure 5.18: The response of a PI-controller with $K_P = 5$ and $K_I = 6$

PID-control

Figure 5.19 shows the response of a PID-controller. It can be seen that the response settles around the reference value (15 rad/s). The settling time of the system is 0.1

seconds and the overshoot is 4.6 % . Since there is practically no overshoot and the settling time is low, the system seems to be critically damped.

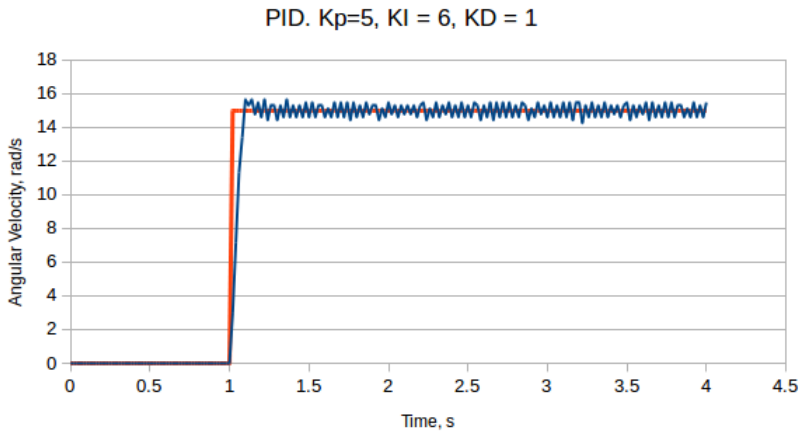


Figure 5.19: The response of a PID-controller with $K_P = 5$, $K_I = 6$ and $K_D = 1$

All 3 controllers perform well, having low settling times and overshoots. It was shown that the P-controller had both the largest overshoot and longest settling time. The PI-controller had a constant steady state error, so in conclusion the best controller for the motors is the PID-controller with $K_P = 5$, $K_I = 6$ and $K_D = 1$.

5.4 Accuracy test

One of the performance requirements of the system was that the position error of the system should be less than 5 cm or $\frac{20cm - crosssection}{2}$, such that the system was able to pass through the gap in the middle of the wall between the manufacturing and storage areas.

This test will be a proof of concept of some of the claims made in 4.3. 4.3.1 states that given a static world map, moving in a straight line between two fixed points would ensure a collision free path. 4.3.2 claimed that a go-to-goal controller, with the linear velocity inversely independent to the angular velocity could imitate a turn-and-go controller, capable of adjusting it's heading without stopping.

In the test, the robot was operating in the warehouse scenario, delivering components to the assembly station as specified by the scheduling. The position of the system was recorded, in order to confirm that it was capable of going through the middle gap in the wall without collision. The coordinate of the center of the gap in the vicon world coordinates is x:1.25, y:1.64. The cross-section of the robot is 10.769 cm. This means, that in order to avoid collision, the x-position of the robot

at y:1.64 have to be in the range of $1.25 \pm \frac{0.2-0.10769}{2}$.

x-values were recorded whenever the y-value was in the range 150 - 170. The PID-values found in 5.3.1 was used for the motor controllers. The go-to-goal controller is a pure P-controller with $K_P = 4.3$. The results are shown in 5.20.

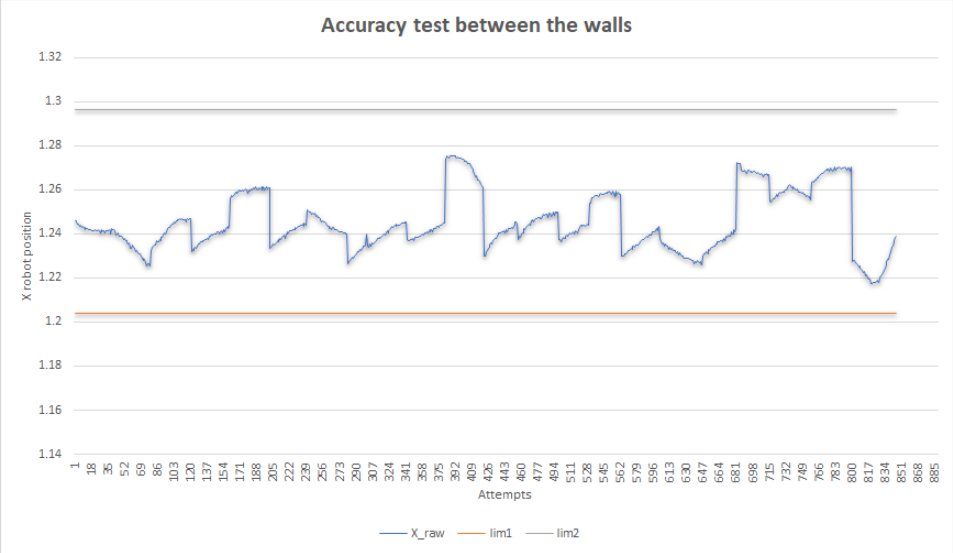


Figure 5.20: Plot of the x-value of the system when passing through the gap in the middle lines and the upper and lower boundaries.

It can be seen that all the x-values lies between the boundaries, meaning that it passed through the gap each time without collision. All component positions was visited during the test. In conclusion, it can be said that the go-to-goal controller achieves the movement accuracy required by the system.

5.5 System Evaluation

In this test the systems was tested in the warehouse scenario, in order to confirm that the system is able to reliably deliver components to the assembly station.

To confirm that there is an improvement when implementing the optimal path algorithm over the greedy algorithm it has been decided to compare both algorithms with the time that it takes to the robot to collect all the components. In order to complete this test, it has been used the parameters that gave better results in the previous control test.

In Figure 5.21 it is possible so to see, that after collecting all the products the optimal path algorithm takes 8 seconds less than the path calculated with the

greedy algorithm. It took in total $12,75min$ to gather 28 components, averaging $27.3sec/prod$.

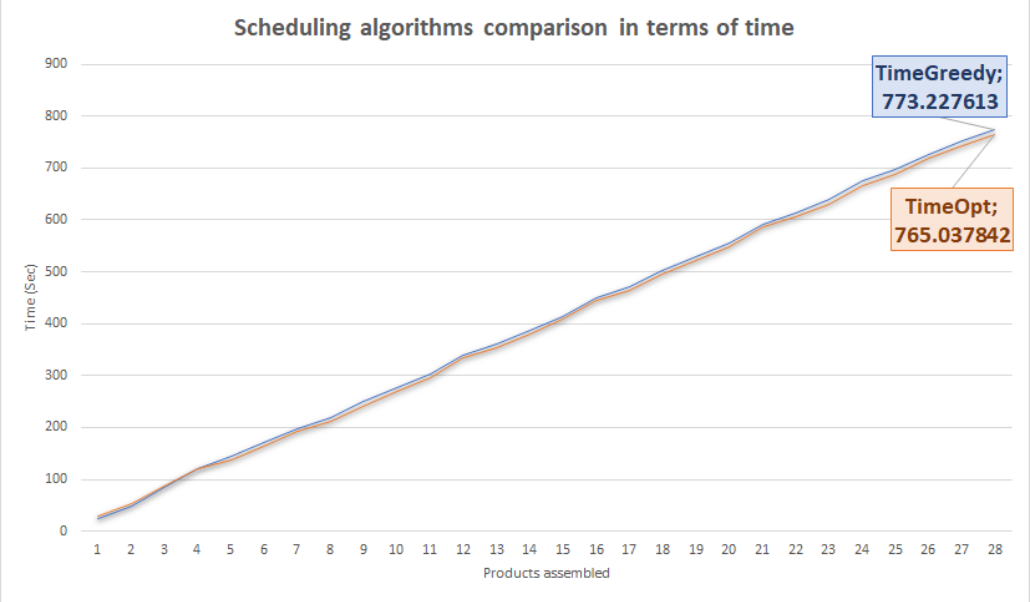


Figure 5.21: Time of greedy and optimal path algorithm when collecting 28 products.

The real-time path plotting of the robot trajectory is shown in Figure 5.22. This plot has been obtained while the final test was being executed and it is possible to see all the 6 nodes corresponding to the components and the node representing the central middle point where the robot is instructed to go to. The scheduling algorithm also takes into account the lateral middle points in order to find the optimal path. However, these two positions will be only used if the robot is placed, at the beginning of the execution, close to them. If not, these two points will never be used as seen in Figure 5.22. It follows the instructions given by the routing algorithm, giving a precisely controlled motion.

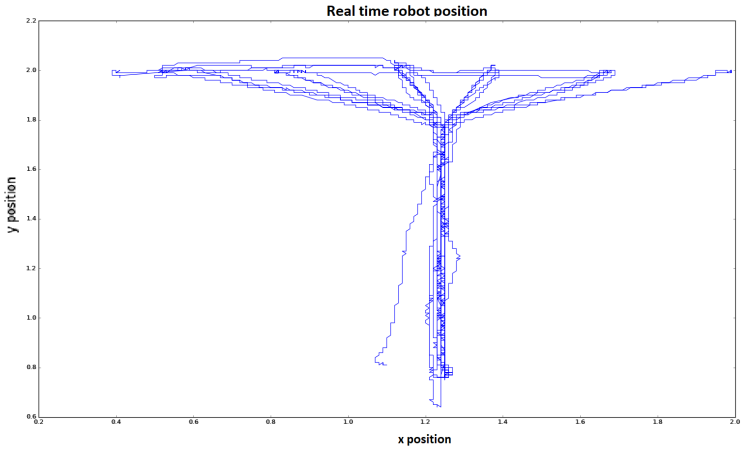
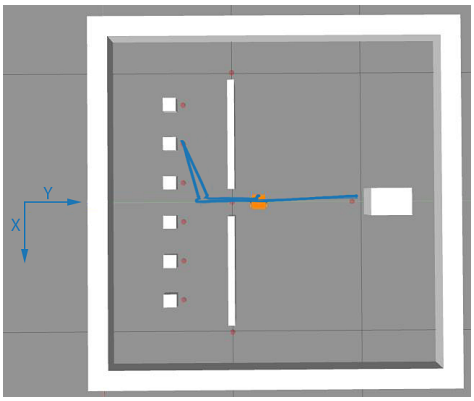


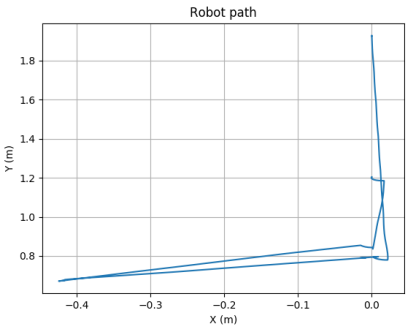
Figure 5.22: Real-Time Plot of Robot Trajectory

5.6 Simulation test

Complementary to the development and testing of the physical robot, a simulation was also created as an extension to the testing tool-set. Preparatory to real life testing of the system, a simulation environment approximating the real system kinematics, dynamics and behaviour was set up. This allowed for safe testing of the motion planning algorithms, with high repeat-ability. The simulation software used for this purpose was Gazebo that offers integration with ROS. This way all nodes previously written could be attached to the simulated environment to control the robot. A 1:1 scale of the production facility was created alongside the approximated model of the robot.



(a) Rendered environment with overlay of path 5.23b

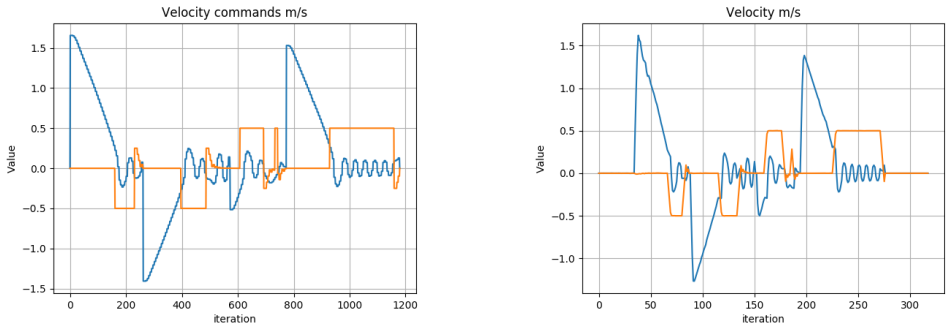


(b) Robot path in the simulation test. Note: the scale of the two axis is non-identical.

Figure 5.23: Images created using the Gazebo simulation tool.

For the sake of simplicity, a test with a single pick up action is presented in the

report. The initial position of the robot was between the obstacle and the drop of point. First the robot crossed between the obstacles, then proceeded to one of the pick up position, then returned to the mid-point between the obstacles, and finally delivered the components to the workstation. The robot was governed by a turn-and-go behaviour.



(a) Velocity commands sent to the robot at 100Hz running in Gazebo simulation.

(b) Velocity data of the robot in the simulated environment recorded at 20Hz.

Figure 5.24: Velocity commands controlling the robot in the simulated environment. (blue - angular velocity, orange - linear velocity)

Figure 5.23b shows, that although the robot completed its trajectory collision free, it is not perfectly following the optimal trajectory and deviates by a couple of centimetres at times. This performance could be improved by increasing the rate of update frequency of the feedback signal, providing the controller with a higher resolution signal or by improving the model of system dynamics. However it should be noted, that due to the relative simplicity of the developed simulated robot model, a finely tuned controller on this system will probably not perform well as part of the real system. As a way to improve on this aspect, disturbances on the system could be applied that would better resemble a real case scenario.

Figure 5.24a shows the output signals of the trajectory generation sent to the robot with a frequency of 100Hz. Figure 5.24b shows the actual velocity of the robot updated at a frequency of 20Hz. Both positive and negative velocity commands are sent to move the robot forwards and backwards, minimising the amount of rotation it has to perform, thus improving overall speed. The behaviour of decoupling translation and rotation is also shown in the figures.

Chapter 6

Discussion

The project proves the capabilities of using an autonomous systems to optimise and perform tasks within a manufacturing environment, transporting components from workstations to an assembly station. However there are limitations to the model, which are to be evaluated in order to decide what it would require to implement this in a real world scenario. This eventually prove as a purpose for the scalability to a dynamic environment, whereas the AGV is not isolated, and there would be interaction in the field of operation.

6.0.1 Capability and limitations

The project achieves the use of a high level commanded autonomous operation, in which the AGV replace labor in an activity of transporting material goods. This have real world cost saving potential as 30-75% of the manufacturing cost is due to material handling. It proves its capability to which the manufacturing process could be automated using AGV(s), possibly improve time, cost and quality.

In terms of this project, it's worth to mention that this is achieved with constraints and that a real world application would require additional functionalities. It can be broken down into a set of limitations, that would need to be solved in order to scale the application to a real world scenario.

Simulation - Before operating the system in a real world scenario, simulating the operation prove as an efficient way to ensure the system is running smoothly and in case of errors, these are handled, so that when finally deployed in the real world, fatal errors do not occur.

Static environment – The simulation of the warehouse take place in a closed environment, where there are no dynamic obstacles such as other robots, inventory or humans.

Fixed components – All component has a fixed dimension, which allows the robot to always carry two components no matter the selection.

Pick and place – would prove as an advanced process for the AGV/workstation, especially given the various dimensions the components could take, and would require additional actuators and sensors.

Constant supply – It is taken as given that components are available at all times both in terms of time and quantity, which differs from the real world.

Continuous usage – Scaling the AGV to a real world application size would require a higher battery capacity. Furthermore if continuous usage is required a charging time would result in a production stop.

Given these limitations the use of AGVs in real world application is still highly viable and the necessary functionalities to implement such will be provided in the next section.

6.0.2 Scalability

To establish a system of AGV(s) in a dynamic environment, various aspects are to be included in the consideration for the solution. This comes down to three topics of discussions: (1) navigation (2) pick and place, and (3) the system.

For the AGV to navigate in a dynamic environment it would require knowledge of not only the position of the AGV and its goals, but also obstacles in its trajectory. Furthermore VICON would not be a viable solution as the field of operation would probably be too big for tracking with stationary cameras. Alternatives to the VICON system would be (a) an optical guideline, (b) magnetic anchoring points or (c) laser navigations, whereas the latter would provide the highest degree of flexibility. This would provide the AGV with the functionalities to track its way to the goal using laser and marking points on e.g. walls. However still lacking the tracking of obstacles in its trajectory, it should be equipped with a real-time, range camera with two cones spanding different lengths in front of the AGV. The far sighted cone would reduce the speed of the AGV depending on the distance to the obstacle. The close sighted cone would bring the AGV to a complete stop to avoid collision with the obstacle.

The functionality of pick and place requires additional operations of either the AGV or the work/assembly stations to (un)load the AGV. Depending on the autonomous level of the operations previous to the transportation from the workstations – and after, the possible solution could be either a robot arm or a conveyor belt loading the AGV. This has to further take into account the complexity of the components dimensions. Furthermore the AGV has a limited power supply due to

its mobility, and therefore it would be most viable to dislocate the pick and place to the stations instead of the AGV.

In a real world application, multiple AGVs can be used to ensure the AGV is not the bottleneck of the production and that there is enough capacity, when an AGV is charging. However this raises two challenges for the operation; (1) intersection between the AGVs are to be collision free and non-significant decreasing in terms of performance, while (2) avoiding deadlocks.

Ensuring that the AGVs do not collide with one another requires the AGV to have a braking distance depending on the camera used for navigation - as explained earlier. This is pictured below (figure 6.1), whereas if the AGV are to move in reverse, sensors here would also be required.

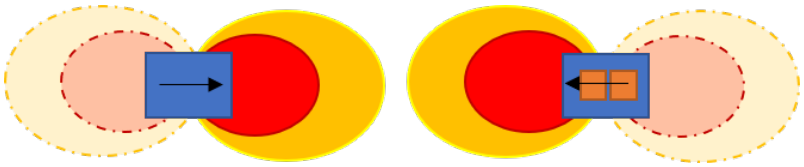


Figure 6.1: Two AGVs using cameras to detect possible collision. One distance would slow down the AGV, while the close distance would bring the AGV to a stop.

Alternatively the AGVs can move within zones, whereas the manufacturing facility is divided into zones with the size depending on the brake time of the AGVs. However the sensors would still be required in a dynamic setting. The deadlocks that can occur in terms of a manufacturing environment are as seen by figure 6.2, where (a) two AGVs approach a crossing and both request access to it, but are blocked by each other's presence. Secondly (b) where an AGV want to deposit components, but as inventory is full, it waits, however the second AGV is queuing to allocate said components, thereby creating a deadlock. Thirdly (c) where several AGVs block each other, each trying to access a "zone" that is occupied by another.

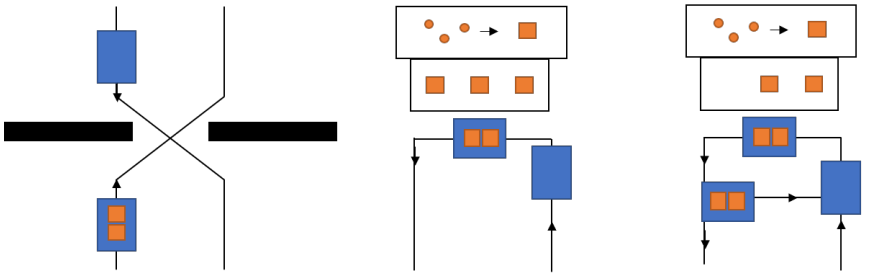


Figure 6.2: Deadlocks happening as (a) two AGVs try to cross the wall at same time, (b) AGV want to deposit components, but inventory is full, (c) AGVs are awaiting one another forming a circle lock. Note that lines are to simplify the understanding and are not fixed guidelines. [15]

Avoiding deadlock would require additions to the algorithm that either predict and prevent deadlocks or solve them in case of emerging. For scenario (a) a possible solution would be to add queuing zones at narrow spaces or place priority on the AGVs depending on inventory. For (b) adding a spacious environment around the workstations allowing for multiple AGVs to both pick and place during the same time. Lastly (c) would be the avoidance of cyclic deadlock require the AGVs to look one zone ahead of the next, thereby ensuring that there's available space for the AGVs to move without becoming locked. Addressing these topics would be beneficial to scale the project to a real life scenario and help achieve the task of both the project and more advanced systems using AGVs. To what degree of flexibility depends on the autonomous level of operations and therefore the above topics are only for the stated scenarios.

Chapter 7

Conclusion

In this project an AGV has been developed, capable of delivering components from a storage area to an assembly station in a static warehouse scenario. The system consists of a scheduling functionality that batches components together and calculates the shortest path to gather them and a navigation system that follows said path accordingly. The system is able to gather all components for a product at an average speed of $27.3 sec/prod$, without colliding with obstacles in the warehouse.

In the schedule section two algorithms were developed and compared in order to optimise the batching. It has been demonstrated the advantages of using the *optimal path algorithm* and thereby reduce the distance travelled by $4.8m$, when the batch size is of 30 products. This gentle reduction is proportional to the amount of product and thereby proves as a significant result in the long haul.
After testing both algorithms in the real world scenario, it has been found that the $4.8m$ become into a difference of $8sec$ between the algorithms. This means that in the long term, if more than 30 products have to be assembled, the difference on time may be bigger and as the goal is to minimise the bottleneck time it can be concluded that the best algorithm implemented is the one that calculates the optimal path using a window of two products.

It was determined that in order to properly control the velocity of the system, a PID-controller had to be implemented for the motors. The tuned PID-controller was shown to be critically damped with a settling time of $0.1sec$. Furthermore, it was shown that a go-to-goal control approach with an inverse proportional linear velocity, was able to accurately drive the AGV through the course without any collisions.

The physical design were met as the cross-section of the AGV was smaller than $20cm$, the 3D VICON markers separation distance was larger than $10cm$.

Bibliography

[1] S. Butdee, "Localization Based on Matching Location of AGV", *Proceeding of the 24th International Manufacturing Conference, IMC24, Waterford Institute of Technology, Ireland*, 2007.

[2] A. S. S. Butdee, "Learning and recognition algorithm of intelligent AGV system", *GCMM*, 2006.

[3] Y. C. Y. E. Sung Ng. K. Loon, "Parallel Linkage Steering for an Automated Guided Vehicle", *IEEE Control Systems Magazine*, 1989.

[4] F. D. D. Y. Lee, "Integrated Scheduling of Flexible Manufacturing System Employing Automated Guided Vehicles", *IEEE Transactions on Industrial Electronics*, vol. 41, 1994.

[5] L.-C. F. P-S. Liu, "Planning and Scheduling in a Flexible Manufacturing System Using a Dynamic Routing Method for Automated Guided Vehicles", *IEEE*, 1989.

[6] M.-C. Z. Naiqi Wu, "AGV Routing for Conflict Planning and Scheduling in a Flexible Manufacturing System Using at Resolution in AGV Systems", *IEEE International Conference on Robotics Automation*, 2003.

[7] ——, "Modeling and Deadlock Control of Automated Guided Vehicle Systems", *IEEE/ASME Transactions on Mechatronics*, vol. 9, 2004.

[8] A. S. M. Ertugrul O. Kaynak, "A Comparison of Various VSS Techniques on the Control of Automated Guided Vehicles", *IEEE*,

[9] Y. B. L. Beji, "Motion Generation and Adaptive Control Method of Automated Guided Vehicles in Road Following", *IEEE Transactions on Intelligent Transportation Systems*, vol. 6, 2005.

[10] C. X. Q. Fang, "A Study on Intelligent Path Following and Control for Vision-based Automated Guided Vehicle", *Proceedings of the 5th World Congress on Intelligent Control and Automation*, 2004.

[11] S. K. Mathur, "Microprocessors and Microcontrollers", *PHI*, 2016.

[12] R. B.P. V. P. Merriaux Y. Dupuis and X. Savatier, "A Study of Vicon System Positioning Performance", *MDPI*, 2017.

[13] A. A. H. Rached Dhaouadi, "Dynamic Modelling of Differential-Drive Mobile Robots using Lagrange and Newton-Euler Methodologies: A Unified Framework", *Advances in Robotics Automation*, vol. 2, 2013.

[14] C. K. Pratap Vikhe Neelam Punjabi, "Real Time DC Motor Speed Control using PID Controller in LabVIEW", *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, 2014.

[15] N. W.-C.T.C.-P. R.L. Moorthy W. Hock-Guan, "Cyclic deadlock prediction and avoidance for zone-controlled AGV system", *ELSEVIER*, vol. 83, 2003.

Appendices

Appendix A

IDEF0

IDEF0 is a function modelling tool used for analyzing and developing complex processes in a simple and structured way. The purpose is to breakdown a system into functions for which a given input leads to a desired output while meeting requirements through utilizing the control. It is graphically displayed by a sequence of boxes that represent functions that makes up the system. To these boxes are arrows indicating input, output, control and mechanism as shown in figure A.1:

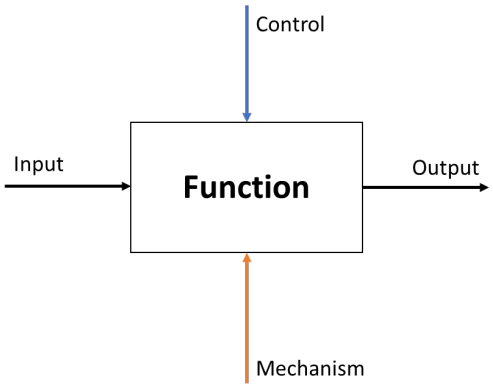


Figure A.1: A representation of a sequenced box in an IDEF0 diagram.

These arrows are either objects or data that create the interrelated connections between the functions.

Input: Is the initial arrow for a function or initialisation of the system and is typically given as a variable. As the functionalities are set in a sequence, the output from one functionality may prove as an input for another functionality.

Output: Is acquired from the input being transformed in relation to any set control information by the mechanism. The parameter given by the control arrow

can either come from an output from another functionality or as an internal parameter for the given functionality. This is required to enable the performance of the transformation of input to output.

Mechanism: May refer to the necessary person, tool or object that carry out the functionality. A mechanism may be shared among multiple functionalities.

If a system is of a very complex matter, usually more than six functions, it can be required to break down a system into further subsystems, though it is not deemed necessary in this project. The IDEF0 model can't stand alone, as though it gives a nice overview and simplify the activities, it lacks descriptive documentation. Therefore documentation for the arrows and functions are necessary to leverage the models use. A successful implementation of IDEF0 would lead to the correct setup of functionalities that can specify a system design, thus meeting the requirements.