FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Cost optimization in AGV applications

Guilherme Sperb Lawless

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. António Paulo Gomes Mendes Moreira

June 27, 2016

Abstract

Automated Guided Vehicles (AGVs) are autonomous vehicles used mostly in material handling applications. In today's industrial world, system flexibility is very desirable, that is, being able to quickly change system variables to adapt to presented challenges or production capacity changes. Due to technology advancements, highly capable robotic systems are available, and every year the adoption of AGV solutions increase. In spite of this yearly increase, the cost of AGV systems is still too high for some companies to fully commit to an AGV based solution.

In this dissertation, some ways of optimizing AGV costs are explored. In one way, the setup process of AGVs, which is currently done by the company selling the AGV, with proprietary solutions being the norm in the market. By shifting the setup process to the company adopting the AGV solution, the setup costs could be greatly reduced.

Other ways of optimizing costs are related to the technological advancements in AGVs, with increased flexibility when defining paths. By using positioning sensors such as the laser rangefinder and adopting the appropriate algorithms, the overall cost can be reduced in the setup and operating processes.

Lastly, the impact of layout choice and number of AGVs in a system is studied. For this purpose, a real world scenario of material handling is analyzed, and a simulation model is developed using the software Simio.

Today's industrial market is always looking for more cost effective solutions, where not only low resources but also high performance is necessary in performing every day's tasks.

Agradecimentos

Ao Prof. Dr. António Moreira um profundo agradecimento pela disponibilidade, pelo debate, pelas críticas construtivas e, acima de tudo, pelo saber e conhecimento partilhado comigo. Obrigado pela orientação e apoio incondicionais, sem os quais não teria sido possível ter realizado esta dissertação.

A toda a equipa com quem tive o privilégio de trabalhar na Continental, em especial à Dr.ª Eduarda Silva, ao Dr. Rui Bonifácio e ao Sr. Manuel Sampaio, um enorme agradecimento pela maneira como me acolheram e por todo o conhecimento que partilharam. Ter a oportunidade de passar por um ambiente empresarial enriqueceu esta tese e tornou o trabalho mais completo e interessante.

Aos meus colegas e amigos, por todos os incentivos, motivação nos momentos mais difíceis, e presença em momentos de desabafo que só podem ser conseguidos com quem compreende e passa pelos mesmos desafios. Em especial, à Cláudia, sempre apressada, que me entendeu e apoiou no decorrer dos últimos dois anos; ao Freitas, cuja presença é sempre marcada por bons momentos.

À Rita, um enorme obrigado pela paciência, pelas conversas, pelo carinho demonstrado e por nunca duvidar de mim e ser sempre a fã número um desta dissertação, mesmo não sendo entendida no assunto.

Aos meus irmãos, Leonardo e Ana Luísa, um profundo agradecimento por todos os momentos que fazem um dia de trabalho intenso ser recompensado.

Por fim, aos meus pais o meu mais sincero agradecimento por serem exemplo de trabalho, perseverança e resiliência, por me terem sempre incentivado a dar o meu melhor e a ver em todos os obstáculos meros desafios. Obrigado por me ancorarem e serem porto de abrigo.

Obrigado a todos!

Guilherme Sperb Lawless

Contents

List of Figures

List of Tables

Abbreviations and Symbols

AGV	Automated Guided Vehicle
AMCL	Adaptive Monte Carlo Localization
API	Application Programming Interface
CSV	Comma-separated Values
EKF	Extended Kalman Filter
FIFO	First-In-First-Out
HRI	Human-Robot Interaction
IMU	Inertial Measurement Unit
JSON	JavaScript Object Notation
IO	Input/Output
LIFO	Last-In-First-Out
MCL	Monte Carlo Localization
OS	Operating System
PF	Particle Filter
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
TOF	Time of Flight
UI	User Interface
WIP	Work in Process

Chapter 1

Introduction

1.1 Context and motivation

Nowadays, many industries see the benefits of using automated guided vehicles, known as AGVs. The AGV international market is growing and is predicted to reach a value of over $2000 million by 2020 [1].

Market competitiveness drives companies to strive for lower waste[1], high flexibility and productivity, long term benefits that usually come with the implementation of an AGV system, for instance in logistic applications. When correctly implemented, this kind of system comes with many advantages, such as continuous operation, with only the smallest deviations from the optimal work plan. In logistics, AGVs can also reduce the number of work accidents that often occur with manually operated forklifts, also lowering costs with damages to loads, infrastructure, vehicle and staff.

Nevertheless, the market share of AGV systems in transportation is still low, as many barriers to implementing AGV systems to transport of materials exist. The initial investment necessary to achieve precision, robustness and low maintenance and operation costs often means that these systems can only be implemented by companies with high profit and market share already. Oftentimes, the proposed solution isn't optimized to the goals of the particular system, resulting in a needlessly expensive solution when lower performance would meet the requirements. This happens because the AGV company usually has low flexibility in the solutions and no interest in developing optimized solutions for its clients, since it results in increased cost. In order to implement an optimized solution, there has to be a previous study of the requirements and possible solutions, especially when it comes to the number of AGVs to be used. This should include logistic simulations of the proposals, and deciding based on quantified criteria. This study is of greater interest to the client than the company selling the solution. For instance, the number of AGV vehicles sold is often higher than effectively necessary.

It is difficult to say what the overall cost of implementing an AGV system will be. Among other costs, the initial setup and operators' training costs are high. Depending on the technology

[1] Activities that do not add value to the final product

for localization used by the AGV, the setup costs can increase even more, such as a high number of artificial beacons. Robot configuration is often done by a specialized operator from the company selling the AGV. There is also a need for a connection with management software (e.g. MES and ERP), usually proprietary. There are different mechanical solutions to transporting materials, which encompass different traction solutions. For instance, an AGV can carry loads on top of it, or haul the materials, at different costs. All these costs add up and increase the final cost of implementing an AGV system substantially, especially for lower numbers of vehicles.

There is a need to lower all the previously mentioned costs, increasing the confidence of small and medium-sized companies in implementing AGV systems. The difficulties arise from the need of precision, robustness and safety, without sacrificing the flexibility in trajectory and workstation definition. The result is usually a need for higher cost technology and more complex navigation and localization systems.

The previously mentioned needs and challenges they present are motivation for this dissertation, which will explore ways for optimizing costs in AGV applications, especially in logistic applications.

1.2 Goals and scope

Different areas of knowledge are integrated when it comes to designing industrial mobile robots. Localization and navigation demand knowledge of computer algorithms, information theory, artificial intelligence and probability theory [2].

This dissertation concerns the exploration of ways to lower costs in setup and operation of AGV systems. Due to the scope of this challenge, there must be a selection of precise goals for this project, taking into consideration the background knowledge of the author. The work's focus will be on optimizing the cost of the robot's navigation system and the setup process of an AGV system implementation. Thus, the following goals are defined:

- Develop a configuration and operation interface for Human-Robot Interaction (HRI) that does not require a heavily specialized worker to operate and setup the system. Examples of operations are: map creation, waypoint and trajectory definition;
- Briefly study the impact of choosing more cost effective sensors for an AGV solution, from robustness to applicability in the industry;
- Analyze and evaluate the impact of adopting an AGV solution in a logistics application case study, with a simulation approach.

1.3 Document structure

This document is organized in six chapters.

The current chapter presents a short context for the topic, as well as the motivation to justify the scope and objectives of the project.

Chapter 2 provides an overview of the relevant software and hardware technologies, followed by the state of the art of localization and mapping techniques used in AGVs.

In chapter 3, the different possibilities for developing a human-machine interface are explored, and the development of a web-based user interface is presented.

Then, chapter 4 includes a brief study of the impact in choosing the positioning sensor and localization algorithms in an AGV system.

The impact of selecting a layout and the number of AGVs to be used in a system is studied in chapter 5, using simulation to present the case of a real world scenario.

Finally, chapter 6 presents the conclusions of this project.

Chapter 2

State of the Art

This chapter will be focused on the most adopted systems in the industry for indoor applications, as it would be impossible to cover every new system, technology or technique developed for mobile robots, and most will not have any application in the AGV industry.

2.1 Related software

Developing systems for mobile robots is a multidisciplinary problem that requires complex software and hardware systems to work together. In order to speed up development and deployment of new systems, frameworks and simulation environments are required.

2.1.1 Robot Operating System

The Robot Operating System[1] (ROS) is an open-source software framework that enables easier development for robot systems [3]. It is a widely adopted framework by the mobile robotics community.

ROS is based on a modular architecture and provides a structured communications layer above the host operating system, meaning code sharing and re-using is easier. Packages are software modules that can be written in different programming languages and, together with the hardware abstraction provided by ROS, can be shared and reused in robotic research and development. For this reason, there is an extensive list of open-source packages[2] and meta-packages[3] with examples to assist in development.

2.1.2 Operation and configuration interfaces

Most interfaces found in industrial AGV are proprietary and require configuration by a specialized operator. In research and development, most of the work is done by command-line interfaces or complex and hard to use software.

[1] http://www.ros.org/
[2] http://www.ros.org/browse/list.php?package_type=package
[3] http://www.ros.org/browse/list.php?package_type=metapackage

The MiR100[4] robot shown in Fig.2.1 is an AGV aimed at the logistics and healthcare industries for small transportation tasks. It also provides a web-based interface to control the robot, define waypoints for trajectories and view the map. This kind of interface is much simpler to the operator than what is usually found in the industry.

Figure 2.1: Left: The MiR100 robot. Right: Overview of a map in a tablet interface.

Relating to section 2.1.1, Crick *et al.* present in [4] the *rosbridge*[5] middleware between ROS and web services via websockets. This allows the developer to create interfaces in programming languages for web development.

The same author used this tool to allow easier collection of data in a large-scale experiment [5]. Older experiments that do not use this tool but use web-based interfaces have been done, but the layer of abstraction provided by this one is valuable in speeding up development.

Robot Web Tools[6] builds on *rosbridge* for its communication layer and provides a core set of libraries granting abstraction from ROS functionality. This enables development of interfaces without much knowledge of the ROS architecture. In contrast with other tools like *wviz*[7], Robot Web Tools focuses on lightweight and efficient communication with little overhead. An overview of the Robot Web Tools architecture can be found in [6], also showcasing different use cases.

2.1.3 System simulation software

Material handling efficiency has severe impact in production costs. Since this activity does not add value to the final product, it can be viewed as waste, and therefore the goal should be to minimize its execution time.

Implementing a flexible solution, such as an AGV system, can increase the efficiency of material handling. This requires development of a complex high level control system to coordinate

[4] http://mobile-industrial-robots.com/en/mir100-2/
[5] http://wiki.ros.org/rosbridge_suite
[6] http://robotwebtools.org/
[7] http://wiki.ros.org/wviz

robot scheduling and routing, that is, which robot does some route at any time. The optimal solution to this problem is often not achieved online[8], and an heuristic approach is taken [7].

Moreover, the cost of an AGV system means that the problem of estimating the number of robots required in a production process is also non-trivial. Approaches to this problem are usually model-based, these divided in two types: using analytical models or simulation models [8]. Simulation approaches present several advantages over analytical approaches for large-scale processes. Although it may not provide the most optimal solution, it is easier to use simulation models when the number of variables is very high.

The main benefits of using simulation in this case are: 1) finding hidden and unforeseen mistakes in the scheduling and coordination systems that are cheaper to solve before implementing a solution; 2) having an estimated improvement value of the production rate.

In order to investigate material handling solutions with AGV, the simulation software should be based on the discrete event simulation paradigm [9], provide support for automated material handling operations and preferably include AGV vehicle models. Examples of software having these characteristics are: Simio[9], FlexSim[10], Arena[11] and Witness[12].

Although all of the previously mentioned software systems should accomplish these tasks, each of them has diversified features and programming language support. These programming languages are usually divided in general-purpose programming languages and special-purpose simulation languages [9].

The choice of the simulation software is an important task in the simulation process. There isn't a clearly better tool for every situation, and so it requires some effort to make a direct comparison of products. Choosing the wrong tool may critically impact the success of the simulation process. Although there can be found work on quantitative comparison of simulation tools, such as [10], this information can become quickly outdated when simulation software developers continuously bring new updates, features and performance improvements to their software. Furthermore, a qualitative comparison is usually not complete without trying out the software itself and its features, as many of the application websites don't provide enough information for this kind of comparison.

Due to the applicability in this case and the fact that a license is already available for use, the Simio simulation software will be used. An overview of this software can be found in section 5.1.

2.2 AGV technology

AGV have been in development for over 50 years. New technologies have been developed, with some having a permanent effect on the market. The following sections review what is mostly used in the industry.

[8] When the system is running
[9] http://www.simio.com
[10] http://www.flexsim.com
[11] http://www.arenasimulation.com
[12] http://www.lanner.com/en/witness.cfm

2.2.1 Material handling solutions

Load size and weight can vary greatly in industrial environments. Both in traditional and AGV systems, the material handling technology must be suited for the purpose.

Traditional systems often rely on common forklifts or hauling trucks as a transportation system, despite the higher throughput realized by rollers or chain conveyors. The reasons for this are usually the reduced investment and flexibility when integrating the solution in an existing environment or moving to another facility [11].

AGV systems available in the industry use mostly standardized material handling solutions. Solutions that require the vehicle to directly carry loads usually increase the vehicle's cost and safety concerns, especially for heavier loads. The most common of these systems is the forklift AGV, with variations depending on load weight, such as: straddle, narrow and counterbalanced. For horizontal movement of loads, there are AGV with roller, chain or belt conveyors that can be used to transfer loads between conveyors or end-of-line transfer to stock. A simpler solution is the unit load AGV, which carries a load on top of it. This can improve the response time but requires automatic external solutions to place the load on top of the vehicle in order to be fully autonomous.

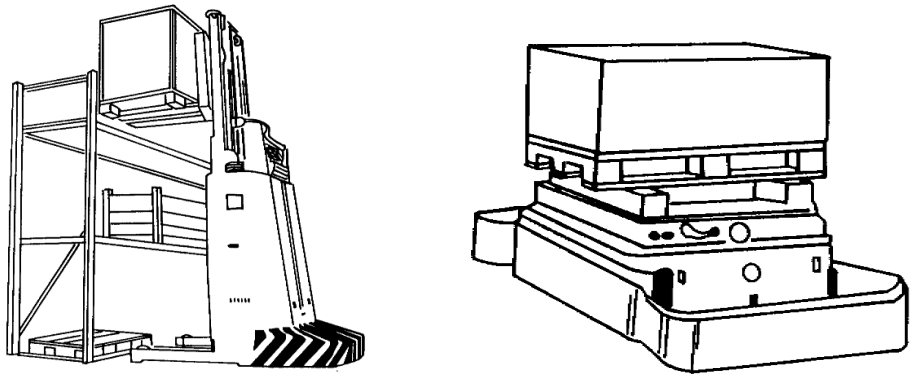

Figure 2.2: Left: The forklift AGV. Right: The unit load AGV. In [12].

Material handling solutions where the vehicle does not carry the load directly are usually cheaper, but with their own inconveniences such as occupying a greater area on the floor. Examples of these solutions are the tugger AGV and the tunnel AGV. While the tugger AGV can move a train of multiple trolleys, the tunnel AGV is used to move individual trolleys by placing the vehicle under trolley and using a hooking system to guide it to its destination. In these solutions, the load's weight is on the ground and is not directly carried by the vehicle. Thus, a similarly robust tugger or tunnel AGV can usually carry more weight than forklift or unit load AGV.

Recently, companies have focused on selling custom made solutions, with modular and flexible systems. These systems can be made exactly according to the application needs. This can lead to a more optimized solution, reducing cost and wasted space, and increasing efficiency. These solutions are in line with lean thinking, creating more value for the customer while using fewer resources.

2.2.2 Positioning sensors and techniques

An extensive and comprehensive description of indoor positioning technologies can be found in [13]. The sensors for positioning are usually either used for relative or absolute localization.

When using sensors and techniques for relative localization, also known as dead-reckoning, the robot's position is given by constantly measuring and integrating successive position differences. This leads to unbounded error accumulation over time. The signal output can be, for instance, linear velocity, acceleration or angular velocity. An inertial measurement unit (IMU) is a device that uses gyroscopes, accelerometers and magnetometers to estimate changes in acceleration, velocity and position of a moving vehicle [2] and is almost a requirement in aerial navigation. They are also important as providing redundant data to check for inaccuracies with other techniques.

The most widely adopted technique for relative localization, however, is odometry [14]. The fundamental principle in odometry is using wheel sensors such as encoders, potentiometers or resolvers to measure changing wheel data, with high sampling rates and good short-term accuracy. Because of the high sampling rate, it is usually fused with data from global positioning sensors.

Borenstein *et al.* [15] classifies errors in odometry into systematic errors and non-systematic errors.

Examples of systematic error sources in odometry are differences between the measured and real values of wheel radius and distance between wheels. Systematic errors can be minimized with calibration, usually conducting different experiments with the robot.

On the other hand, non-systematic errors are random and their effect could only be minimized by averaging, which is unwanted in odometry. Sources of non-systematic errors are: floor irregularities, unexpected obstacles, wheel slippage and finite encoder resolution. Other relative positioning techniques can be used to detect non-systematic errors such as wheel slippage. For instance, suppose a robot is rotating on itself. If the wheels were to slip, odometry sensors would register very disparate values when compared to the output of a gyroscope.

Overall, the errors in odometry and other techniques for relative positioning mean the estimation will not be accurate over a long period of time. Since the position is obtained by integrating, the errors will also be cumulative and lead to uncertainty in the robot's positioning. This is especially the case for the estimation of the robot's orientation.

Global positioning techniques estimate the robot's position in the world referential. To achieve this goal, a robot can detect known landmarks or beacons. These can be active or passive, natural or artificial. Landmarks usually refer to objects near the robot which can be identified. When near a landmark, the robot can update one or more variables of its global pose. This update is inherently intermittent, and can only be done when in the presence of landmarks. Beacons, on the other hand, can usually be detected at higher distances. A robot can measure its orientation and/or distance relative to a beacon. In order to update its global pose, the robot needs to simultaneously (or in quick succession) detect various beacons. The most common techniques to update a robot's pose when using beacons are: 1) trilateration (when measuring distances); 2) triangulation (when measuring angles); 3) filtering measurements (e.g. using a Kalman Filter).

Some sensors only allow the robot to move in fixed trajectories, many times not knowing its global position. In those cases, the robot may detect a marker so it knows where to stop, and continues to follow its trajectory when commanded. Guiding systems for fixed trajectories are usually magnet or wire guided. In magnet guiding systems either magnets are embedded in the floor or a magnetic tape can be placed above it. In wire guiding systems, an inductive wire is embedded in the floor and the robot uses its sensors to follow the wire's path. Other systems use optical tracking, either along lines or using ground markers such as QR codes. All the previous systems require the environment to be modified, and have increased costs, based on how extensive the trajectories are. The cost of changing the trajectories is also high. In the case of magnetic tape or ground markers being used, there is a risk of damaging the equipment.

On the other hand, sensors that allow measurements of distance and/or orientation can be used for flexible trajectories, either using acoustic or optical sensors. The most widely adopted of these sensors for indoor positioning is the laser rangefinder. Other techniques exist, such as visual positioning, but it is not widely adopted in the industry so far.

2.2.3 Laser rangefinder

The laser rangefinder can be used for global positioning of a robot, as well as obstacle detection. The advantages over other sensors with the same application are increased accuracy, sample rate and range. It uses a narrow laser beam to determine the distance to an object. The most common form of this sensor operates on the time of flight principle, by sending a short laser pulse towards an object and measuring the time it took for the pulse to be sent and received again [16]. Fig.2.3 shows a simplified block diagram of a time of flight laser rangefinder. Using this technique, the sensor can often achieve centimeter or millimeter accuracy, and tens of meters in range. Other specifications for laser rangefinders are the frequency and angular resolution.

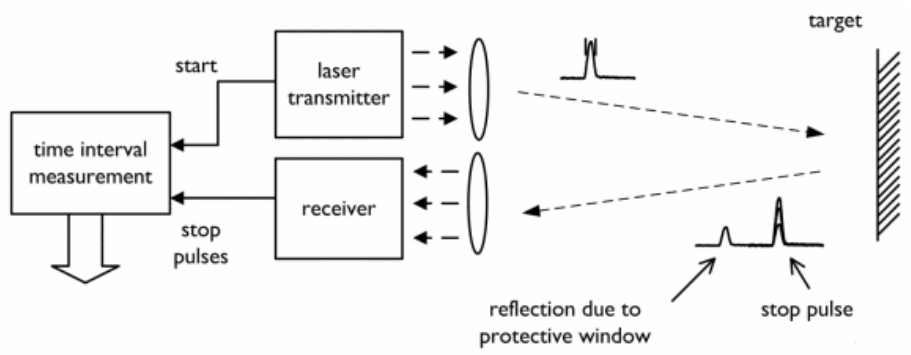

Figure 2.3: Block diagram of a TOF laser rangefinder. Adapted from [16].

The operation of a laser rangefinder consists of taking a measurement, then rotating the laser for a fraction of its full scan range, and taking a new measurement. Due to mechanical constraints, the laser isn't able to take measurements in its dead zone, also known as blind region, as depicted in Fig.2.4.

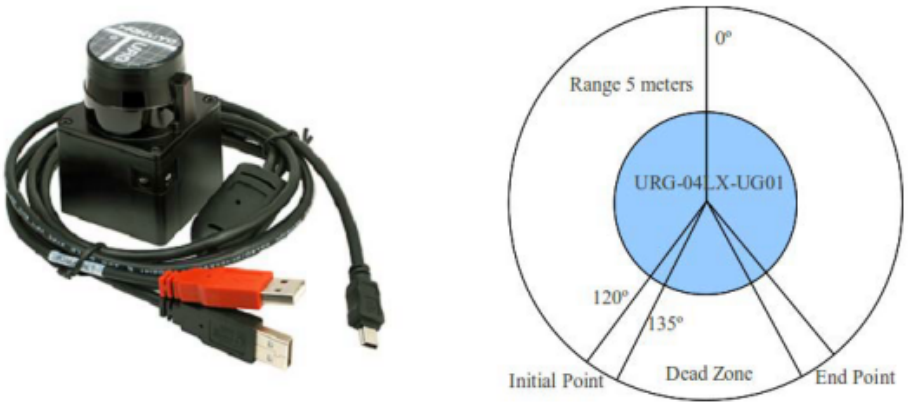

Figure 2.4: Left: Laser rangefinder Hokuyo URG-04LX-UG01. Right: Laser rangefinder scanning and dead zone. In [17].

2.3 Indoor localization techniques and algorithms

Due to the issue of unbounded error in odometry, as explained in section 2.2.2, robust techniques for localization must use sensors for global positioning, usually fusing data from both sensors. Although there is an extensive list of localization techniques for mobile robots, this review will focus on those which are used in the AGV industry and use laser rangefinders.

The localization problem is the problem of estimating a robot's pose relative to a known map, and usually techniques are based on the Markov assumption, that is, a static world and independent noise assumption. While some techniques can show robustness even in dynamic environments, others fail in the solution to the localization problem. The map of the environment is a list of objects along with their properties. Maps can usually be indexed in one of two categories: *feature-based* and *location-based* [18]. The latter can be, for instance, a point cloud map or a occupancy grid map. Localization techniques can also be divided in that way.

Siegwart *et al.* [2] state that the multiple-hypothesis belief representation is the key to enabling robots with limited sensory information to navigate robustly in a varied array of environments. Multiple-hypothesis belief is the representation of multiple possible solutions for the robot's pose. The most common approach to the uncertainty is representing it explicitly using probability theory - probabilistic robotics.

There is no single technique that solves every localization problem, and there are many recent works on the subject. By now, the most common approaches are: Extended Kalman Filters (EKF) and map matching for landmark-based maps; particle filters and map matching for grid maps.

In [19] a laser scanner is used, along with indistinguishable artificial landmarks for global localization, with a matching technique. Fig.2.5 illustrates this scenario. Relating to the setup of this solution, the artificial landmarks, especially with numbers in hundreds as is the case, can increase setup costs dramatically, mostly in installation costs. The work in [20] explores the perfect match algorithm [21] using natural features of the environment. The robot's pose is estimated by minimizing the fitting error between the data acquired by sensors and the known map. The result

Figure 2.5: Localization using artificial landmarks. In [19].

was a repeatability of 5 mm in a laboratory environment. Other matching related work can be found in [22, 23].

The Monte Carlo Localization (MCL) family of algorithms is presented in [24]. Building on the basic algorithm, a more robust one is developed. In MCL, the robot's belief is represented as a set of weighted particles. The AMCL[13] (Adaptive MCL) algorithm is an implementation of the KLD-Sampling, an adaptive particle filter described in [25]. This adaptive approach increases the efficiency of particle filters by adapting the particle count depending on the approximation error in the localization procedure.

Usually, AGV have laser scanners above the robot (see Fig.2.5), making it easier to detect artificial landmarks and harder to be obstructed by obstacles. One interesting approach in an industrial environment is [26] which uses an AGV security laser scanner (required for safety in obstacle detection, or other methods) as the laser for localization. It uses artificial beacons placed in the environment. The localization algorithm is the EKF with unknown correspondences as described by Thrun *et al.* in [18]. This algorithm estimates landmark correspondences via a maximum likelihood estimator, though there are alternatives, such as based on the Euclidean distance [19]. The data from landmark correspondences and odometry sensors is fused in an EKF. The challenges presented by the use of a laser scanner near the ground are related to obstacle and outlier detection, and thus a filter was used.

2.4 Autonomous map building

Section 2.3 had an overview of different techniques used for robot localization. All the discussed strategies require the existence of a map of the environment. This map can either be built by hand or autonomously by the robot. Building a map by hand can be costly and time-consuming, especially for large environments such as a warehouse. When a robot is capable of sensing its

[13] http://wiki.ros.org/amcl

environment through exteroceptive sensors, it makes sense for the robot to build its own map. The goal of autonomous map building can be expressed as follows: "starting from an arbitrary initial point, a mobile robot should be able to explore autonomously the environment with its on-board sensors, gain knowledge about it, interpret the scene, build an appropriate map, and localize itself relative to this map" [2].

The previous sentence exposes the main challenge with autonomous map building. Indeed, the robot must build a map at the same time as positioning itself in the environment. This is known as the Simultaneous Localization and Mapping (SLAM) problem. A mathematical definition of SLAM can be found in [2] and an in-depth study of the problem in [18].

Comparing SLAM algorithms is important both from a perspective of map accuracy and the computational cost. The choice of the right method depends on the the type and number of features in the environment, the desired resolution of the map - which is directly related to localization accuracy -, computational time, and other factors [2].

EKF-SLAM is the most popular approach to SLAM [27]. This approach represents discrete landmarks as part of a state-space framework. Many successful implementations of this approach in indoor applications are reported. EKF-SLAM inherently has some problems for a large number of landmarks, as the computational complexity increases quadratically with the number of landmarks. Another shortcoming in their application is high sensitivity to failures in landmark association. Efforts to increase the efficiency of this approach are: working in local sub-maps [28] and Extended Information Filters [29], with results for outdoor environments in [30] achieving constant-time update steps.

The previous techniques are known as filters and address the online SLAM problem, where only the current pose and the map is estimated at any given time. Smoothing approaches such as graph-based SLAM techniques [31, 32] address the full SLAM problem, where the full trajectory is estimated. These approaches are also known as optimization-based approaches.

Particle Filters (PF) are another approach to SLAM. Here, the posterior probability is represented by a set of weighted particles and each particle is given an importance factor. Montemerlo *et al.* presents the FastSLAM technique in [33], where each particle uses N Kalman Filters to estimate the position of the$N$ landmark locations. This approach has complexity $O(P\log(N))$, where P is the number of particles used. The map created by this approach is a feature-based map. An approach based on PF that creates an occupancy grid map is [34], building on the grid-based FastSLAM algorithm as described in [18]. This algorithm is due to the work of Hähnel *et al.* in [35].

Relating to ROS, Santos *et al.* [36] provide an evaluation of 2D SLAM techniques available in ROS. The laser rangefinder used was an Hokuyo URG-04LX-UG01. Results for two indoor environments using a metric based on map accuracy show that the most promising algorithms are *HectorSLAM*[14], *gmapping*[15] and *KartoSLAM*[16].

[14] http://wiki.ros.org/hector_slam
[15] http://wiki.ros.org/gmapping
[16] http://wiki.ros.org/karto

HectorSLAM is based on robust scan matching from sensor data, thus taking advantage of better performance laser scanners. Gmapping is an implementation of the algorithm described in [34], using PFs to create an occupancy grid map and getting different results depending of the number of particles used. Finally, KartoSLAM is a graph-based SLAM algorithm which creates a landmark-based map.

Chapter 3

Human-Machine Interface

In service robotics, interfaces are created so the user can fully utilize the robot's features. Due to the diversity of the robot's features, every interface tends to be unique and is made for a specific robot model.

When it comes to industrial AGV, the robot's purpose is usually to transport loads, although much more complex goals exist, such as carrying a robot manipulator for wielding purposes. For most mobile robot purposes, such as material handling, the robot's control scheme is usually similar to the one in Fig.3.1. If both the robot's purpose and it's internal control scheme are standardized, then building a unified setup and operating interface is facilitated and advised.

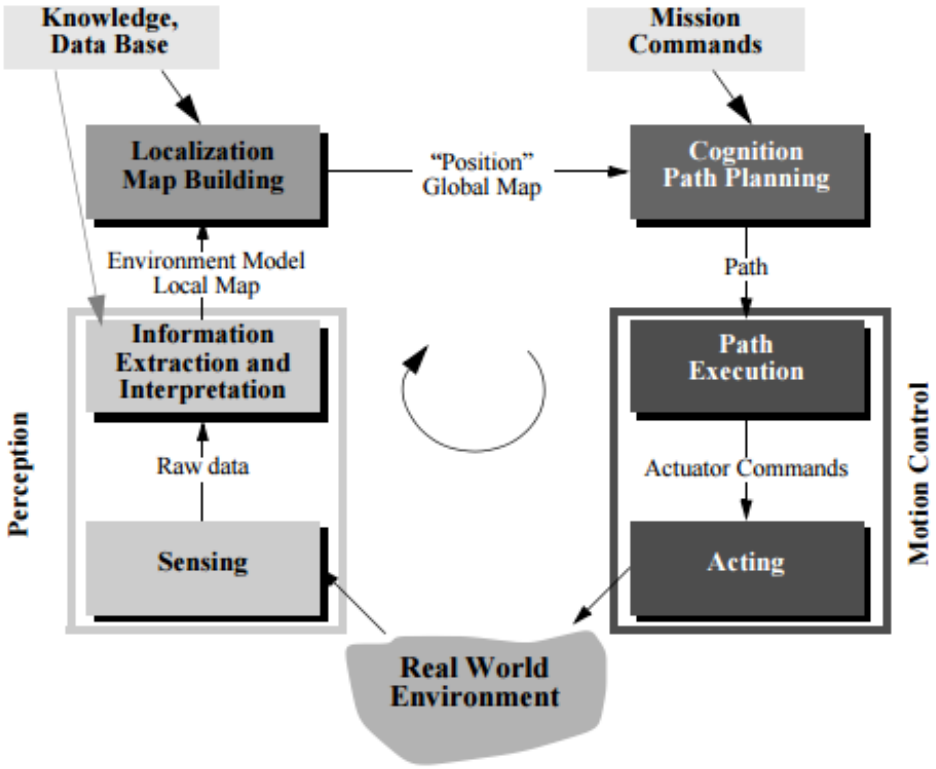

Figure 3.1: Reference control scheme for mobile robot systems. In [2].

This chapter concerns the problem of the high initial costs of configuring one or more AGV, which is related to the usually proprietary interfaces bundled with AGV in the market.

Section 3.1 provides an analysis on this problem, while characterizing the environment for which the interface will be created.

In section 3.2 the different solutions to the problem at hand are discussed.

Then, section 3.3 consists of a description and overview of using web technologies for building interfaces in the ROS environment.

The development process of a tool for remote process launching using ROS is presented in section 3.4.

Finally, section 3.5 concludes the chapter presenting the results.

3.1 Problem analysis

The interfaces for industrial AGV are created by the manufacturer and proprietary, which ensures the buyer will be tied to their software, and potentially only compatible hardware. This increases the total initial cost of an AGV solution. Furthermore, configuring the robot(s) is usually done by the manufacturing company, and not by the less expensive company operator and/or engineering staff. A more standardized interface could potentially decrease these costs, but it must also be easy enough to learn and use, achieving the previously stated goals. In order to provide a setup and operating environment in these conditions, a simple interface will be presented, achieving the basic goals of interaction with an AGV.

For the purposes of the development of an interface, the following scenario is considered. An AGV with material handling capabilities is available. The positioning sensors available are wheel encoders and a laser rangefinder. The wheel encoders are used for relative positioning through odometry, and the laser rangefinder is used to obtain a global position within the environment, by making use of different mapping and localization algorithms (see 2.2 and 2.3). The robot's software is built on ROS, using its standard data types and communication layers.

The robot needs a map of the environment to obtain a position relative to it. This map should be created by the robot itself, in order to reduce costs. After the map is created, the user should define a trajectory for the robot to follow indefinitely. In this scenario, the map is two-dimensional and a trajectory is defined by curved lines between 2d points. Thus, the initial setup consists of the following steps:

1. Starting the robot;
2. Initializing the mapping algorithm;
3. Moving the robot manually, if required; [1]
4. Ending the mapping algorithm;
5. Making manual changes to the output map, if necessary;
6. Creating, editing and deleting waypoints in trajectories;

[1] If the algorithm is an Active SLAM one, for instance, the robot will create the map by itself

7. Saving the trajectory.

The operation process is simpler: the operator should choose the map of the environment and trajectory to follow, and begin the execution of that path.

The scenario and processes described lead to a requirement analysis, resulting in Table 3.1, where the primary and secondary requirements are presented, respectively, with P* and S* IDs.

ID	Name	Description
P1	Mapping	Interact with the robot's mapping algorithms in order to build a map of the environment and save it for future use
S1	Visualize mapping	Provide updates of the current map being built
P2	Trajectory editor	Add, edit and remove trajectories and points in trajectories
P3	Execution	Choose one from the available trajectories and execute it
S2	Visualize execution	Show the robot's position within the map of the environment and the path it's executing
S3	State	Provide information on the robot's state, as well as alerts
P4	Ease of use	The setup and operating processes should be easy to complete

Table 3.1: Interface requirements.

While using the interface should be made simpler, it is the robot's software which will dictate how complex the processes behind the interface are. The main challenge and ultimate goal in creating an interface is putting all the hard work in its designer and programmer, while letting its use be simple and straightforward. The background processes will depend on the robot's software applications. In the case of ROS, a framework developed from the ground up to encourage collaborative robotics development, many of the background processes necessary for an HMI are already available, as will be seen in section 3.3.

3.2 Possible solutions

When designing an user interface (UI), one is faced with a decision concerning the technologies and programming languages to be used. There are mainly two kinds of UIs: web based and desktop based interfaces. The decision on which type of interface to design will be based on many factors, such as:

- The application goal;
- The convenience and applicability of the programming languages;
- Internet access availability;
- Specific hardware connection requirement;

- Update frequency requirement;
- Complexity of the setup process;
- Cross-platform access requirement;
- Who will use the interface.

Most UIs built to use with ROS applications are desktop type interfaces. The main reason for this is the existence of tools such as rviz[2], a visualization tool for ROS with custom plugin support. With rviz, most tasks of system debugging by watching the robot's internal state can be accomplished. It is mostly used by researchers in robotics, and does not easily solve the problem of making the interface friendlier to an operator. Other interfaces are developed as desktop applications, making it easier to use system processes. These are also very versatile, but suffer from one critical disadvantage: they are OS dependent.

The other kind of interfaces to be used with ROS applications are web-based interfaces. Web development has been on the rise for years, both on client and server side applications. In the ROS community, some applications and frameworks to work with web applications have been built, as mentioned in section 2.1.2. Unlike the previous type of interfaces, web interfaces are not OS dependent, and a developer can more easily create an interface to be used, for instance, in mobile devices with a modern web browser. Web development also increases scalability in high-performance network programs with Node.JS server-side applications [37].

Developing a web-based interface will allow the operator to use any modern device and not be restricted to an OS supported by ROS. Additionally, it allows the developer to take advantage of the wide array of high-performance Javascript graphics libraries available. Furthermore, the flexibility and robustness of the network layers in web applications will mean the web interface can easily be stored in a web server and be easily accessed either locally or remotely. As such, a web-based interface will be designed for this problem.

3.3 Web development for ROS

The integration of web applications and ROS is facilitated by *rosbridge* (see 2.1.2), which provides ROS API functionality to non-ROS programs. It works by converting JavaScript Object Notation (JSON) based commands to ROS functionality. Using this tool, any application that can send JSON commands will be able to take advantage of the ROS middleware. As of version 0.7.14, *rosbridge* is distributed in the form of the *rosbridge_suite*[3] meta-package, consisting of the following packages:

- *rosbridge_library*[4], which provides the core JSON to ROS API functionality;

[2] http://wiki.ros.org/rviz
[3] http://wiki.ros.org/rosbridge_suite
[4] http://wiki.ros.org/rosbridge_library

- *rosbridge_server*[5], providing a websocket transport layer to rosbridge;
- *rosapi*[6], which exposes core ROS functionality to rosbridge clients. Examples are: getting and setting parameters, getting the existing topics and services list.

If desired, other servers providing the transport layer in different forms can be developed and used with *rosbridge*.

With the core functionality of ROS available to web browsers via *rosbridge*, the *roslibjs*[7] Javascript library will allow interaction with ROS from a web browser, providing most functions available to standard ROS programs, such as publishing topics, calling services and performing actions. Although still under development, this library is used by hierarchically higher level libraries to perform high-performance tasks in robotics.

The basic system communications of a robot with ROS and a *rosbridge* server connected via websockets to a client using the *roslibjs* library as its API is shown in Fig.3.2. The ROS Master is the main program in a ROS based system, enabling all the core ROS functionality.

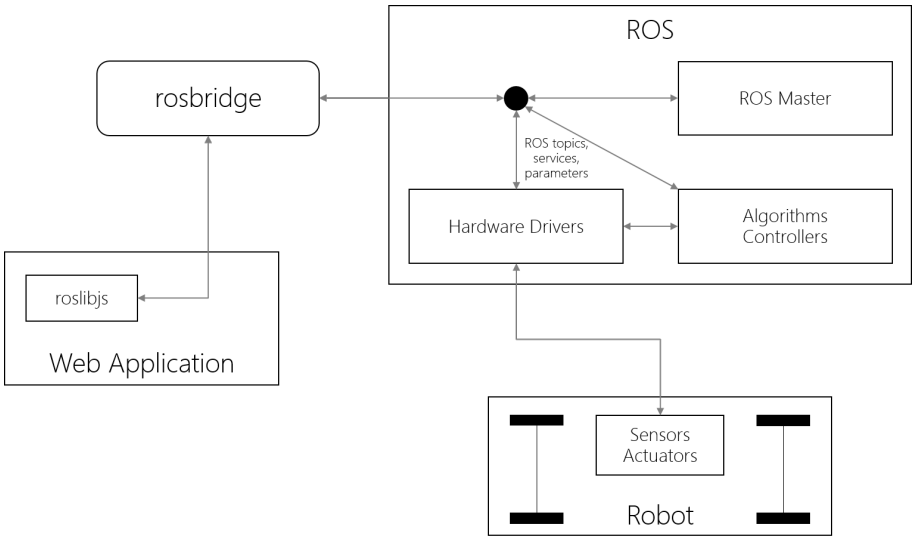

Figure 3.2: Rosbridge server and client connections in a ROS based system.

When it comes to tools that have similar purpose to *rviz*, there are various libraries under development. Two of the most important libraries are *ros2djs*[8] and *ros3djs*[9], both part of the Robot Web Tools effort. Both libraries are standard Javascript visualization tools for ROS, one focusing on 2D visualization, while the other focuses on 3D visualization. HTML5 support is required to use any of these libraries, making its use restricted to modern web browsers. Using these libraries usually doesn't require any additional non-web development, although there are exceptions such

[5] http://wiki.ros.org/rosbridge_server
[6] http://wiki.ros.org/rosapi
[7] http://wiki.ros.org/roslibjs
[8] http://wiki.ros.org/ros2djs
[9] http://wiki.ros.org/ros3djs

as using *tf2_web_republisher*[10] to throttle the high frequency *tf*[11]http://wiki.ros.org/tf updates to *rosbridge* clients.

The *ros2djs* library features map and image visualization tools and uses *EaselJS*[12] as its graphics library, allowing to build high-performance interactive 2D content in HTML5. Building on *ros2djs*, *nav2djs*[13] adds interaction with mobile robots by rendering the robot's current position over a map. The user can also send navigation goals to the robot by double clicking a position on the map.

On the other hand, *ros3djs* uses the *three.js*[14] library to create interactive 3D graphics in a browser using the WebGL API. This allows for interaction with ROS interactive markers, as well as displaying a 2D map in a 3D space.

While using the previously mentioned libraries should require little to no knowledge of Javascript to the developer, adding more advanced features certainly will. Some contributions were made to both the *ros2djs* and *ros3djs* repositories. In *ros2djs*, continuous map updates were fixed and an example of this feature was added to show how this library can be used for mapping visualization purposes. This feature was also added to *ros3djs*. Using this feature, the HTML5 canvas containing the map will be regularly updated as the robot builds its map of the environment using some mapping algorithm and sends map updates. Other minor fixes were pushed to the latter library's repository.

Recent developments in the ROS community have brought a *Node.js* implementation of ROS which allows *Node.js* applications to take advantage of the ROS API directly. This library is named *rosnodejs*[15] and is available for the latest ROS release, Kinetic Kame[16].

Node.js is a runtime environment for developing server-side web applications, with inherent asynchronous I/O capability and an event-driven architecture. It is a very promising tool which allows Javascript frameworks to be developed for robotics. Although unrelated to ROS, it is important to mention the most widely known robotics framework using the *Node.js* runtime: *Cylon.js*[17].

3.4 Remote Launch

The available tools in the ROS community facilitate the development of web user interfaces, but building the interface still requires moderate Javascript and HTML knowledge, at least.

One of the most basic tasks consists of launching processes in the server, such as the mapping process. In ROS, this is usually accomplished via *roslaunch*[18], a tool which allows for launching multiple ROS nodes, specifying their parameters, among other features. Many ROS packages

[10] http://wiki.ros.org/tf2_web_republisher
[11] \unskip\penalty\@M\vrulewidth\z@height\z@depth\dp
[12] http://createjs.com/easeljs
[13] http://wiki.ros.org/nav2djs
[14] http://threejs.org/
[15] http://wiki.ros.org/rosnodejs
[16] http://wiki.ros.org/kinetic
[17] https://cylonjs.com/
[18] http://wiki.ros.org/roslaunch

come with launch files, which specify the ROS nodes to run. These files are read by *roslaunch* with a system command similar to: *roslaunch package_name file.launch*. These launch files can be chained hierarchically to allow for more complex configurations, a very useful feature when setting up advanced processes such as mapping and self-localization.

In order to allow quicker web development for ROS in the future, a tool for using launch files via web browsers is proposed. The purpose of this tool is to give a ROS Node the capability to have both typical ROS node and desktop based interface capabilities. Due to the limited ROS API functionality concerning launch files, that is, the inability for a ROS node to launch other nodes, the procedure for running processes will be using system commands directly. Using this method, it is possible for the user to call any system commands as if running them from a basic command-line terminal. Thus, a user can interact with ROS via commands such as *roslaunch* and *rosservice*.

Fig.3.3 illustrates how this tool accomplishes system-level process launching while retaining ROS functionality. The ROS Master communicates with every ROS node to give it ROS functionality. The Remote Launch Server receives commands to either start or stop specific processes through system commands sent to the operating system. These commands can be either ROS related commands, which communicate with the ROS Master, or other software related commands. In the figure, two types of Remote Launch Clients are depicted: a ROS node client, and a web client. Due to the way ROS standardizes its messages, both clients will use the same API to communicate with the Remote Launch Server. As the purpose of this work is to build a web-based interface, only the web client will be developed.

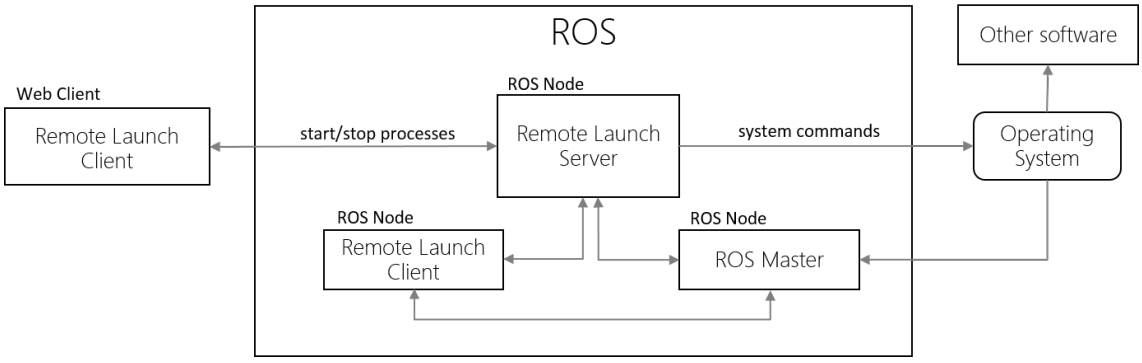

Figure 3.3: The Remote Launch tool's architecture within ROS.

The method for launching processes to be used results in a big disadvantage for the Remote Launch tool, which is the fact that the remote process launcher ROS node will not be able to easily view the state of the process launched using system calls, by parsing the output of the application. The program has no knowledge of the process being executed, and as such can't communicate with it. That functionality is sacrificed in order to make the tool more generic, achieving both faster and easier development and usage. In future work, one could define special processes such as ROS node launching and parse their output in a specific way, which would provide information of its state to clients.

The *remote_launch* package was developed as a result of this work and made available in an open-source repository[19] to be used by the ROS community. It consists of a server package and a web-based client library with a working example provided. In the following sections, the Remote Launch Server and Remote Launch Client applications are described in more detail.

3.4.1 Remote Launch Server

In this section, the Remote Launch Server is described, a ROS package developed in the course of this work that has process execution capabilities and is fully adapted to ROS standards.

Most ROS packages are written either in C++ or Python. The former is preferred when a high-performance task is to be accomplished, such as robot self-localization. Python was preferred for the development of the remote launch server due to the *subprocess*[20] and *psutil*[21] modules being available to accomplish the system process execution tasks. As the tool in development does not require high-performance execution, special attention to its performance was not required.

In terms of security, it is defined that the clients (web or ROS) should not be able to launch any process in the server, and instead should choose from a list of available processes. The server is configured by modifying a CSV file, where each line corresponds to a process and a process is defined by three properties:

- Name - which will be the title presented to the clients;
- Command - the command that will be executed by the operating system;
- Working directory - used by some processes for relative paths.

Using this method, the server is easily configurable by the developer and although multiple configuration files can be created, only one can be read by the server when launching. In spite of this method, when launching a process, the client can choose to add custom arguments to the default command.

Custom arguments are necessary when specifying an input file to a process, for instance. These arguments are filtered by the server to disallow chaining multiple commands, even though command chaining is allowed in the configuration file. Examples of argument filtering can be found in Table 3.2. If the filtered argument is different from the argument, the argument itself is ignored and only the command is used. If the filtered argument is the same as the original argument, then the argument is concatenated at the end of the default command. Although this results in less flexibility when setting custom arguments, this method proved to satisfy the requirements for this project. In future work, allowing commands to be entirely modified could prove valuable, although the security concerns would increase. One example of a method to achieve this purpose would be to define the arguments a list of strings, each taking a position defined by a special character in

[19] https://github.com/guilhermelawless/remote_launch
[20] https://docs.python.org/2/library/subprocess.html
[21] https://github.com/giampaolo/psutil

the defined command. The decisions taken for the purpose of this project proved to achieve the desired versatility while having a moderate level of security.

In a situation where security is not an issue, the security measures imposed to the clients could be removed to allow for further flexibility in arguments and allowing custom process chaining.

Command	Argument	Resulting filtered command
roslaunch example_package example_launch	arg1:=value1 arg2:=value2	roslaunch example_package example_launch arg1:=value1 arg2:=value2
rosrun map_server map_saver	-f map_name	rosrun map_server map_saver -f map_name
rosrun example_package example_node	& rosrun example2_package example2_node	rosrun example_package example_node
find example_file /	| grep example_text	find example_file /
qtcreator	; rosrun rviz rviz	qtcreator

Table 3.2: Example of filtering arguments in the Remote Launch Server.

The server periodically publishes (using ROS topics) a list of the processes available for clients to launch, including its execution state (either running or not). A client can execute the desired process using a custom ROS service where the process identification number (available in the published process list) is specified. Feedback of the state of the process is sent to the client after starting or stopping a process execution. The client can then check the process state periodically using the published process list.

Table 3.3 summarizes the use of standard and custom ROS data types and definitions. The triple dashes in the service definitions separate the input arguments from the return arguments. The server publishes periodically to the *list_launch_files* topic an array of RemoteLaunchFile message types, each containing a specific process information. Using information from this list, the client can use the StartLaunchFile or StopLaunchFile to start or stop, respectively, a process in the server. In the case of the StartLaunchFile service, a string of arguments is accepted, as previously described.

The process termination task is accomplished as follows: first, the server finds, recursively, any child processes that have been created by the original one. Then, it sends the UNIX interrupt signal (*SIGINT*) to each child process, as ROS nodes are prepared to receive this signal and safely terminate. For non-ROS processes, the server then tries to send the termination signal (*SIGTERM*). Finally, if any processes remain, the kill signal (*SIGKILL*) is sent, ending the termination task.

This application was tested in Ubuntu 14.04 using ROS Indigo[22], being proven to initiate and terminate any process successfully. Furthermore, custom arguments sent with the purpose of chaining multiple commands were filtered by the server.

As a result of being developed for ROS, this tool can be used not only for the purposes of developing a web interface, but for any application requiring basic process management. In order to make use of this tool, the developer needs no knowledge of system processes. When developing an application on top of ROS, it can be a powerful tool while requiring little configuration.

[22] http://wiki.ros.org/indigo

Name	Type	Definition
RemoteLaunchFile	ROS message	uint8 id string name string command string working_directory bool running
RemoteLaunchFileArray	ROS message	RemoteLaunchFile[] rlf_array
list_launch_files	ROS topic	RemoteLaunchFileArray msg
StartLaunchFile	ROS service	RemoteLaunchFile rlf string args - - - bool success
StopLaunchFIle	ROS service	RemoteLaunchFile rlf - - - bool success

Table 3.3: ROS standard definitions used by the remote launch server.

The name, remote launch server, was the initial name thought for this application, as it would serve the purpose of using ROS launch files remotely. Even later, as its scope changed to include any kind of system process, the name remained.

3.4.2 Remote Launch Client

The remote launch server previously discussed can be used in three ways: using ROS command-line tools, using the ROS API in developed nodes or using a user interface. The remote launch client was built as a Javascript library to use with the web interface, as discussed in 3.2. This library can be found in the same open-source repository[23], created by the author of this work, as the server. It follows the ROS Javascript style guide[24] and is named *remotelaunchjs*. This library was developed to be used by developers with little web development experience, and is meant to be used with the remote launch server described in 3.4.1.

The core Javascript library for web development with ROS, *roslibjs*, depends on *EventEmitter2*[25]. This library is an implementation of the NodeJS *EventEmitter* module that is browser compatible. It allows to fully decouple the producers and consumers of events, providing more flexibility in development than the usual callback pattern. Thus, the remote launch client was also developed using this library.

[23] https://github.com/guilhermelawless/remote_launch
[24] http://wiki.ros.org/JavaScriptStyleGuide
[25] https://github.com/asyncly/EventEmitter2

Furthermore, as the standard Javascript libraries that provide ROS 2D visualization, *ros2djs*, uses *EaselJS* to power its graphics, the remote launch client will make use of it. The goal is to make the setup for this library as easy as possible. Importantly, *EaselJS* also provides support for touch-type devices such as modern mobile phones.

The interaction is achieved through on/off buttons displayed using *EaselJS*. The buttons are displayed automatically in an HTML5 canvas, which can be configured simply by setting the number of buttons per column property. The client will display the buttons arranged correctly, as long as there is a connection to ROS through *rosbridge*. More advanced features can also be used, such as scaling the whole canvas to change the button's size, setting a variable to define which of the process IDs available should be shown, and setting a list of tooltips to show when mousing over each button. Choosing only some process IDs to be shown is useful to build a user interface with different tabs associated to different tasks.

Fig.3.4 shows how each HTML element is displayed in this library. Each button has a name, state and arguments properties. The name is set by the remote launch server's published processed list, the state changes when the button is clicked, and the arguments can be set at any time. In this example, the property dictating how many buttons should be displayed in each column is set to 3, although only one process information is being sent by the remote launch server, resulting in only one button being displayed. On the bottom, a sample tooltip text is shown when the user mouses over the button.

When the web page is loaded, the client verifies if any process is already running by analyzing the first message sent with the list of available processes. For each process already in execution, an event is emitted using the *EventEmitter2* library. This event, as well as a stopping event, is also emitted when clicking a button to start or stop a given process, respectively. This feature is what allows other Javascript objects to know when a process is started or stopped, and will be proven useful when setting up other components of the full interface.

The developed library can be used by any web developer in need of a way to remotely launch processes, as long as ROS is being used. Some flexibility was achieved, although some improvements are necessary for future work. Some examples for improvements are: setting different button sizes and color pallets; changing the order the buttons are displayed.

Finally, a complete working example of this library can be seen in its open-source repository. It is not shown in this section as its behavior is similar to what will be accomplished in the complete user interface.

3.5 Results

Given the scope of this project, the user interface was developed for an AGV with a fully working mapping, navigation and self-localization system - Jarvis. This robot uses standard ROS data types and algorithms to accomplish its tasks. As such, these tasks can also be accomplished with a web user interface, using the previously described libraries such as *ros3djs*.

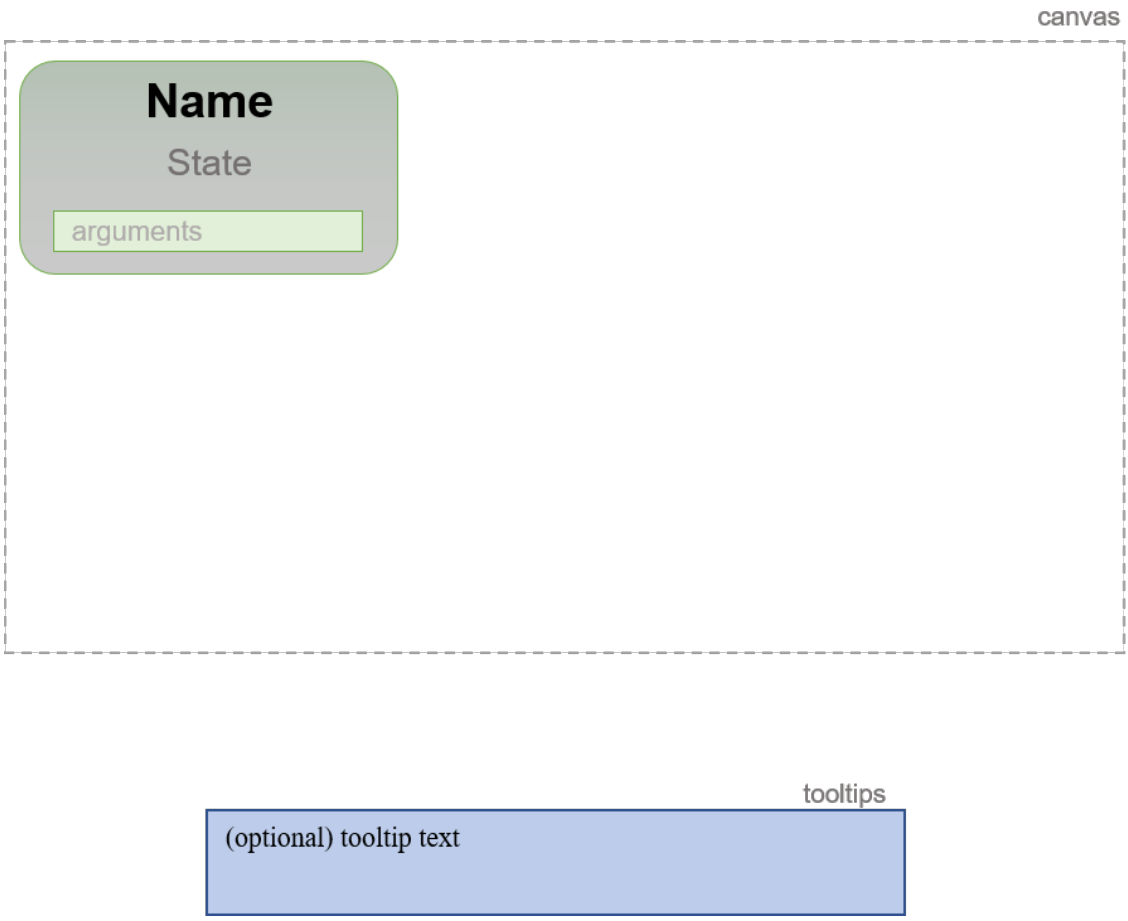

Figure 3.4: Displaying the HTML elements in *remotelaunchjs*.

The user interface is split in four pages, one is the initial setup page, and the others are used for each of the P1, P2 and P3 requirements in Table 3.1. In order to help a first-time user, all the buttons show tooltips that guide the user through every process.

The connection page is the landing page of the interface, which guides the user towards a successful connection setup. This page is represented in figure 3.5. The only step necessary to start using the user interface is using *roslaunch* to execute a launch file on the robot's system. Alternatively, one could execute this process as the robot's system initiates and no additional steps would be necessary for the user. The connection status is shown, which will be either "Not connected" or "Connected", meaning the page can or can't connect to the *rosbridge_server* ROS node.

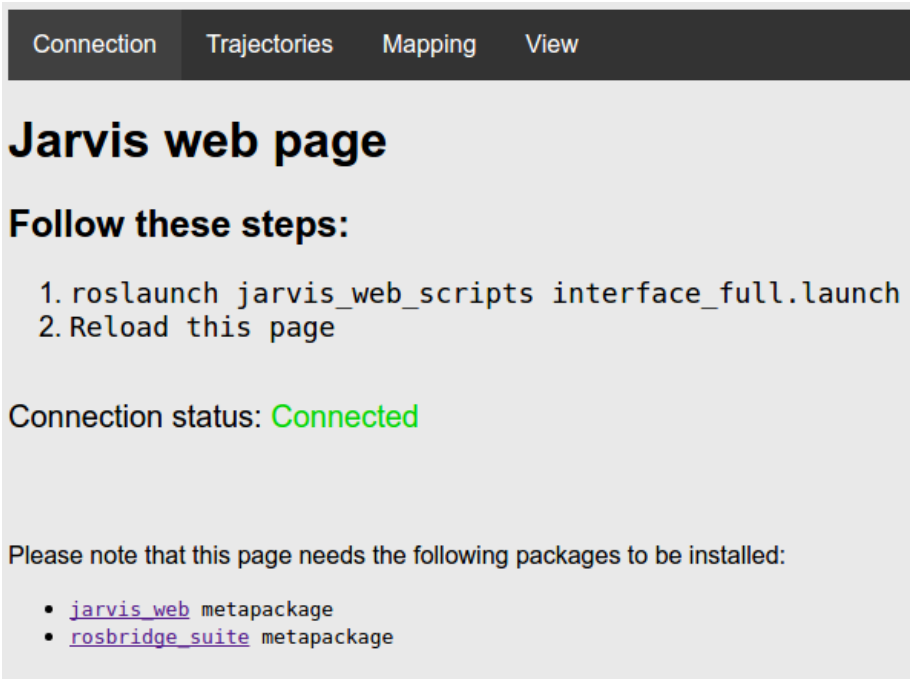

Figure 3.5: The interface's Connection page.

For mapping purposes, the robot has several working algorithms, and an example was chosen for demonstration purposes. The mapping process is standard in ROS, thus visualization can be accomplished with either *ros2djs* or *ros3djs*. Here, *ros2djs* was chosen as the map being built is two-dimensional. Within the interface's Mapping page, the user can start and end the mapping process, as well as save the map currently being built using only two buttons. When the mapping process is started, it also loads the joystick drivers so it can be used to move the robot through the environment. Fig.3.6 includes several steps of the mapping process within the interface, and Fig.3.7 shows the built map which is saved after pressing the "Save Map" button.

For trajectory editing, the robot has a fully developed trajectory editor which uses the *interactive_ markers*[26] system from *rviz*. This system is also featured in *ros3djs*, thus this library is used for the same purpose. The Trajectories page within the interface is shown in Fig.3.8. In this

[26] http://wiki.ros.org/interactive_markers

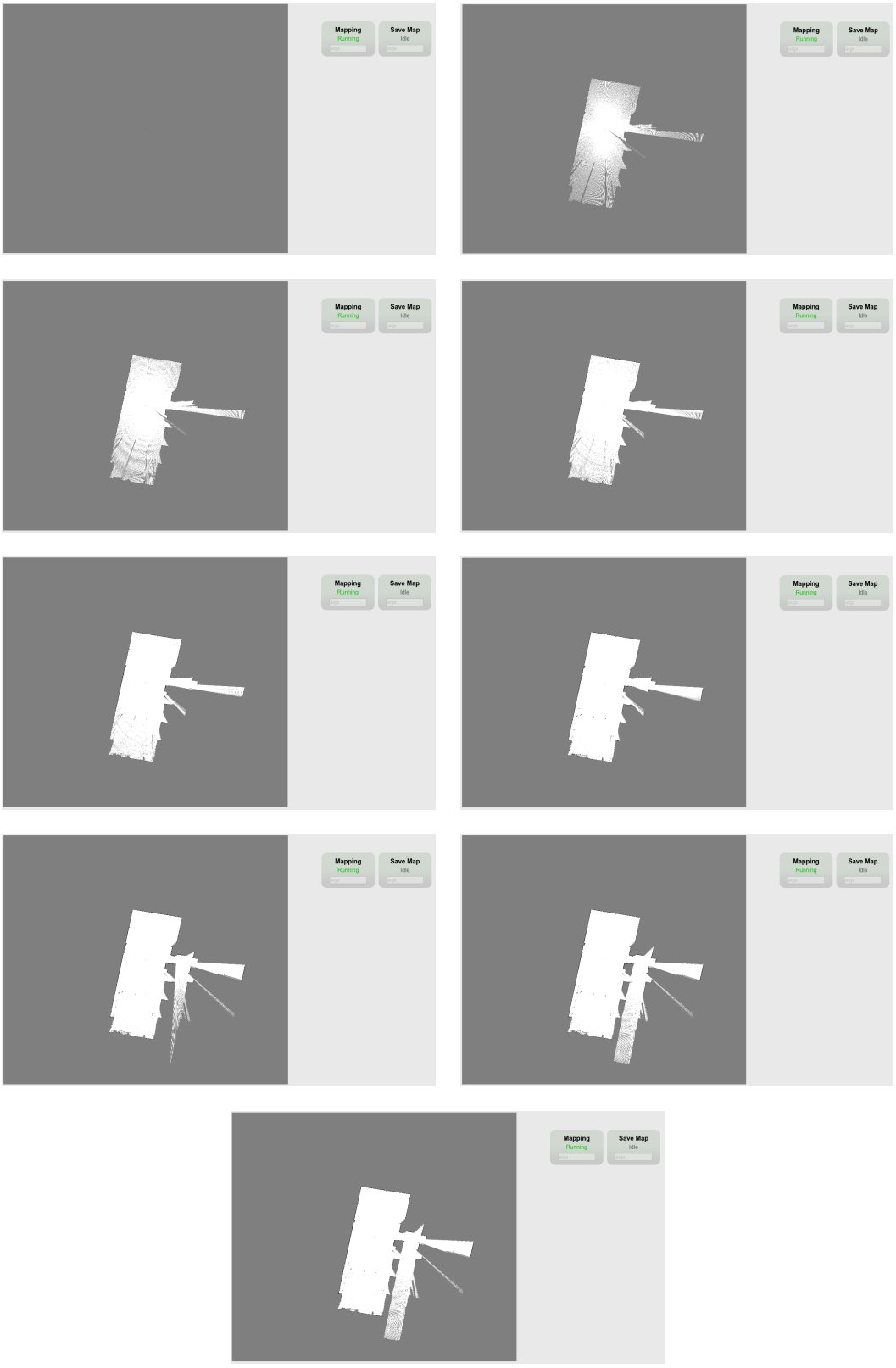

Figure 3.6: Mapping process visualization, from left to right, top to bottom

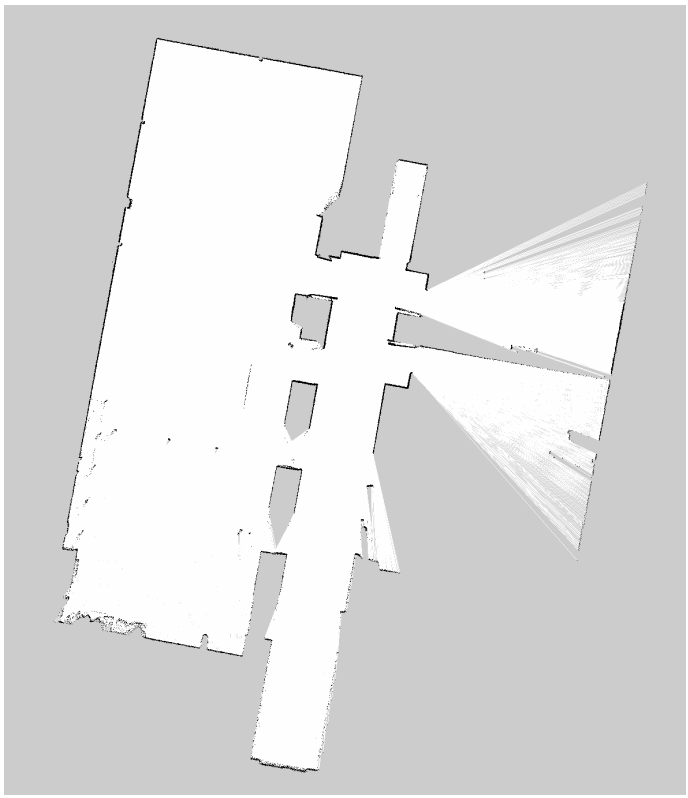

Figure 3.7: Map saved using the "Save Map" button.

page, the user begins by pressing the "Editor" button, which executes a launch file containing the ROS node configuration for using the trajectory editor. If existent, the previously saved trajectory is loaded and the user can edit it by adding, removing or changing the position of waypoints using the menu accessible with a mouse right-click. This system is already fully developed in the robot for use with a native interface, and only adapting it to web interfaces was required. At this point, the user can also load another trajectory using the "Load Traj." button and specifying the file name. The Trajectories page is shown in Fig.3.8. The black arrow represents the robot's position. After building the desired trajectory, the user can set the robot to follow it using the "Start Jarvis" and "Stop Jarvis" buttons. If satisfied with the result, the user can proceed to save the trajectory using the "Save Traj." button, and a custom name can be set for the saved file. The process of building a new trajectory is shown in Fig.3.9.

Finally, the robot's execution process involves self-localization, for which it uses standard ROS algorithms. Viewing the robot's state can be accomplished with either *ros2djs* or *ros3djs*. Currently, *ros3djs* features zooming and panning, which can prove useful when the constructed map increases in size, and thus this library was used. The execution process is visualized in the interface's View page. To use this page, the user has to press the "View" button, which will show the canvas and map of the environment. Then, the user can press either the "Joystick" or "Decision" buttons. By pressing the "Joystick" button, the user can then use a pre-configured joystick to control the robot. Alternatively, by pressing the "Decision" button, the robot's decision

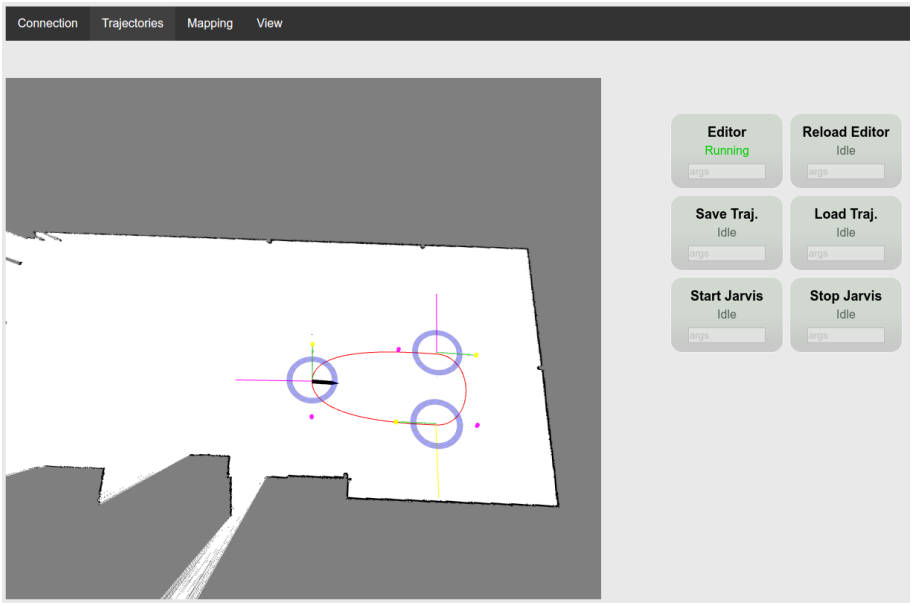

Figure 3.8: The interface's Trajectories page.

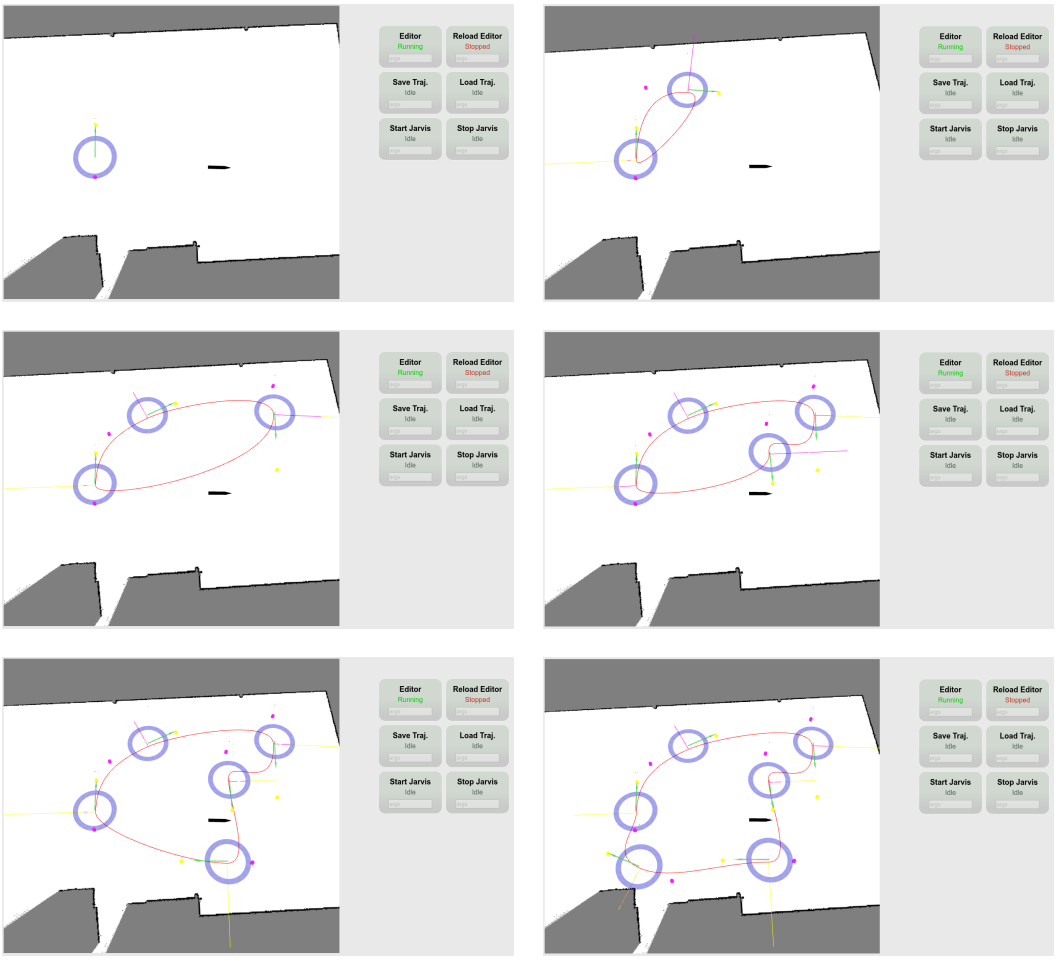

Figure 3.9: The process of building a new trajectory, from top to bottom, left to right.

process is launched and the default trajectory is loaded. As with the Trajectories page, the robot's trajectory is shown, although in this case it is shown in a minimal, non-editable state. The robot's position is also shown using a black arrow. Finally, the user can press the "Start Jarvis" and "Stop Jarvis" buttons to make the robot follow the represented trajectory. Fig.3.10 shows the View page with a loaded trajectory and the robot's position immediately after pressing the "Start Jarvis" button. As such, the robot is not yet following the desired trajectory, though as updates are received concerning the robot's position, it will be updated on the interface.

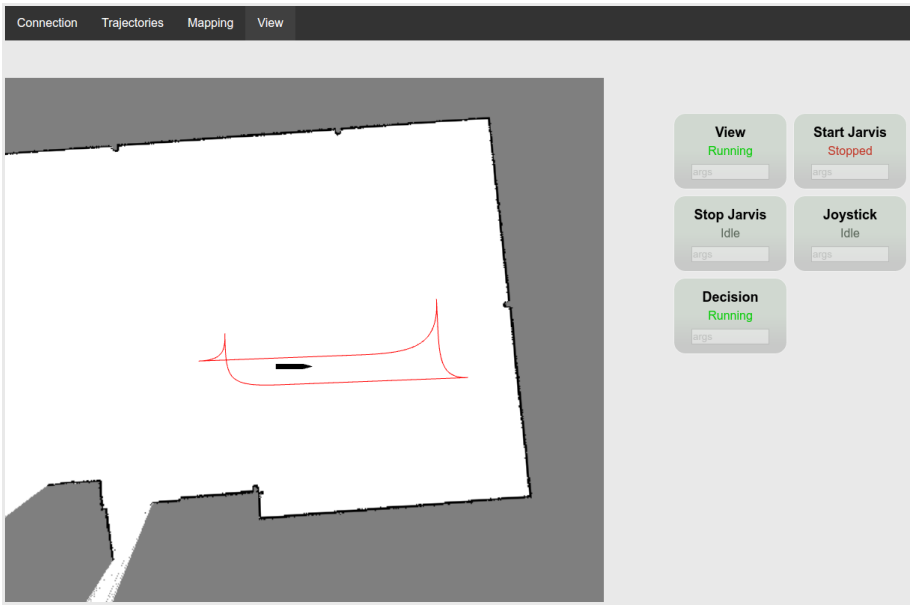

Figure 3.10: The interface's View page.

The developed interface satisfies most of the requirements previously proposed in 3.1. The ease of use is accomplished by using simple buttons to accomplished complex tasks. By putting most of the work in the user interface's development, the user can complete both setup and operation tasks even when not familiarized with the background processes behind the robot's systems.

Table 3.4 summarizes the requirements and their fulfillment. The only requisite unfulfilled by the developed interface is providing alerts concerning the robot's state. This feature should be considered for future work. The situations where an alert message should be shown vary greatly with the application, but some examples are:

- The robot does not know or is not confident in its position relative to the environment. In the case of an industrial AGV, the robot should immediately stop and wait for human action. Here, an alert could be provided to the operator;
- The robot's systems are malfunctioning. It's important that operators know about malfunctions as soon as possible, to prevent stopping the robot's operation for a long time.

ID	Name	Fulfillment
P1	Mapping	Total
S1	Visualize mapping	Total
P2	Trajectory editor	Total
P3	Execution	Total
S2	Visualize execution	Total
S3	State	Partial
P4	Ease of use	Total

Table 3.4: Interface requirements fulfillment. Described in Table 3.1.

Chapter 4

Global positioning using laser rangefinders

After reviewing the technology, localization and mapping algorithms in Chapter 2, this chapter will present a small discussion on how the choice of the positioning sensors impact the overall cost of the AGV.

One of the main reasons for an increased cost in industrial AGVs is the equipment used to determine its location within the environment, that is, its positioning sensors and associated landmarks. As flexibility becomes more and more an issue in adopting AGV solutions, robust and cost-effective sensors need to be used to reduce the overall costs. The laser rangefinder is the most widely used global positioning sensor which allows for flexible trajectory definition.

In order to preserve accuracy and robustness, a laser rangefinder is usually paired together with artificial landmarks which help in self-localization. These are often highly reflective beacons so they can be distinguished from other objects in the environment. The number of artificial landmarks can range from a few to several hundred, increasing setup costs greatly, mostly due to installation and calibration of the landmark's position. The number of necessary landmarks vary with environment conditions, trajectory length, desired accuracy and other variables. Thus, there is an incentive to performing self-localization based on natural landmarks or features.

It can be seen that cost analysis for AGV applications depends greatly on the application itself. Besides the setup, the laser itself can also represent a significant percentage of the whole system's cost. Having more cost optimized laser rangefinders is important in an application where the AGV technology used is a greater percentage of the overall costs. Lower cost sensors will compromise in several ways, such as the following laser characteristics:

- Lack of an inherent localization algorithm;
- Lack of a reflection intensity measurement feature;
- Lower maximum measurement range;
- Higher minimum measurement range;

- Lower accuracy and accuracy decrease with distance to object;
- Lower angular scanning range;
- Lower scanning frequency;
- Lower angular resolution;
- Higher power consumption;
- Lower resistance to vibration and impact;
- Higher measurement noise;
- Higher sensitivity to lighting, temperature and humidity conditions.

However, there are cases, although mostly in research, where more cost optimized laser sensors are used and the application requirements are met by using appropriate localization algorithms and a lower number of artificial landmarks, or even using only natural features of the environment.

The widely used AMCL algorithm, available for ROS, uses a particle filter to track the robot's position relative to the environment. This algorithm does not need artificial landmarks to perform accurately in many environment conditions. It takes the laser measurements, known map and particles and estimates the robot's position while adapting the number of particles in use. In order to be accurate, the algorithm requires a great amount of particles, often hundreds. This causes the algorithm to be particularly computationally heavy. With an increase in complexity and computational weight there is a need to use a faster processor. Besides increasing the cost of the robot, a faster processor usually requires more power, increasing the cost of operating the robot and causing battery charging or changing to occur more frequently. Alternatively, batteries with more capacity or a higher number of batteries can be used, also increasing the cost. Even if the power usage difference in two processors does not seem like it makes an impact in the overall power usage of the robot, it should be noted that a processor is always in use when the robot is operating. In opposite, functions such as moving are not always in use. As such, a more efficient processor is preferred.

An attempt on self-localization that is computationally light weight and does not need artificial landmarks is presented in [38], where position tracking is achieved without modifying the environment with artificial landmarks or beacons. The used algorithm is a fusion of the EKF with the Perfect Match algorithm. This results in a computationally lightweight algorithm that does not need any artificial landmarks. The reason for it being a fast algorithm is the use of precomputed gradient and distance matrices.

The localization algorithm was tested in two applications: a robot designed to use in football robot games, and a surveillance robot. In the first application, a camera with a 360 degree field of view is used to obtain visual information from the environment. The second application makes use of a tilting laser rangefinder to obtain a 3D occupancy grid. Overall, experimental results show distance errors lower than 25 cm and angle errors lower than 9 degrees between the localization

algorithm and the position obtained from a robust Sick NAV350 sensor's inherent localization algorithm as the ground truth. More recent results show errors lower than 10 cm and 4 degrees.

While the results prove the algorithm to be sufficiently accurate for both applications, it is not accurate enough to be used in most industrial applications with AGVs such as docking and load transferring, where usually an accuracy of a few millimeters is required. The resulting accuracy is, though, sufficient for less demanding tasks such as path following between waypoints. As such, a lower number of artificial landmarks can be used in locations where more accuracy is required. In conclusion, the research shows promise and works towards a future where the number of artificial landmarks to be installed is vastly reduced in industrial applications, thus reducing the overall setup costs.

At this point the author would like to propose an application where some of the mentioned laser characteristics can be artificially simulated by using a high performance laser and affecting its measurements with different methods. For instance, measurements could be affected according to a lower cost laser's accuracy characteristics. Another way to impact the measurements would be to throttle the scanning frequency and angular resolution by accepting every other measurement, as an example.

The proposed application could provide some insight on how more cost optimized lasers would perform in an application or when testing different algorithms. If a widely used and fully featured tool was to be developed, for instance to use with ROS, it could provide better comparison results between different research papers, especially research which differs in the self-localization or mapping algorithm used. If laser characteristics could be set as standard for algorithm comparison, the algorithm choice may become an easier task.

It is however impossible to artificially affect all laser characteristics. Non linear characteristics are difficult to reproduce. In spite of this fact, some results could be achieved with a laser comparison application. An interesting result would be a study of localization algorithms robustness to laser choice in different applications and environment conditions.

Chapter 5

Layout simulation with AGV

In big and complex production systems, the number of variables which cause an impact on the system's performance is extensive. Often, the most appropriate method for previewing the resulting performance is using a simulation-based approach. In the case of an AGV system, some examples of variables that can cause an impact on the performance are: the layout, the number and type of AGV, their speed and capacity, the load and unload mechanism, and more. The resulting impact is often too complex to analyze using a non-simulation based approach.

This chapter analyzes the impact of layout choice and AGV characteristics in an industrial system's performance. To accomplish this, a real material handling scenario will be studied and, using real input data, the resulting performance will be analyzed. Using data obtained from the real system is of major importance in simulation, as it greatly reduces the expected simulation error. The simulation error is defined as the performance difference between the system modeled in the simulation application and the resulting performance in the real system.

The underlying scenario is a push-pull system inside a factory, where between a raw materials warehouse and the production center a buffer is located. The transport system is currently accomplished by common forklifts, and it is desirable to study the effect of changing to an AGV-based transport solution. Furthermore, some study is required concerning the buffer size and how reducing it would affect the system's performance.

As will be seen in section 5.2.1, the current layout causes some problems in the system for an AGV solution. An alternate layout will be designed and presented, while briefly discussing mechanical solutions for the storage spaces.

The main tool of analysis in this case study will be computer simulation. To accomplish this, the Simio software will be used. The developed simulation will be used in the future for other material handling related studies.

Firstly, the simulation software to be used in modeling this application is presented in section 5.1.

Then, the case study description, goals, as well as the simulation approach and results are presented in section 5.2.

5.1 Overview of Simio

In this section the reader can find a brief introduction to the Simio simulation software. For a more comprehensive introduction, there are free resource available on the product's website[1].

Simio, an acronym for *Simulation Modeling framework based on Intelligent Objects*, is a simulation software package initially released in 2009. The founder and creator of Simio also led the creation of the simulation products ARENA and SIMAN.

The software is available in a free edition for evaluation and training, as it is limited in the number of models, objects and process steps. Other commercial editions are available which remove these limitations and extend the software's features. The version used for the purposes of this work is 8.136.13569.0.

As with other simulation software applications, Simio can be used to preview a system's behavior when multiple variables are changed, allowing the user to see the impact of business ideas, decisions and rules prior to their implementation. Simio is presented in an interface similar to Microsoft Office products, with ribbon-style tabs system.

Simio models are usually built using object oriented programming. Typically, simulation models are built using a modeling paradigm known as process orientation. It is the case of the widely used ARENA software and other products. In process oriented models, the model is built by defining elements which define the state of the system, and then building process flow blocks that take actions on these elements. Usually, the process flow can be described with flowcharts. In the case of Simio, however, an object oriented approach is supported. Objects are built with logic and can interact with other objects, even of different types. They are then placed in a model and the actions are defined completely by the objects' internal logic. Objects are characterized by their properties and states. Properties are static; their initial value can be changed prior to running the simulation and does not change during execution. On the other hand, states specify dynamic values, which change while the model is running and are also defined by an initial value. Overall, object oriented programming is a different approach to simulation, not well suited for every application, but can prove to be useful in building elegantly designed models with hierarchically organized objects.

The facility window, where objects are placed to interact with each other, is presented in either a 2D or 3D graphics environment, which can be changed on the fly. An example of a model viewed from a 2D and 3D perspective can be seen in Fig.5.1 The collection of objects and how they are placed form the facility model. Besides having internal logic, an object is also defined by its graphical symbol and, if desired, animation. An object's animation is defined by several symbols and a variable whose state value defines the current symbol being displayed. Simio has an extensive symbol library for graphical visualization, but also accepts typical 3D modeling objects and has Google 3D Warehouse integrated directly in the software, providing thousands of symbols to be used. After placing an object with a drag and drop method, the user can then modify an object's properties which will define the interaction with other objects. It should be said that

[1] http://www.simio.com/index.php

the facility view does not exist purely for viewing purposes. In fact, the distance and organization of objects within the modeling environment will also dictate the objects' interactions.

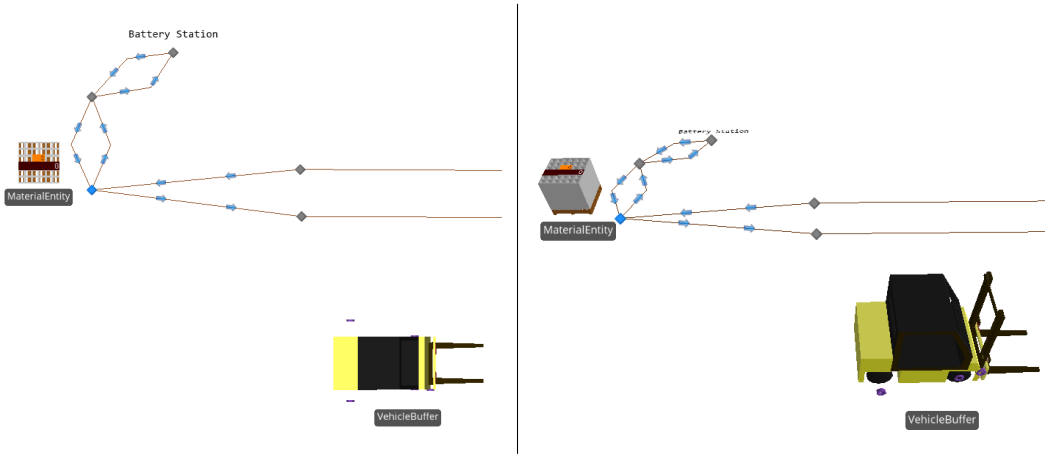

Figure 5.1: A Simio model. Left: 2D perspective. Right: 3D perspective.

Objects that are placed in the facility model are user-defined and stored in libraries that can be interchanged with other models. These objects are defined by other objects and their interaction, forming a small model that can be placed in the facility view. The limitations in modeling are then imposed by the most basic objects which come with the software, also called object classes. The following object classes are defined:

- Fixed objects - they have a fixed location in the model and represent machines, work stations, anything with inputs, outputs and some kind of processing that takes actions on objects that visit it;
- Entities - these objects are dynamically created and destroyed during the simulation run and move through the simulation 3D space. They usually represent people, materials and similar objects;
- Transporters - they are a special type of entity that can be used to carry other entities between locations. AGV are modeled using this object class;
- Links - objects used to move entities over a pathway, either a basic entity or a transporter carrying another entity. AGV paths are modeled using links;
- Nodes - they define the starting and ending point of a link, and are used to transfer to and from objects of the fixed class.

It should be noted that an object definition builds on its class and adds internal logic to customize its behavior. An object is built just like the main model, and can be used in other models as an object definition. After objects are defined, they can be placed in the facility model. Simio includes two object libraries, the standard library and the flow library. The flow library contains objects to model flow processing systems. The standard library contains the basic Simio objects.

For many applications, the standard library of Simio objects can be sufficient. Table 5.1 summarizes the function of each object in the standard library, while Fig.5.2 contains the graphical representation of these objects in Simio. There are no objects of the entity class besides the entity object itself which is part of Simio and not present in this library.

Name	Class	Function
Source	Fixed	Create entities to arrive in the system
Sink	Fixed	Destroy entities
Server	Fixed	Process entities, with queues and processing time logic
Workstation	Fixed	Process entities, with added logic for setup and tear-down activities
Combiner	Fixed	Batch entities together and match with a different type of entity
Separator	Fixed	Separate batched entities or make copies of an entity
Resource	Fixed	To be seized and released by other objects as a resource in the system
Vehicle	Transporter	Transport entities between nodes
Worker	Transporter	A mobile resource which can also transport entities
BasicNode	Node	Transfer entities to and from other objects, and provide link intersections
TransferNode	Node	Route an entity to a destination and reserve transports
Connector	Link	Directly connect two nodes with zero travel time for entities
Path	Link	Model a pathway with speed limit, passing rules and travel time defined by its length
TimePath	Link	Same as a Connector, but with user-specified travel time
Conveyor	Link	Model the properties of conveyors where entities can accumulate on its end

Table 5.1: The standard library's objects in Simio

To create new objects, the user can choose between creating objects by using existing object definitions, sub-classing an existing object definition or creating a base object which is defined using a set of processes. Processes define Simio's process oriented programming capabilities. They encompass the most basic tasks Simio can perform, and even user-defined processes can be used, but these must be created outside Simio with other, .NET programming languages. Outside programming is also used to build custom data integration for importing or exporting data automatically, or creating complex algorithms and rules. Processes are comprised of steps, elements and tokens. A token is the traveling entity between steps, which perform a certain action. Elements are the object's basic components that perform tasks such as generating queues, failures and statistics, as well as data importing and exporting operations.

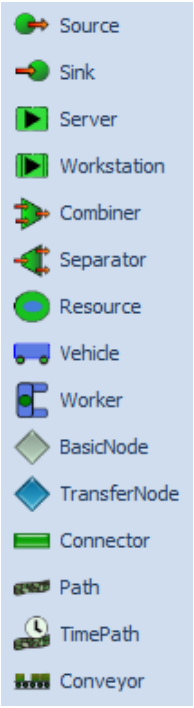

Figure 5.2: Simio's standard library objects and default symbols.

The role of processes is to extend object definition capabilities. In fact, all objects in the standard library are built using Simio's available elements and processes, and their behavior processes can be viewed by sub-classing the object, facilitating debugging tasks. Process steps can range from decision-making and locating a specific object in the model, to resource seizing and transferring entities between objects. As an example, consider the process in Fig.5.3. This process is used to transfer the first item of a station's ranking queue to its output buffer. In case of transfer failure, the process token is assigned with an error value. Here, the process token is the default Simio token, as no state variables are required within the process scope. Other processes may use different, more advanced tokens with multiple state variables.

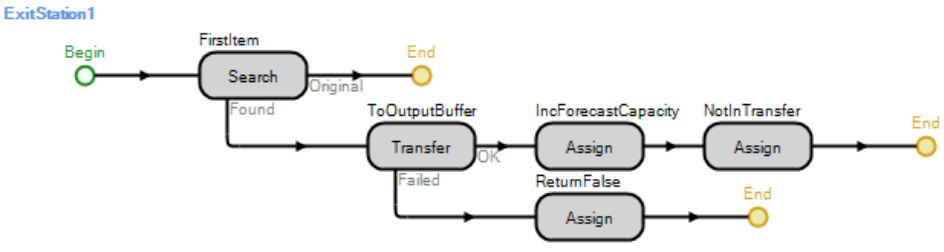

Figure 5.3: An example of a Simio process.

For data integration, Simio can inherently import and export to Microsoft Excel spreadsheets. As previously mentioned, the user can build custom data integration using .NET programming languages.

It is important to mention one other feature of Simio, which is randomness. Using random numbers is important in a study with simulation, allowing not only to approximate real life variable behavior, but also to see the impact of the general variance of a variable. It is possible to define non-random and random variables individually. The distributions used are varied, both discrete and continuous distributions, and the parameters are configurable. Examples of random distributions are the normal, bell, poisson, uniform and triangular distribution functions. In Simio, the user can run the model with a replication number which all random variables will look at to know their next value. This, in turn, means that the user can re-run the model with the same characteristics for debugging, or run the model several times with different replication numbers.

The previous feature applies to Simio's experiments, which are comprised of scenarios, that is, runs of the model without any animation to increase simulation speed. When running an experiment, the user can specify how many replications should be ran, for which the results will not only display the average of all replications, but also measurements of confidence intervals for a given confidence level. Also, when running multiple scenarios in an experiment, the same replication numbers will be used in each scenario for consistency. The experiment's results - known as responses in Simio - can be presented in a multitude of ways, such as charts with statistical data, histograms, pivot grids, and extensive reports. The response sensitivity and sample size error of certain input variables can also be analyzed. On the other hand, when running a specific scenario in the facility view, feedback is given to the user in the form of dynamic graphs and dynamic labels with state values. An example of a dynamic graph can be seen in Fig.5.4. Interactive controls are also available which can be useful for debugging the model. At the end of the run, output data can be organized in a dashboard-based report.

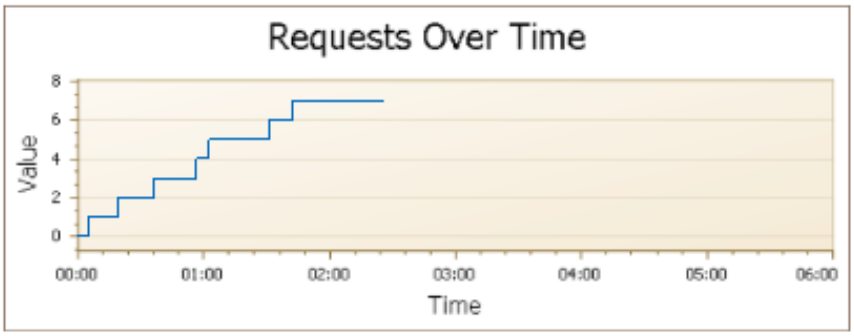

Figure 5.4: An example of a dynamic graph in Simio.

The overview in this section should provide enough background information to understand the methodologies which will be mentioned in later sections. Furthermore, the author refers to the resources located on the product's website and all the support tools in the free version of the product, such as the reference guide, a vast collection of examples, e-books and introductory videos.

5.2 Case study

5.2.1 System description

The target application of this work is based in a real scenario where raw materials are moved, in pallets and by forklifts, to a buffer zone, in a push-type system. On the other hand, the production side gets the materials necessary from this buffer zone, in a pull-type system. This is described in Fig.5.5 It is of great importance that material requests by the production side are fulfilled as quickly as possible, as delaying this delivery would potentially stop production temporarily.

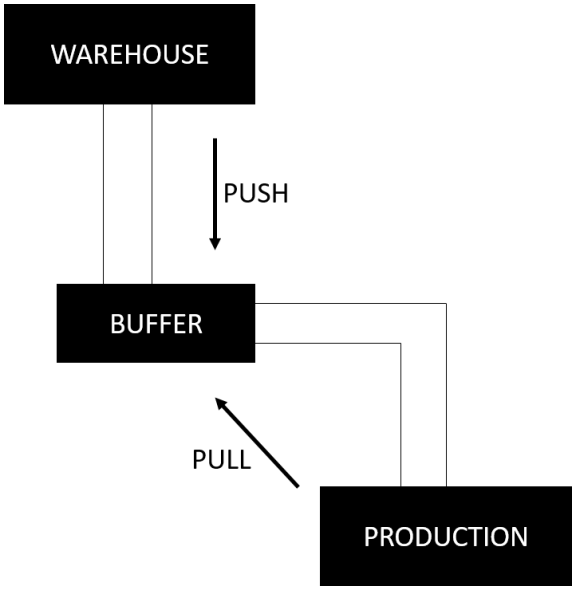

Figure 5.5: Description of the case study underlying system.

The material types are not of interest in this study, though, instead the focus is on how the pallets are transported, and how the buffer usable area is being utilized. At the moment, 39 different material types populate the system, some being used much more often than others. The material utilization varies monthly, and changes can occur both in the warehouse and buffer layouts and material distributions. Currently, the material handling task is accomplished by common forklifts, using pallets standardized in size.

Materials are requested by the production side to fulfill the production needs. Requests refer to a particular material type and weight. Two forklift vehicles owned by the production side, from here on called **production vehicles**, pull materials from the buffer in a LIFO order system. It is desirable to change to a FIFO order system. The overall system does not stop at any time, and requests may be generated at any time and with any material weight.

On the other hand, the warehouse side is tasked with replenishing the buffer zone with materials. As requests are generated, the system keeps track of the buffer capacities and orders to replenish are generated accordingly. The material handling tasks are also handled by forklifts, from here on called **warehouse vehicles**. Currently, a single forklift is tasked with all the material handling tasks.

There is enough space in the transporter zone lanes for two vehicles to travel on it in both directions without any traffic being blocked.

The buffer zone is characterized by a free space system with pallets. Most materials can be stacked to a maximum of two pallets, though some can't be stacked. In the middle of the area is a shared zone for transporters, both production are warehouse vehicles. A specific material is designated to any storage space. There are also specific storage spaces for compounds and empty pallets, although the desire is to not use these in the future.

5.2.2 Goals

Given the system's description in the previous section, it is desired to study the possibility of changing the warehouse transportation system to an AGV based system, as well as the effect of using more vehicles than currently. The results should thus include a comparison of using common forklifts or AGV and a varied number of vehicles. On the other hand, the production vehicle system will not be analyzed as it is not the scope of this project.

For the reasons described in the next section, a new buffer layout will be developed and the results should include a comparison based on the performance using either system layout. The layout's utilization, that is, the storage space maximum utilization, will be used as an input to the system, as it is desirable to decrease the overall utilization, both to keep stocking costs down and to allow the resulting available space to be used for new materials, empty pallets that require unplanned storage space, or even other installing other equipment. Essentially, having free space in the layout benefits the overall system's performance, although a quantitative index isn't used in this case.

The system performance is described mainly by the time a request order from the production takes to be fulfilled. Examples of relevant measurements are the average time to fulfill all requests and the maximum time it took to complete any request order. Other descriptions of the system's performance is the vehicle utilization, that is, the time warehouse vehicles are tasked with a material handling task.

5.2.3 Buffer layout - current and new proposal

As mentioned in section 5.2.1, the system's process is a push-pull type process of material handling between a warehouse, a buffer and the production zone. A simplified representation of the current system's layout is presented in Fig.5.6, as well as the material distribution and available pallet capacity in each storage location. The real layout can be seen in Fig.5.7, where the buffer, warehouse and production zones are marked, and the assigned vehicle lanes are marked in gray but are not to scale.

The issues verified in the current system layout are the following:

- The layout is not uniform, that is, the different storage spaces hold different amounts of pallets and their location is irregular, because of doors, passageways and others;

- The shared transporter zone between the production and warehouse vehicles (grey area in Fig.5.6) is undesirable, as AGV must have a lower operating speed in a shared zone;
- There is a great amount of unused volume space as some materials can't be stacked.

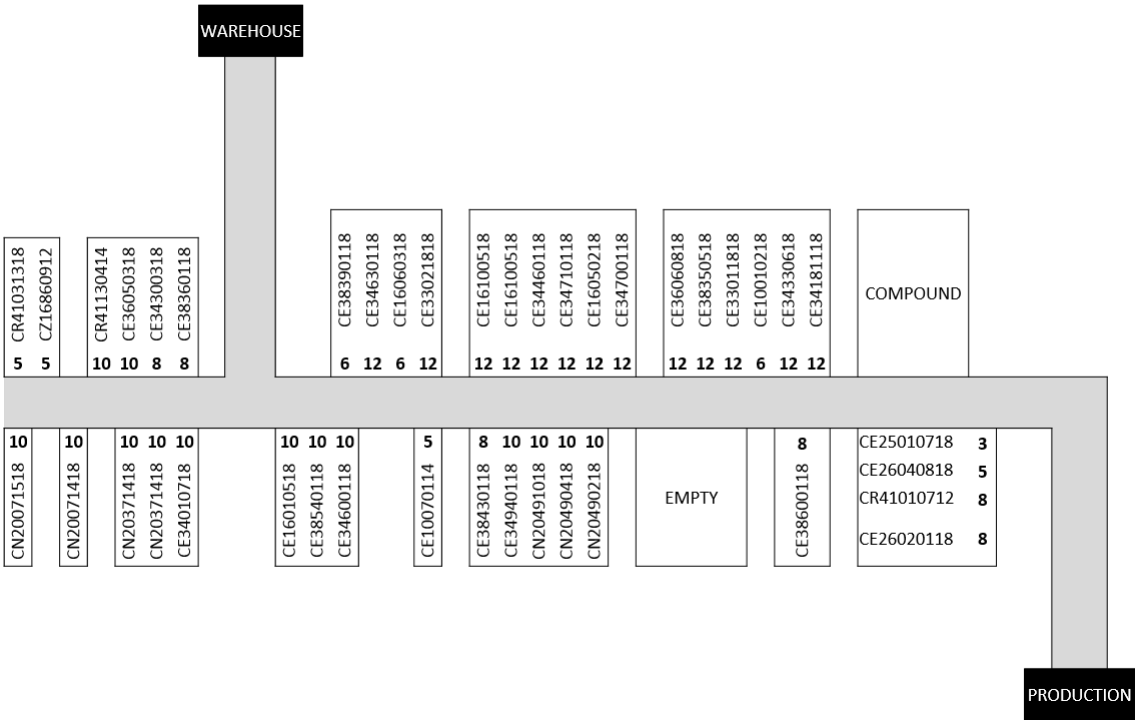

Figure 5.6: Original layout, simplified and showing the material distribution.

The referred issues will make the adoption of an AGV solution harder. The irregular space will increase traffic blocking time, and the shared transporter zone, where AGV would share space with forklifts, will mean the maximum speed of the AGV will have to be reduced. Finally, the unused volume is an obvious problem as it is desired to increase the usable volume in the buffer, allowing for more capacity.

In an attempt to solve or at least diminish the effect of the mentioned issues, a new layout is proposed and presented in Fig.5.8. The simplified version of this layout with material distribution can be found in Fig.5.9. The storage locations available are 34, each with 2 levels for pallets. Table 5.2 specifies how many pallets can be stored in each storage location in the new layout. Storage locations can be assigned either one or two material types, as depicted in Fig.5.10.

Storage Type	**Capacity Available (pallets)**
One material	16
Two materials	8 (each)

Table 5.2: Storage location capacity in the new layout.

Figure 5.7: Original buffer layout. The assigned vehicle lanes are marked in gray.

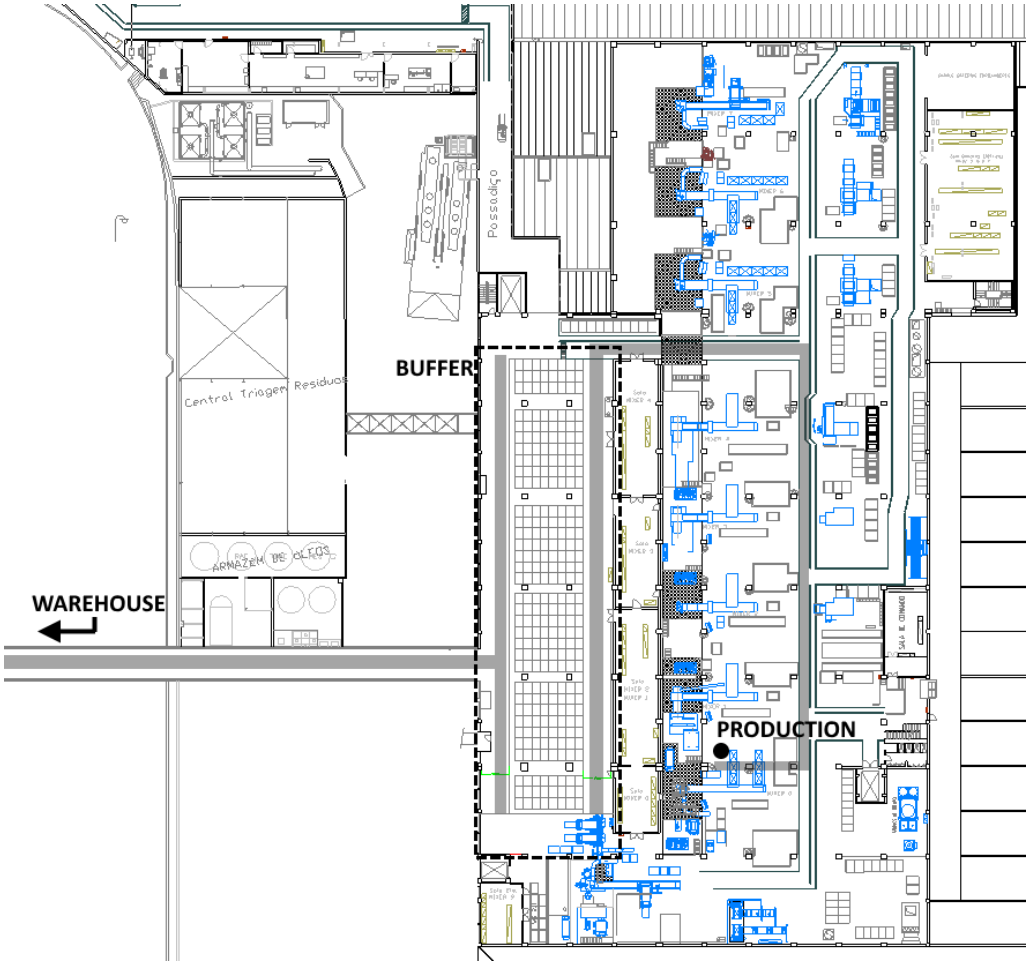

Figure 5.8: Proposed buffer layout. The assigned vehicle lanes are marked in gray.

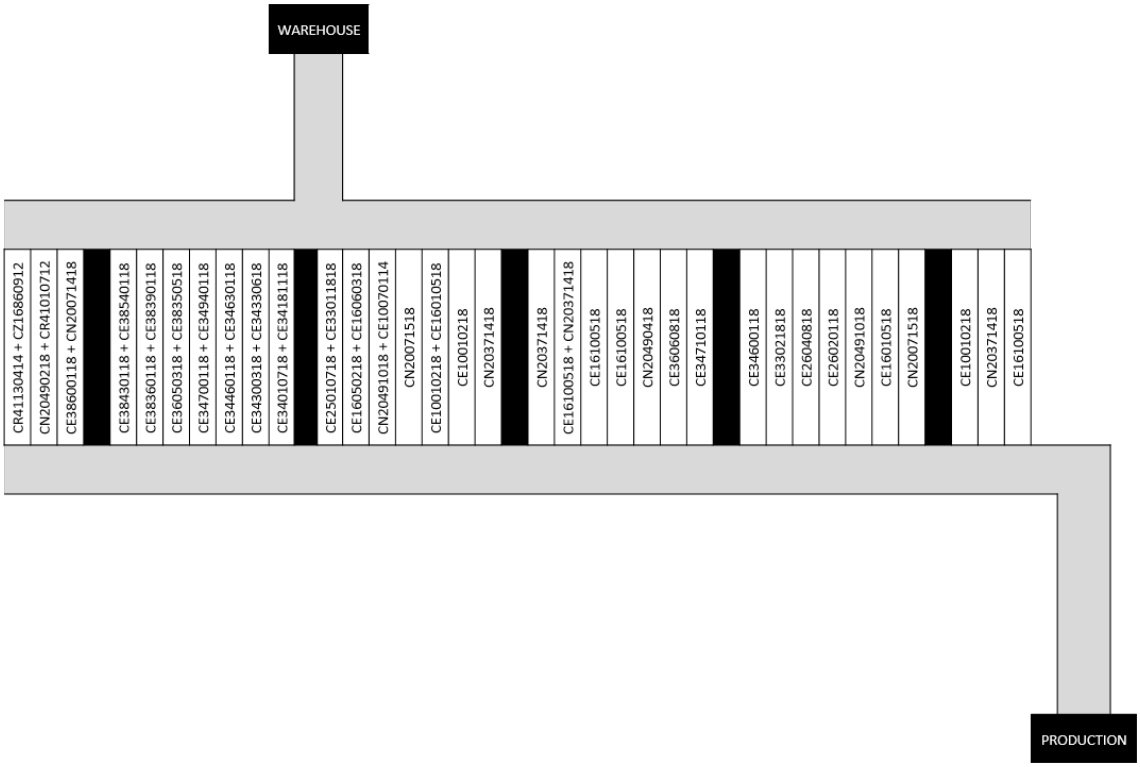

Figure 5.9: Proposed layout, simplified and showing the material distribution.

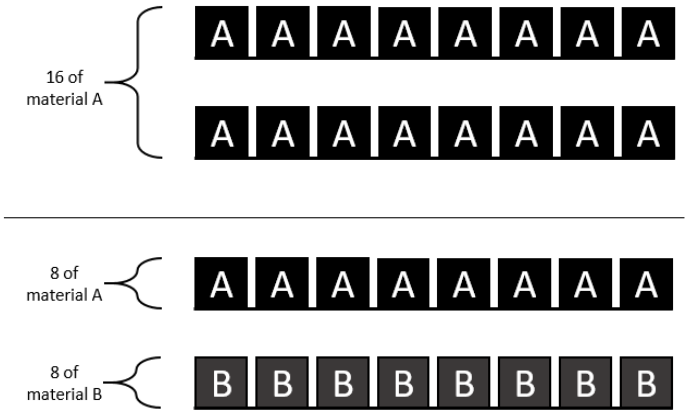

Figure 5.10: Top: Storage location with one material type assigned. Bottom: Storage location with two material types assigned.

The total storage capacity, in pallets, is calculated as follows:

$$capacity_{total} = capacity_{storage} * levels * locations = 544 \tag{5.1}$$

The material distribution is a result of a study based on the material requests that occurred in the course of a month. The percentage of requests was calculated for each material, which is proportional to the calculated capacity. This capacity was manually adjusted to conform to the restrictions of a minimum capacity of 8 per material, as only one material should be assigned to each storage location level. Also, capacities must be a multiple of 8 as that is the storage capacity per storage location. The results are presented in Table 5.3.

The approach taken in this study takes into consideration the requests of one month, thus being only temporarily valid. Also, this approach does not reflect the burst effect of requests. That is, even for a material that does not have many requests over a month, those requests may happen non-uniformly, for instance, during a single hour, and the storage capacity available for this material may prove to be inadequate. That said, this is the exact purpose of having a buffer, so this effect should be minimized as long as new material quantities are provided by the warehouse immediately. To conclude, it should be said that this study should not be considered sufficient for a change in the layout, and a more in depth study is required. For the purposes of this project, this study was deemed adequate.

The proposed layout encompasses two non-shared transporter zones, effectively separating the warehouse vehicles from the production vehicles entirely. This will facilitate the introduction of AGV vehicles on the warehouse side. Furthermore, the uniformity achieved with the storage location disposition will also provide benefits when setting paths for an AGV. Finally, the introduction of a fixed two-level system for each storage location, the effective volume of the buffer increases.

The two-level system can't be achieved by keeping the free space stacking method currently in use, and a different mechanical solution should be adopted. The solution should provide a FIFO order system, as it is desirable that the older materials are consumed first. The proposed solution is a pallet flow racking system, also known as gravity flow. This solution is AGV-friendly, as it does not require the AGV to leave the main transporter zone and go inside the storage location. An alternate solution would be a drive-through racking system. Although much cheaper and requiring less maintenance, as it does not require movable parts like the pallet flow system, using a drive-through system would imply a change of the material distribution for the storage locations. With a two-level system, unused materials in the lower level can block the vehicle's path, and the top-row materials may become unavailable from the other side. This is only a problem when that storage location is assigned to two different material types. Fig.5.11 provides an example of this situation, where the production side is unable to retrieve material B until material A is consumed. As seen from this figure, the pallet-flow system circumvents this problem. During the model development it is assumed that the pallet flow system is adopted.

In order to find if the new layout can satisfy the system's performance needs, simulation models for the old and new layouts will be developed. In the next section, the simulation modeling

Material Type	**Percentage of Requests (%)**	**Calculated Capacity**	**Corrected Capacity**	**Storage Locations**
CE16100518	12.58%	68.44	56	7
CN20371418	12.25%	66.65	56	7
CE10010218	10.08%	54.82	40	5
CN20071518	6.49%	35.29	32	4
CE16010518	5.86%	31.89	24	3
CN20491018	5.66%	30.81	24	3
CE26020118	3.49%	18.99	16	2
CE26040818	4.02%	21.86	16	2
CE33021818	4.15%	22.57	16	2
CE34600118	3.42%	18.63	16	2
CE34710118	4.03%	21.95	16	2
CE36060818	4.33%	23.56	16	2
CN20490418	3.69%	20.07	16	2
CE10070114	0.12%	0.63	8	1
CE16050218	0.68%	3.67	8	1
CE16060318	0.68%	3.67	8	1
CE25010718	0.56%	3.05	8	1
CE33011818	1.35%	7.35	8	1
CE34010718	0.13%	0.72	8	1
CE34181118	1.05%	5.73	8	1
CE34300318	0.99%	5.37	8	1
CE34330618	1.71%	9.32	8	1
CE34460118	1.91%	10.39	8	1
CE34630118	0.12%	0.63	8	1
CE34700118	0.07%	0.36	8	1
CE34940118	1.70%	9.23	8	1
CE36050318	0.40%	2.15	8	1
CE38350518	0.46%	2.51	8	1
CE38360118	0.16%	0.90	8	1
CE38390118	0.23%	1.25	8	1
CE38430118	0.48%	2.60	8	1
CE38540118	0.99%	5.37	8	1
CE38600118	0.26%	1.43	8	1
CN20071418	2.11%	11.47	8	1
CN20490218	0.43%	2.33	8	1
CR41010712	1.88%	10.21	8	1
CR41130414	1.48%	8.06	8	1
CZ16860912	0.02%	0.09	8	1
Sum	100.00%	544.00	544	68

Table 5.3: Study results for material distribution in the new layout.

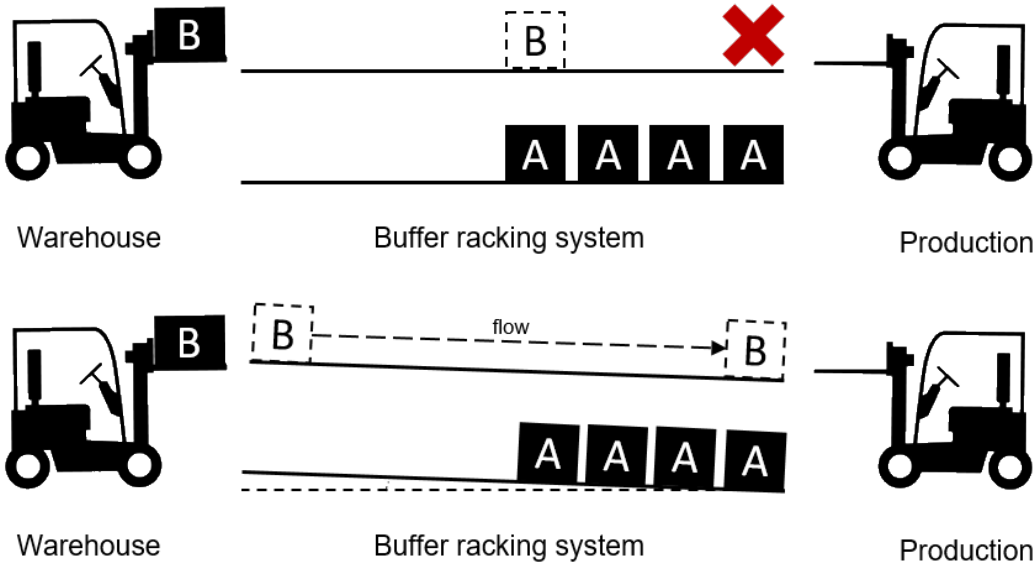

Figure 5.11: The problem of using a drive-through racking system when a storage location is assigned to two different material types. Top: drive-through system. Bottom: pallet-flow system.

approach, as well as the application's inputs and outputs, are presented.

5.2.4 Simulation approach

The model has been developed and tested using Simio version 8.136.13569.0. Although future versions will usually support the developed model, older version may not have all available features and thus may not be able to run the model. The reader should refer to section 5.1 for some background concepts which will be discussed here.

Due to the project's scope of analyzing the buffer system's performance, many simplifications are made in order to streamline the model's development. These simplifications will be discussed during this section when relevant.

The most significant simplification in the developed model is assuming all materials in the warehouse are in a single location and all the production destinations are also in one location. This simplifies the model development, although an extension could be made to include a realistic model of the warehouse and production's systems. In order to not increase the simulation error, the time to grab a material in the warehouse is assumed constant during the model run, but varies with the type of vehicle. A number in the upper bound of the time to grab any material is used. The time to deliver a material to the production side is also assumed constant. In order to not have this simplification, the model would have to extend to the real warehouse and production systems, an effort that will be the next step of the project if required, but outside the scope of this work.

5.2.4.1 Model entities

The main entity in the model is a material pallet, and the model's base material measurement is one pallet. This approach is used to prevent dealing with material weight while running the model. As a consequence, it may require additional pre-computation to the input parameters before running the model, if the arriving request refers to material weight instead of pallets. A **material entity** is created in the warehouse, from where a warehouse vehicle should carry the entity to a storage location reserved for that entity's material type. The destination storage is immediately decided when the entity is created, since it is the storage space itself that generates the creation of the entity, which is possible due to Simio's object oriented modeling approach. This process is described in more detail in section 5.2.4.2. After transporting to the storage space and being stored, when a request order is generated for the same material type, the entity (or any other of the same type) is carried to the production, where it is batched together with the request and, when the request is fulfilled, all entities involved are destroyed.

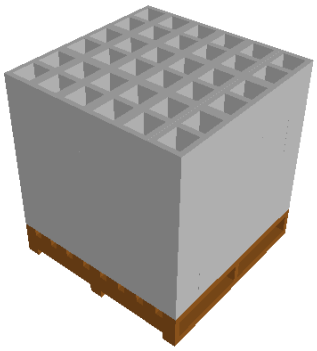

Figure 5.12: The material entity from a 3D perspective.

The other entity which populates the model is a request. A **request entity** is created according to the input request table, which consists of a set of rows, each specifying which material type the request is for, the necessary quantity, and the date it arrives. This input is taken from the real system and used to generate the simulation results. Since the real system only provides the day a request arrived, and the request can arrive at any time during the day, the arrival time is generated as any number from a uniform distribution between 0 and 24 hours. It should be noted that not only the hour of arrival is randomly generated, but also the minute and second. This approach, paired with the randomness factor available in Simio's simulation runs, should provide reasonably accurate results over many replications. When the request entity is created, it goes to a combiner object where it is then processed. It should be noted that the combiner is fictitious and only exists in the simulation model to simulate the request being fulfilled.

Processing a request consists of searching all storage locations for material availability of that request's material type. The search considers a ranking queue of materials, ordered by an entity's time in system, that is, the time the entity has spent in the model since its creation. The material with the higher time in system is chosen to exit its storage location and a production vehicle is tasked with carrying it to the combiner. This process is repeated concurrently until the request

quantity is fulfilled. When there aren't any available materials in stock, the request waits for a new material arrival, as the re-stocking task is done by the warehouse side.

A priority system is defined for requests, so that higher priorities can be assigned to requests requiring large amounts of materials instead of the ones already or close to being fulfilled. Thus, the warehouse vehicles will be aware of request priorities when choosing which material pallets to take to the buffer first. Simio's priority state values are required to be integers. Requests of the same material type all have the same priority, defined by the maximum priority of all requests of that material type. All priorities are updated whenever a new request arrives or is destroyed. A request's priority, r_p, of a material type m was defined as in equation 5.2, after which all requests of the same type take the maximum value of all those requests' priorities.

$$r_p = round(\alpha.r_{TIS} + \beta.Rm_{remaining} - \gamma.m_{available}) \tag{5.2}$$

Where $round(x)$ rounds x to the closest integer, r_{TIS} is the request's time in system (in hours), $Rm_{remaining}$ are the remaining quantity to be fulfilled of all requests of type m, $m_{available}$ are the materials in stock (in the buffer) of material type m, and α, β, γ are constants.

By using positive α, β, γ constants, the request's priority will increase with its time in system and unfulfilled orders of its material type, while also decreasing with material availability. When the time in system is not considered, a case where material availability is greater than the material needs, the resulting priority is lower than zero. On the other hand, if the request needs materials not currently in stock, the resulting priority is greater than zero. Thus, by considering higher priority requests firstly, and re-stocking materials according to those priorities, the system's response time should decrease, increasing overall performance, specifically the maximum fulfillment time. The request's time in system is considered when computing priority in order to prioritize those with greater time in system when multiple requests of the same material type exist.

Fig.5.13 shows the request processing queue with 4 different request entities. For animation purposes, labels are used to show the material type, time in system, priority and order amount, respectively from to to bottom.

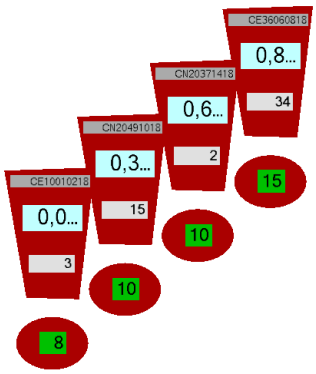

Figure 5.13: The processing queue for requests, with dynamic labels.

5.2.4.2 Storage locations

In order to model storage locations, usually the *server* object from the standard is library is used. This is the case of the simple "WarehouseExample" model available in Simio. The material stays in the storage location for a certain amount of time - which is the processing time in the server's properties. However, this kind of approach is adequate only to pure push type systems, in which the entities - in this case, the material pallets -, would be sent to the production without request orders happening. The reason for this difference is that the server object does not accept an infinite processing time and a triggering event to release the object. In this case, it is required that the material entity stay in the storage object until a request order requires that specific material.

The storage location modeling was accomplished with a new object definition, called **storage**. A storage object consists of four main elements, which are all stations. Stations are used to hold discrete items and define an internal capacity. The four stations consist of two storing stations, an input buffer serving as a **deciding station**, and an **output buffer**. To access the object, two external nodes are defined in the model, for input and output of entities to the object. Material entities (that is, the material pallets) are dropped by vehicles at the input node and transfered to the deciding station, after which the vehicle has completed its task. When entities are set to exit the object by a request order, they are firstly transfered to the output buffer where they wait for a production vehicle to arrive. In the deciding station, the entity is analyzed to decide which storage station to move it to. Although there are two storage station elements, **station 1** and **station2**, the object can be setup to have one or two material types. When one material type is selected, only station 1 is used. On the other hand, if two material types are selected, both station 1 and station 2 are used, but each have half their original capacity. The storage capacity is a property of the object. Other properties include the initial WIP for each station, that is, the initial stock when the model run begins, and the **limit utilization** property.

The limit utilization property is defined for each storage object individually, although a general model property is used in this work, corresponding to the buffer utilization which will be one of the variable inputs for which the performance impact is to be analyzed. When the property is set to 100%, the object will use all its capacity to store materials. However, if a lower number is used, the object will have a lower effective capacity during the model run. During the course of this project, the initial WIP was set to the expected steady state material stock, according to effective capacity. For instance, in a storage location with a capacity of 8 and limit utilization of 75%, the effective capacity is 6, as is the initial WIP.

The effective capacity is used together with a monitor element for each storage station. The monitor triggers a process whenever the station's capacity goes above the target effective capacity. This process creates new material entities in the warehouse system in order to re-stock the station, assigning the entity's destination to the storage itself. Also, the entity's priority is assigned so that warehouse vehicles can rank material entities in a queue by their priority. The priority assigned is the maximum priority of any request with that material type currently processing. It is important to recall that the request priority is set according to equation 5.2. Thus, with these two priority sys-

tems, the warehouse vehicles will effectively prioritize urgent request orders with larger amounts of materials.

It is important to mention that the process of creating new material entities to re-stock a storage location is not a request, as the production order. Instead, it is a modeling approach that does not reflect on the real system. As previously mentioned, the warehouse-buffer system is a push-type system, that is, the warehouse replenishes the buffer when necessary. In the model, however, it is the storage object that creates a material in the warehouse and sets its destination. After the entity creation, the entity is carried from the warehouse to the storage object in the buffer, by a warehouse vehicle. The result is, thus, the same as if it was the warehouse creating the entity, but due to the object oriented modeling approach in Simio, it is simpler to use this method.

Fig.5.14 shows the external definition of the storage object, from a 2D perspective. Some dynamic labels are used to give feedback to the user. On the left, the bottom station stock and assigned material type, and on the right, the top station stock and assigned material type. The nodes to the left and right of the storage object are the input and output external nodes, respectively, for transferring material pallets in and out. Fig.5.15 shows the storage object from a 3D perspective, both empty and socked with material entities.

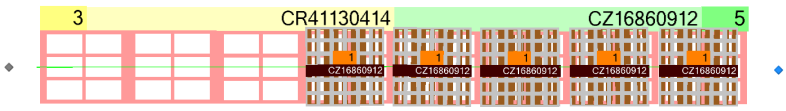

Figure 5.14: The implemented storage location object, from a 2D perspective

5.2.4.3 Vehicles and paths

Vehicles in Simio are standard objects which can accomplish mostly all of the tasks required for this project. Capacity, load time and unload time properties can be set. All properties - and this extends to all objects - can have an expression value, which is evaluated every time the property is referenced, instead of a constant value, giving properties more flexibility. The default Simio's vehicle can be seen in Fig.5.16 carrying two material entities.

A vehicle can be set to travel either in **free space** or in **networks**. Networks consist of a set of **paths**, and are important for traveling network modeling. Traveling in free space is more appropriate for workers and, in general, people. When it comes to vehicles, especially AGV, the path traveling mode should be used, so that specific trajectories are followed. The downside to this traveling mode is that many nodes and paths have to defined. In this model, hundreds of paths and nodes were used, increasing the total model running time as the complexity of the model also increases. Another disadvantage of the path traveling is that acceleration is not used to model the vehicle's speed, although Simio features acceleration in free space traveling. This results in further simplification of the model. The adopted solution was to use a lower vehicle speed to obtain the results.

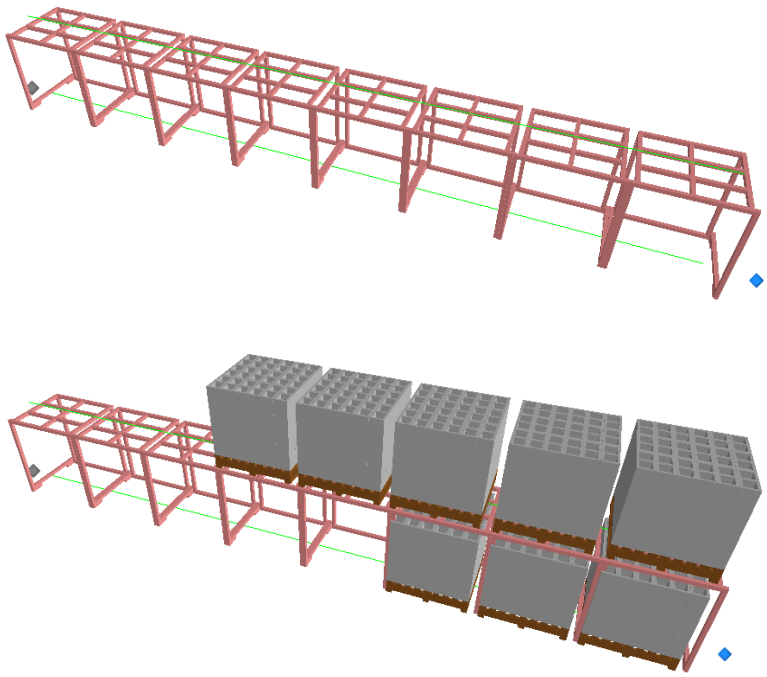

Figure 5.15: The implemented storage location object, from a 3D perspective. Top: empty storage location. Bottom: storage location with 2 material types and different stock amounts.

Paths can be either unidirectional or bidirectional. In both the defined layouts, there are always two vehicle lanes available, one for crossing in each direction. As such, unidirectional paths were used. It was considered that all vehicles, including AGV, can cross between lanes at defined crossing points along the lane. Thus, a vehicle does not have to go to the end of the buffer space to begin traveling in the opposite direction.

Simio's vehicles have resource logic capabilities, that is, a work schedule can be defined for busy and working states. A schedule was defined for operators according to the real system. On the other hand, AGV resource logic isn't usually defined by a work schedule, but by battery state. In the model, the vehicle can be defined as an AGV, in which case it will go to a battery station located in the warehouse. Two different solutions to battery management are considered: automatic replacement and charging. Both these cases can be modeled by the time it takes for this process to complete. A battery charging/replacement station is defined in the warehouse. A simplification of a potentially real system was made here, as it is assumed the time it takes until the AGV needs to go to the battery station is constant. A more accurate solution would be to define a battery charge level which decreases at different rates according to the AGV current task. When it is not moving or loading/unloading materials, the battery level would decrease at a slower rate. This implementation is proposed as future work to further decrease the model's simulation error. It is possible to develop such an implementation in Simio.

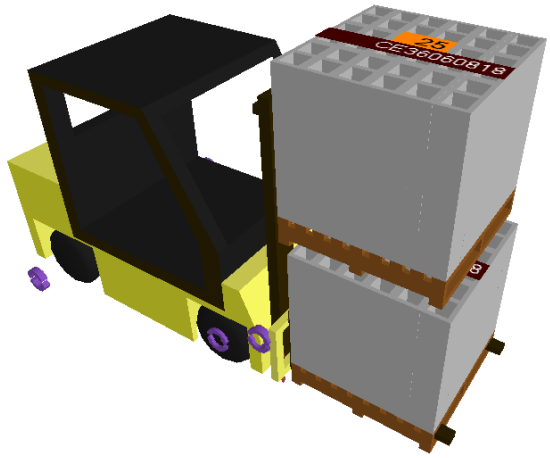

Figure 5.16: A production vehicle carrying two material entities.

5.2.4.4 Deadlock prevention

As previously stated, the modeling approach in Simio is mostly object oriented. Effectively, the model logic is distributed along its objects, which can make previously unforeseen issues occur. The most common issue that occurs due to this modeling paradigm is deadlocks.

A deadlock consists of a model state in which the objects are simultaneously waiting for any other object to perform a task and no object can change its current state. Effectively, this locks the model and the complete simulation does not occur.

Two situations where a deadlock could occur were detected in the model. The first is vehicle access to storage locations. Initially, vehicles would access storage locations through a bidirectional path with unit capacity so that only one vehicle could cross that path at any time. As hinted in Simio's reference guide, this can often result in a deadlock situation where a vehicle is crossing the path in one direction and another vehicle is waiting on the other end to cross and there is no space for the first vehicle to exit the path. This deadlock would never be resolved during the simulation run. To prevent this situation, two unidirectional paths were defined instead of one bidirectional path. This results in a simplification of the real system, increasing the simulation error. However, this situation is rare when a low number of vehicles is used.

The other situations where deadlocks would occur is when production requests are being processed. As previously mentioned, processing requests consists of searching storage locations in the buffer for available materials to fulfill the request. When a material is found, the corresponding entity is transferred to the storage location's output buffer. When multiple requests of the same material type are being processed, often a deadlock would occur in that a request would try to transfer a material to the output buffer, but since another request was already executing that action, the first request would enter a state of permanently trying to transfer the material and being unsuccessful. This situation happens in Simio because when a Transfer step is used in a process, the simulation schedules that action to the next simulation step. Before the action completes, the other request's process begins and also tries to transfer the same material. To prevent this situation, it was defined

that requests of the same material type can't be processed simultaneously, effectively preventing the deadlock situation with little to no impact in the simulation error.

5.2.5 Results and discussion

The simulation of this application was designed to study the effects of changing to the new proposed layout and using AGV, as well as the impact of changing the buffer utilization, that is, the global storage space utilization, and the number of vehicles. Recalling, the storage space utilization artificially reduces the capacity of its station (top and bottom stations) in the storage. For instance, when buffer utilization is set to 50%, all storage locations are used, but each only have half effective capacity. Deciding the way in which reducing buffer utilization would reflect on the real system is out of the scope of this project. However, if a lower buffer utilization scenario seems more advantageous, a new buffer layout revision can be studied to increase the free space in the buffer zone.

Firstly, it is important to mention the constant model variables, that is, the characteristics that model the real system and do not change over all the results. These characteristics can be found in Table 5.4. The table mentions that the time to replace the battery is 10 minutes. Thus, it is implied that the choice of the battery management system is a replacement system. Using a charging system would greatly impact performance when only one AGV is used. In addition to the table, there is the operator's work schedule, which consists of two 10 minute breaks and a 110 minute break every 8 hours, for a total 130 minutes of break every 480 minutes. Moreover, relating to equation 5.2, the values used throughout the different model runs were $\alpha = 10, \beta = 1, \gamma = 1$.

Using manufacturers' information on real AGV characteristics, the AGV speed was fixed at 1.3 meters per second in the buffer zone, taking into account that other AGV can cross between lanes and thus a lower speed should be used. However, when traveling to and from the warehouse, as the lane is reserved for the AGV, its speed can go up to 1.5 m/s. Since this path is modeled as a time path, the travel time is affected by the increased speed, although lower than the operator vehicles at 1.8 m/s. The travel time of the AGV is obtained from the travel time of the operator vehicle and the ratio between both vehicle's speeds.

The first part of the study consists in validating the model, ideally by using the current system's characteristics in the model and comparing the resulting performance to the real system's. Although real input data is available, no performance measurements exist from the real system. Thus, a qualitative analysis is made to validate the model. The measured statistics are the average and maximum request fulfillment time and the average warehouse vehicle utilization. The request fulfillment time, in Simio, is equivalent to the request's time in system. Vehicle utilization is measured as the percentage of the run in which the vehicle was operating. As the operator break times consist of 130 minutes for every 480 minutes, the theoretical maximum individual vehicle utilization is 73%. Acceptable measurements for the average request fulfillment time should be lower than 20 minutes; this is the most important measurement when comparing different scenarios.

The request orders used as input to the model are taken from the real system's orders generated during the whole month of May, 2016. As it is desired to study the buffer's performance, and it is

Variable	Description	Layout	Vehicle Type	Value (units)
Replications	The simulation replications for a given scenario	-	-	10
StorageContentsRankingRule	How materials are stored; affects the next pallet to exit the storage station	Original New	-	LIFO FIFO
TimeTravelProduction	Time a vehicle takes to travel to and from the production	-	-	3 (min)
TimeTravelWarehouse	Time a vehicle takes to travel to and from the warehouse	-	Operator AGV	3 (min) 3.6 (min)
VehicleSpeed	How fast a vehicle will move when traveling in links	-	Operator AGV	1.8 (m/s) 1.3 (m/s)
VehicleLoadTime	Time a vehicle takes to load an entity	-	Operator AGV	25 (s) 25 (s)
VehicleUnloadTime	Time a vehicle takes to unload an entity	-	Operator AGV	25 (s) 25 (s)
VehicleCapacity	The amount of entities a vehicle can transport at any time	-	-	2
BatteryChangeDuration	Time it takes to replace or recharge a vehicle's battery	-	AGV	10 (min)
BatteryChangeInterval	Time in between battery replacements or recharges	-	AGV	8 (hours)

Table 5.4: Fixed variable values in the different runs of the model.

unknown what the system bottleneck is, the number of production vehicles was set to **10**, although in the real system, the number of vehicles is 2. The number of vehicles was chosen to make sure there wouldn't be a bottleneck created by lack of vehicles. The results of running the simulation with the current layout, using operators, can be seen in table 5.5. Buffer utilization ranges from 20% to 100% and the number of warehouse vehicles goes from 1 to 5. The simulation duration was a whole month, starting on the 1st of May.

The overall results are consistent in the sense that with a higher number of vehicles and/or a higher buffer utilization result in lower request fulfillment times. Expected minimum request fulfillment times are about 7 minutes considering the 3 minutes it takes to travel from production to the buffer, as well as material loading and unloading times.

It can be seen that with the actual system's configuration, that is, 100% buffer utilization and 1 warehouse vehicle, the results are acceptable with an average of 13 minutes for the requests' fulfillment time. However, from the simulation results, increasing the number of vehicles to 2 further improves this measurement. Vehicle utilization is high, although not reaching the expected maximum, meaning there are times where re-stocking is not necessary. The maximum request fulfillment time measurement is impacted mostly by request order bursts, that is, a larger number of request orders happening during a smaller period of time. Alternatively, large request quantities can impact this measurement. It can be seen that going from 1 to 2 vehicles decreases this measurement by about 260%.

In order to lower the buffer's utilization below 90% with acceptable results, at least 2 vehicles would have to be used, and buffer utilization can go down as low as 40%. On the other hand, using 3 vehicles means even a 20% buffer utilization is close to being acceptable. It is in fact a trade-off between free space in the buffer, and the number of vehicles. The performance gains obtained from going over 3 vehicles are very minimal.

Overall, the system bottleneck in this situation is a combination of the following factors:

- High traveling distance between the warehouse, buffer and production locations;
- Low number of warehouse vehicles;
- Limited buffer capacity, causing a lack of materials in some scenarios.

For the second part of the study, it is desirable to see the impact of using the new layout. In this case, since the base capacity is 8 per storage location, the buffer utilization is varied to result in an effective capacity of 1,2,...,8. Other variables kept the same value as when using the first layout. The results are presented in Table 5.6.

It is clear that using the new layout, performance has improved considerably along all scenarios. In this case, a buffer utilization of 25% and 2 vehicles are sufficient to achieve acceptable results, compared to the 40% in the old layout. Comparing the current system's characteristics at 100% utilization and 1 vehicle, the average fulfillment time decreases by 4 minutes to 9 minutes. This number is fairly close to the expected minimum of 7 minutes for any request. With the new layout, having more than 1 vehicle results in negligible performance gains until buffer utilization

Buffer Utilization (%)	Warehouse Vehicles	Average Vehicle Utilization (%)	Average Request Fulfillment Time (min)	Maximum Request Fulfillment Time (min)
20%	1	69%	118.683	481.738
20%	2	30%	27.4893	190.914
20%	3	18%	21.3234	125.403
20%	4	12%	19.6987	102.709
20%	5	10%	19.232	88.0494
30%	1	67%	101.078	453.646
30%	2	33%	23.9132	192.142
30%	3	20%	18.118	129.142
30%	4	14%	16.56	102.154
30%	5	11%	16.0731	91.467
40%	1	66%	75.8325	426.584
40%	2	29%	18.625	148.049
40%	3	17%	14.179	92.0095
40%	4	12%	13.0824	70.2803
40%	5	9%	12.6493	55.5369
50%	1	66%	52.9089	388.298
50%	2	29%	15.4579	131.967
50%	3	19%	12.304	79.9495
50%	4	13%	11.4266	58.3405
50%	5	10%	11.1389	47.3046
60%	1	65%	36.6893	368.33
60%	2	29%	13.611	127.601
60%	3	18%	11.3664	77.1737
60%	4	13%	10.7204	57.1993
60%	5	10%	10.5019	45.0108
70%	1	65%	31.1351	340.926
70%	2	29%	12.6619	124.947
70%	3	18%	10.8516	70.003
70%	4	12%	10.3509	51.5235
70%	5	10%	10.1574	41.4387
80%	1	65%	21.4133	280.55
80%	2	29%	11.1695	122.073
80%	3	18%	10.1007	68.6795
80%	4	12%	9.74237	48.9583
80%	5	10%	9.61827	39.6145
90%	1	65%	18.4014	269.168
90%	2	29%	10.6906	117.566
90%	3	18%	9.84534	62.788
90%	4	13%	9.52643	42.6215
90%	5	10%	9.43095	34.8935
100%	1	66%	13.3057	218.048
100%	2	29%	9.67323	83.4089
100%	3	18%	9.27159	54.5416
100%	4	13%	9.17127	39.9749
100%	5	10%	9.13494	33.2097

Table 5.5: Results of the current layout using operators, in May.

is lower than 62.5%, when considering only the average fulfillment time. If the maximum time is considered, the performance impact is considerable even at 100% utilization.

An important note for the previous study is that the study of material distribution, presented in section 5.2.3, was made according to the request orders of May, the same request orders which were used to obtain the results of Table 5.6. The results can thus be misleading if later months have different request order distributions. However, the case in study has a fairly uniform monthly distribution of requests. Later in this section, another study uses the request orders of June, and the results will be compared to the the previous month.

The third part of the study consists of analyzing the impact of adopting an AGV-based system in the new layout. The same characteristics of the previous study are used, except for the work schedule, as previously defined. The AGV characteristics are presented in Table 5.4. The results for the new layout using AGV, for the month of May, can be seen in Table 5.7.

The results show an overall decrease in performance, due to the AGV slower speed. The decrease is greater when lower buffer utilizations are used, meaning that an AGV system using the same amount of vehicles as the current system would require a higher buffer utilization to achieve the same performance. A comparison is shown in Fig.5.17 for the more interesting results of using 1 and 2 vehicles. It can be seen that although there is a considerable hit in performance when 1 vehicle is used for a buffer utilization lower than 75%, adding another AGV brings both scenarios' performance much closer. In fact, it shows that the performance gains from using more than 1 vehicle are much greater in the AGV solution. However, this is true only when using a lower buffer utilization. Overall, acceptable performance can be reached with 2 vehicles and 25% buffer utilization, similarly to the case of using operators in the new layout. Using a single vehicle, the buffer utilization can be as low as 62.5% to achieve acceptable performance, a worse result than the previous study's 50% (in the new layout, using operators, for the month of May).

Moreover, the results show that using a single AGV may not satisfy the system's performance requirements when the maximum request fulfillment time is considered. They point to a possible choice between 2 and 3 AGV, deciding based on the trade-off between free space and the number of vehicles.

As previously mentioned, a fourth part of the study refers to testing the new layout with the same model characteristics, in the month of June. Table 5.8 shows the results and comparison to the month of May, focusing only on the main performance measurement which is the average request fulfillment time. Moreover, only the measurements for 1, 2 and 3 warehouse vehicles are presented; as seen from previous results, having over 3 vehicles yields negligible performance gains.

The results show some variation in performance, mostly increasing it, when 2 or 3 vehicles are used. However, when one vehicle is used, the performance lowers. As the results do not point to a strict increase or decrease in performance, the previous results for the month of May are considered valid. The differences in the results from May to June are attributed to different request distributions over the month and variable burst request quantities. Some differences are, however, considerably large, especially for a single vehicle and buffer utilization lower than 75%.

Buffer Utilization (%)	Warehouse Vehicles	Average Vehicle Utilization (%)	Average Request Fulfillment Time (min)	Maximum Request Fulfillment Time (min)
12.5%	1	68%	103.521	500.7
12.5%	2	31%	23.9854	168.091
12.5%	3	19%	18.4669	105.565
12.5%	4	14%	17.0442	86.0684
12.5%	5	11%	16.5431	80.2622
25.0%	1	65%	52.9219	405.496
25.0%	2	28%	14.5647	129.745
25.0%	3	17%	12.2041	74.423
25.0%	4	12%	11.5316	54.7798
25.0%	5	9%	11.2993	49.0395
37.5%	1	65%	26.5355	294.345
37.5%	2	30%	11.6554	103.212
37.5%	3	18%	10.5082	55.8937
37.5%	4	13%	10.163	40.1364
37.5%	5	10%	10.0228	36.0311
50.0%	1	65%	13.2257	188.752
50.0%	2	29%	9.74233	61.5411
50.0%	3	17%	9.44654	40.9837
50.0%	4	12%	9.30627	33.0646
50.0%	5	10%	9.27794	30.0149
62.5%	1	65%	10.8855	126.626
62.5%	2	29%	9.40742	45.1471
62.5%	3	18%	9.20946	35.4714
62.5%	4	13%	9.11596	29.6651
62.5%	5	10%	9.09725	29.0918
75.0%	1	65%	9.40209	65.2924
75.0%	2	29%	9.03003	34.9275
75.0%	3	18%	8.95432	30.9633
75.0%	4	12%	8.92815	29.6034
75.0%	5	10%	8.90104	29.129
87.5%	1	65%	9.23228	61.566
87.5%	2	29%	8.98862	33.198
87.5%	3	18%	8.91631	30.2416
87.5%	4	13%	8.89676	29.2563
87.5%	5	10%	8.87167	28.6155
100.0%	1	65%	8.89308	49.6534
100.0%	2	29%	8.79642	28.9554
100.0%	3	17%	8.77742	28.3034
100.0%	4	12%	8.76942	28.0282
100.0%	5	10%	8.76871	27.9991

Table 5.6: Results of the new layout using operators, in May.

Buffer Utilization (%)	**Warehouse Vehicles**	**Average Vehicle Utilization (%)**	**Average Request Fulfillment Time (min)**	**Maximum Request Fulfillment Time (min)**
12.5%	1	84%	248.453	946.387
12.5%	2	38%	32.8581	243.703
12.5%	3	18%	22.3395	149.019
12.5%	4	11%	19.7792	111.703
12.5%	5	6%	18.8503	99.837
25.0%	1	82%	126.536	766.373
25.0%	2	34%	18.6008	182.918
25.0%	3	17%	13.8572	101.515
25.0%	4	10%	12.6474	70.4766
25.0%	5	6%	12.2277	62.0771
37.5%	1	82%	65.1777	556.318
37.5%	2	37%	13.5335	146.835
37.5%	3	19%	11.4557	79.1313
37.5%	4	11%	10.8902	54.6059
37.5%	5	7%	10.6449	46.1087
50.0%	1	82%	22.3654	311.07
50.0%	2	34%	10.4954	85.6129
50.0%	3	17%	9.96179	55.1737
50.0%	4	11%	9.71048	42.5198
50.0%	5	6%	9.66027	37.3283
62.5%	1	82%	16.42	341.358
62.5%	2	34%	9.94018	59.9048
62.5%	3	18%	9.63393	45.9519
62.5%	4	11%	9.47678	35.9846
62.5%	5	6%	9.43772	33.1193
75.0%	1	82%	10.7933	174.29
75.0%	2	34%	9.39243	43.2565
75.0%	3	18%	9.2847	36.6492
75.0%	4	11%	9.22865	32.059
75.0%	5	6%	9.18921	30.4668
87.5%	1	82%	10.0925	117.03
87.5%	2	34%	9.33796	39.0057
87.5%	3	18%	9.23337	34.9949
87.5%	4	11%	9.19041	32.1741
87.5%	5	6%	9.14889	29.5679
100.0%	1	82%	9.26938	72.031
100.0%	2	34%	9.07059	31.4976
100.0%	3	18%	9.04709	30.4892
100.0%	4	11%	9.03144	28.73
100.0%	5	6%	9.02918	28.8277

Table 5.7: Results of the new layout using AGVs, in May.

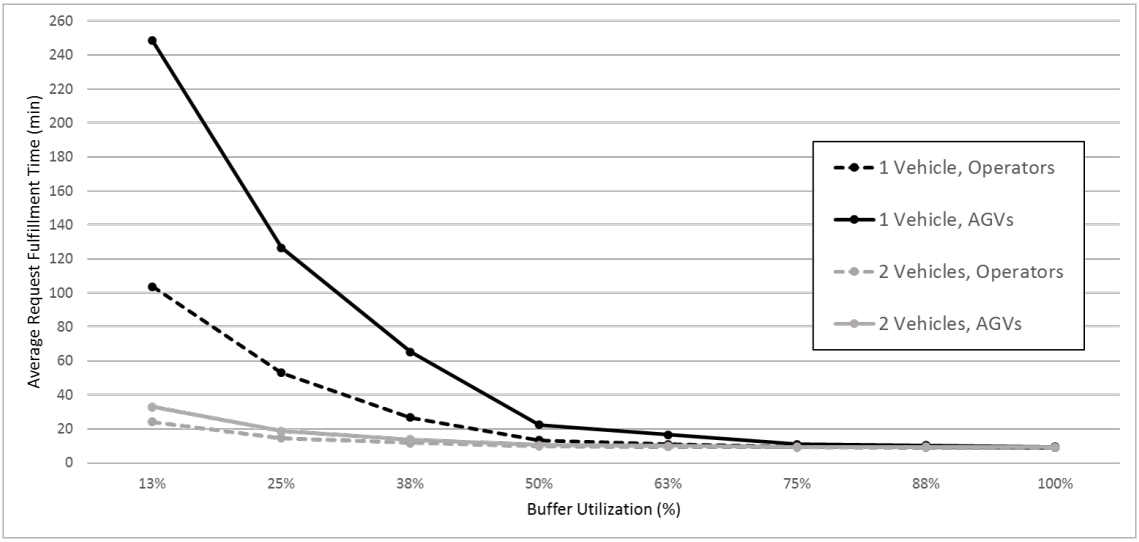

Figure 5.17: Comparison of using operator and AGV vehicles in the new layout.

Buffer Utilization (%)	**Warehouse Vehicles**	**Average Request Fulfillment Time (min)**	
		May	**June**
12.5%	1	248.453	137.347
12.5%	2	126.536	33.3363
12.5%	3	65.1777	25.0214
25.0%	1	22.3654	77.942
25.0%	2	16.42	19.5896
25.0%	3	10.7933	15.3326
37.5%	1	10.0925	49.5609
37.5%	2	9.26938	15.7075
37.5%	3	32.8581	12.8774
50.0%	1	18.6008	29.0468
50.0%	2	13.5335	12.6158
50.0%	3	10.4954	11.2015
62.5%	1	9.94018	21.9827
62.5%	2	9.39243	11.6917
62.5%	3	9.33796	10.7523
75.0%	1	9.07059	13.3629
75.0%	2	22.3395	10.4498
75.0%	3	13.8572	10.0836
87.5%	1	11.4557	12.5253
87.5%	2	9.96179	10.2843
87.5%	3	9.63393	9.99203
100.0%	1	9.2847	10.5775
100.0%	2	9.23337	9.79094
100.0%	3	9.04709	9.70941

Table 5.8: Results of the new layout using AGVs, in June, comparing to the results of May.

These differences can probably be attributed to request order bursts. Previously it was stated that using a single vehicle, a buffer utilization of 62.5% would be sufficient. However, for the month of June, the results show it is not enough.

The difference in performance over two months show that, when deciding on the final application's characteristics, data from more months should be used to make sure the results are valid over every month. One can also use a smaller time frame and perform more simulation runs to have more results. Alternatively, the system can be somewhat over-dimensioned to accommodate performance variability and uncertainty. It should also be noted that the simulation error in this model is unknown.

Up until this point, the model characteristics from Table 5.4 have been used. Specifically, 10 vehicles have been used in the production side in order to not create a potential bottleneck in that side of the system.

The final part of this study consists of running the model while setting the number of production vehicles to the real system's number, which is 2. This model is run for the month of June, as the previous part of the study, and all other model characteristics are maintained. The results are shown in Table 5.9.

The results show there is a clear bottleneck in the system created by the production side, where the system can't get an average request fulfillment time lower than 19 minutes, for scenarios where the previous results showed an average of 10 minutes was easily possible. Recalling, 20 minutes for the average request fulfillment time is the number chosen as the maximum time for a scenario to be considered acceptable.

Given the previous results, there seems to be essentially two options. One option is to target the system's performance so that the average request fulfillment time is at 20 minutes, while retaining a low maximum request fulfillment time as well. Looking at the results from Table 5.8 where there isn't a bottleneck, this can be achieved consistently over the months of May and June with scenarios using 2 vehicles and at least 50% buffer utilization. The other option is to further improve the production side's performance, for instance by using one more vehicle. However, in order to have more accurate results on the performance bottleneck created by the production side, the model should be extended to simulate not only the buffer layout but also the production and, potentially, the warehouse layout. Moreover, the simulation error should be taken into account.

With this project it has been shown that an AGV based system with 2 vehicles is capable of generating acceptable performance for this case study when using the proposed layout. It was previously stated that companies often sell an over-estimated number of vehicles to make sure it meets the system's performance goals, while many times not generating a similar study to have conclusive data. In this case, the number of vehicles acquired could easily have been 3, while it was shown that at a maximum of 2 vehicles shows acceptable performance in different buffer utilization scenarios.

Buffer Utilization (%)	Warehouse Vehicles	Average Vehicle Utilization (%)	Average Request Fulfillment Time (min)	Maximum Request Fulfillment Time (min)
12.5%	1	82%	138.316	695.81
12.5%	2	37%	35.9184	256.723
12.5%	3	23%	29.9063	188.992
25.0%	1	78%	80.8175	556.163
25.0%	2	34%	24.9913	156.758
25.0%	3	20%	22.5458	134.109
37.5%	1	78%	55.3365	479.689
37.5%	2	36%	24.0239	151.411
37.5%	3	22%	22.382	155.028
50.0%	1	78%	35.9849	394.455
50.0%	2	34%	21.0873	128.822
50.0%	3	21%	20.4058	128.478
62.5%	1	78%	30.738	330.007
62.5%	2	35%	21.3046	145.646
62.5%	3	22%	20.8592	146.234
75.0%	1	77%	22.8363	179.163
75.0%	2	34%	20.3957	130.017
75.0%	3	21%	20.2464	134.822
87.5%	1	78%	22.5891	181.549
87.5%	2	35%	20.6486	151.962
87.5%	3	21%	20.4692	159.071
100.0%	1	77%	20.3693	146.706
100.0%	2	34%	19.8546	137.742
100.0%	3	21%	19.817	139.473

Table 5.9: Results of the new layout using AGVs, in June, with 2 production vehicles.

Chapter 6

Conclusions

This dissertation is tri-fold. First, the exploration of the new web technologies applied to robots and ROS applications allowed to create a simple and clear user interface to allow non-proficient operators to execute the setup and operating processes with AGV. Although the back-end technology in web-based ROS interfaces are still under developed when compared to the usual interfaces, it was interesting to use programming languages created for this exact purpose instead of the more common - in robotics - desktop application programming languages. Some contributions were made to the repositories of official ROS web related libraries, to be used by the ROS community. The advantages that come from building a web-based application are several, but more movement towards this paradigm is required to achieve the same results as desktop-based interfaces.

The initial hypothesis was that setup costs in AGV-based solutions can be reduced by allowing operators to execute this process instead of the manufacturer or company selling the solution. In fact, a simpler interface can be built by putting most of the work on its developer, and finding an appropriate balance between flexibility and simplicity is a step in the way towards this goal.

The second, smaller part of this dissertation consisted of a study of more cost effective positioning sensors to be used in an AGV's self-localization solution. Robustness and applicability in the industry is important, as more flexibility and greater accuracy requirements become the norm. It was found that there has been an effort towards lowering the setup costs by using a lower number of artificial landmarks, which are commonly found to be installed in dozens or even hundreds in AGV applications. The concern with using more efficient processors, which in turn needs efficient and applicable localization algorithms to be used.

The final part of this dissertation focused on evaluating the performance impacts of adopting an AGV solution for a logistics application, as well as a focus on keeping the solution cost effective by choosing an appropriate number of AGV, among other variables. To this end, a real material handling scenario was analyzed.

Modeling was accomplished through computer simulation using Simio. As it was seen from the system's complexity, a simulation approach was definitely adequate. Besides giving results to the study in this dissertation, the developed model can be used in the future as a platform to study the implication of material changes, layout changes, and other situations.

It was decided to use real input data on the simulation application. Despite the extra effort this imposes when it comes to model development, this decision has the benefit of providing much more accurate results and reducing the overall simulation error, and it is advised to use real data whenever possible for simulation of complex systems.

Working together with people experienced in industrial environments and applications proved to be a rewarding experience, certainly one that will be useful in the future. As the market gets more competitive everyday, sharing knowledge and experiences in improving systems' performance with cost effective solutions also grows in importance.

References

[1] MarketsandMarkets. Automated Guided Vehicle Market by Type (Unit Load Carrier, Tow Vehicle, Pallet Truck, Assembly Line Vehicle), Industry Vertical (Automotive, & Others), Application (Transportation, Distribution, & Others), & Geography - Global Forecast to 2020, 2015. URL: http://www.marketsandmarkets.com/Market-Reports/automated-guided-vehicle-market-27462395.html.

[2] R Siegwart, IR Nourbakhsh, and D Scaramuzza. *Introduction to Autonomous Mobile Robots*, volume 2nd ed of *Intelligent Robotics and Autonomous Agents*. MIT Press, Cambridge, MA, 2011. URL: http://www.google.com/patents/US4638445$\delimiter"026E30F$nhttp://books.google.com/books?hl=en{&}lr={&}id=4of6AQAAQBAJ{&}oi=fnd{&}pg=PP1{&}dq=Introduction+to+Autonomous+mobile+robot{&}ots=2vZ81-tNF{_}{&}sig=VGm5Y{_}CHeLLi3cLK3mtraumaSdE.

[3] Morgan Quigley, Ken Conley, Brian Gerkey, Josh FAust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Mg. ROS: an open-source Robot Operating System. In *Icra*, volume 3, page 5, 2009. URL: http://pub1.willowgarage.com/{~}konolige/cs225B/docs/quigley-icra2009-ros.pdf, doi:http://www.willowgarage.com/papers/ros-open-source-robot-operating-system.

[4] Christopher Crick, Graylin Jay, and S Osentoski. Rosbridge: Ros for non-ros users. In *Proceedings of the 15th ...*, pages 1–12, 2011. URL: http://cs.okstate.edu/{~}chriscrick/Crick11b.pdf.

[5] Christopher Crick, Sarah Osentoski, Graylin Jay, and Odest Chadwicke O.C. Jenkins. Human and robot perception in large-scale learning from demonstration. In *Proceedings of the 6th international conference on Human-robot interaction - HRI '11*, pages 339–346. ACM, 2011. URL: http://dl.acm.org/citation.cfm?id=1957656.1957788, doi:10.1145/1957656.1957788.

[6] R Toris, J Kammerl, D V Lu, J Lee, O C Jenkins, S Osentoski, M Wills, and S Chernova. Robot Web Tools: Efficient messaging for cloud robotics, 2015. doi:10.1109/IROS.2015.7354021.

[7] Henrik Andreasson, Abdelbaki Bouguerra, Marcello Cirillo, Dimitar Nikolaev Dimitrov, Dimiter Driankov, Lars Karlsson, Achim J. Lilienthal, Federico Pecora, Jari Pekka Saarinen, Aleksander Sherikov, and Todor Stoyanov. Autonomous transport vehicles: Where we are and what is missing, 2015. doi:10.1109/MRA.2014.2381357.

[8] Saadettin Erhan Kesen and Ömer Faruk Baykoç. Simulation of automated guided vehicle (AGV) systems based on just-in-time (JIT) philosophy in a job-shop environment.

Simulation Modelling Practice and Theory, 15(3):272–284, mar 2007. URL: http://www.sciencedirect.com/science/article/pii/S1569190X06000852, doi:10.1016/j.simpat.2006.11.002.

[9] J. S. Dagpunar. Discrete Event Simulation. In *Simulation Modeling and Analysis with ARENA*, pages 135–156. Elsevier, 2007. URL: http://www.sciencedirect.com/science/article/pii/B9780123705235500031, doi:10.1002/9780470061336.ch7.

[10] Amin Nikakhtar, Kuan Yew Wong, Mohammad Hossein Zarei, and Ashkan Memari. Comparison of Two Simulation Software for Modeling a Construction Process. In *2011 Third International Conference on Computational Intelligence, Modelling & Simulation*, pages 200–205, 2011. doi:10.1109/CIMSim.2011.42.

[11] L Schulze, S Behling, and S Buhrs. AGVS in logistics systems state of the art, applications and new developments, 2008. URL: {\T1\textless}GotoISI{\T1\textgreater}://WOS:000254816800029.

[12] R.E. Ward. An Overview of Basic Material Handling Equipment, 1986.

[13] Rainer Mautz. *Indoor Positioning Technologies*. PhD thesis, Habilitationsschrift ETH Z{ü}rich, 2012, 2012. URL: http://e-collection.library.ethz.ch/eserv/eth:5659/eth-5659-01.pdf, doi:10.3929/ethz-a-007313554.

[14] J. Borenstein, H. R. Everett, L. Feng, and D. Wehe. Mobile robot positioning: Sensors and techniques. *Journal of Robotic Systems*, 14(4):231–249, 1997. URL: http://doi.wiley.com/10.1002/(SICI)1097-4563(199704)14:4{\T1\textless}231::AID-ROB2{\T1\textgreater}3.3.CO;2-1.

[15] J Borenstein and Liqiang Feng. Measurement and correction of systematic odometry errors in mobile robots, 1996. doi:10.1109/70.544770.

[16] S. Kurtti, J. P. Jansson, and J. Kostamovaara. A laser scanner chip set for accurate perception systems. In Gereon Meyer, editor, *Advanced Microsystems for Automotive Applications 2012: Smart Systems for Safe, Sustainable and Networked Vehicles*, chapter A laser sc, pages 313–322. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. URL: http://dx.doi.org/10.1007/978-3-642-29673-4{_}29, doi:10.1007/978-3-642-29673-4_29.

[17] Miguel Armando Migueis Pinto. SLAM for 3D map building to be used in a matching localization algorithm, 2012. URL: http://digitool.fe.up.pt:1801/webclient/DeliveryManager?custom{_}att{_}2=simple{_}viewer{&}metadata{_}request=false{&}pid=743128.

[18] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[19] D Ronzoni, R Olmi, C Secchi, and C Fantuzzi. AGV global localization using indistinguishable artificial landmarks, 2011. doi:10.1109/ICRA.2011.5979759.

[20] Héber Sobreira, Miguel Pinto, António Paulo Moreira, Paulo Gomes Costa, and José Lima. Robust robot localization based on the perfect match algorithm, 2015. URL: http://gateway.webofknowledge.com/gateway/Gateway.cgi?

GWVersion=2{&}SrcApp=METALIBSearch{&}SrcAuth=EXLIBRIS{&}DestApp=WOS{&}DestLinkType=FullRecord{&}UT=WOS:000345285200058, doi:10.1007/978-3-319-10380-8_58.

[21] Martin Lauer, Sascha Lange, Martin Riedmiller, and Martin Riedmiller Martin Lauer, Sascha Lange. Calculating the perfect match: an efficient and accurate approach for robot self-localization. *Robocup 2005: Robot soccer world cup ...*, 4020(c):142–153, 2006. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.68.253$\delimiter"026E30F$nhttp://link.springer.com/chapter/10.1007/11780519{_}13$\delimiter"026E30F$nhttp://www.springerlink.com/index/V59M05J034N40228.pdf.

[22] Shung Han Cho and Sangjin Hong. Map based indoor robot navigation and localization using laser range finder, 2010. doi:10.1109/ICARCV.2010.5707420.

[23] J Minguez, L Montesano, and F Lamiraux. Metric-based iterative closest point scan matching for sensor displacement estimation, 2006. doi:10.1109/TRO.2006.878961.

[24] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141, may 2001. URL: http://www.sciencedirect.com/science/article/pii/S0004370201000698, doi:10.1016/S0004-3702(01)00069-8.

[25] Dieter Fox. KLD-sampling: Adaptive particle filters. In *Advances in Neural Information Processing Systems 14*, volume 14, pages 713–720, 2002. URL: http://www-2.cs.cmu.edu/Groups/NIPS/NIPS2001/papers/psgz/AA62.ps.gz$\delimiter"026E30F$nhttps://papers.nips.cc/paper/1998-kld-sampling-adaptive-particle-filters.pdf, doi:10.1.1.21.5786.

[26] Heber Sobreira, a. Paulo Moreira, Paulo Gomes Costa, and Jose Lima. Robust Mobile Robot Localization Based on Security Laser Scanner, 2015. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7101627, doi:10.1109/ICARSC.2015.28.

[27] Shuzhi Sam Ge and Frank L Lewis. *Autonomous mobile robots: sensing, control, decision making and applications*. CRC Press, 2006. URL: http://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:Autonomous+mobile+robots:+sensing,+control,+decision-making,+and+applications{#}0.

[28] S.B. Williams, G. Dissanayake, and H. Durrant-Whyte. Efficient Simultaneous Localization and Mapping Using Local Submaps. *IEEE International Conference on Robotics and Automation*, 1(November 2001):128–134, 2001. doi:10.1109/ROBOT.2002.1013394.

[29] Sebastian Thrun, Daphne Koller, Zoubin Ghahmarani, and Hugh F. Durrant-Whyte. SLAM updates require constant time. *Workshop on the Algorithmic Foundations of Robotics*, pages 1–20, 2002. URL: http://cogvis.nada.kth.se/{~}hic/SLAM/Papers/thrun{_}paper2.pdf.

[30] Yufeng Liu and S. Thrun. Results for outdoor-SLAM using sparse extended information filters. *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, 1:1227–1233, 2003. URL: http://ieeexplore.

ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1241760, doi:10.1109/ROBOT.2003.1241760.

[31] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based SLAM, 2010. doi:10.1109/MITS.2010.939925.

[32] S Thrun and M Montemerlo. The GraphSLAM algorithm with applications to large-scale mapping of urban structures. *INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH*, 25(5-6):403–429, 2006. doi:10.1177/0278364906065387.

[33] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proc. of 8th National Conference on Artificial Intelligence/14th Conference on Innovative Applications of Artificial Intelligence*, volume 68, pages 593–598, Edmonton, Canada, 2002. AAAI. URL: http://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:FastSLAM+:+A+Factored+Solution+to+the+Simultaneous+Localization+and+Mapping+Problem{#}0, doi:10.1.1.16.2153.

[34] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with Rao-Blackwellized particle filters, 2007. doi:10.1109/TRO.2006.889486.

[35] Dirk Hähnel, Dirk Schulz, and Wolfram Burgard. Mobile robot mapping in populated environments. *Advanced Robotics*, 17(7):579–597, nov 2003. URL: 10.1163/156855303769156965http://search.ebscohost.com/login.aspx?direct=true{&}db=a9h{&}AN=10876647{&}lang=pt-br{&}site=ehost-live, doi:10.1163/156855303769156965.

[36] Joao Machado Santos, David Portugal, and Rui P. Rocha. An evaluation of 2D SLAM techniques available in Robot Operating System, 2013. doi:10.1109/SSRR.2013.6719348.

[37] Ioannis K. Chaniotis, Kyriakos-Ioannis D. Kyriakou, and Nikolaos D. Tselikas. Is Node.js a viable option for building modern web applications? A performance evaluation study. *Computing*, 97(10):1–22, 2014. URL: http://link.springer.com/10.1007/s00607-014-0394-9, doi:10.1007/s00607-014-0394-9.

[38] Miguel Pinto, Héber Sobreira, A. Paulo Moreira, Hélio Mendonça, and Aníbal Matos. Self-localisation of indoor mobile robots using multi-hypotheses and a matching algorithm. *Mechatronics*, 23(6):727–737, sep 2013. URL: http://linkinghub.elsevier.com/retrieve/pii/S0957415813001281, doi:10.1016/j.mechatronics.2013.07.006.