Università degli Studi di Ferrara

DOTTORATO DI RICERCA IN
SCIENZE DELL'INGEGNERIA

CICLO XXIII

COORDINATORE Prof. Trillo Stefano

Traffic Management of Automated Guided Vehicles in Flexible Manufacturing Systems

Settore Scientifico Disciplinare ING-INF/04

Dottorando
Dott. Olmi Roberto

Tutori
Prof. Bonfè Marcello

Prof. Secchi Cristian

Anni accademici 2007/2009

Day of the defense: March 25, 2011

Abstract

The objective of the research carried out by Roberto Olmi has been the development of an innovative traffic management system (TMS) for fleets of automated guided vehicles (AGV) operating in industrial environments. These vehicles are used for the transport of materials in environments where the presence of people and moving machinery has to be expected. The TMS has to control the motion of the vehicles so as to avoid collisions and optimize transport times.

The approach is based on the decomposition of the problem into three parts: assignment of transport tasks, path computation, motion control of each AGV along its path. Transport orders are assigned to available vehicles by solving a linear assignment problem through the Jonker Volgenant algorithm. The objective is to minimize the sum of the times required by each vehicle to reach the pick station assigned to it. The motion of the vehicles is defined incrementally by a controller that periodically computes the list of the trajectory segments that each AGV is allowed to cover. The sequence of segments is determined so as to minimize the nominal travel time. In order to define the movement of vehicles along their routes, a tool called Coordination Diagram (CD) has been applied. Through this instrument, the problem of coordinating the motion of N AGVs is transformed into a shortest path problem within an N dimensional space with obstacles.

The main contribution of this dissertation is the application of the CD to the problem of coordinating a fleet of AGVs in industrial environments. In [1], [2] and [3], the student has developed an original method to calculate efficiently the CD. This method is based on an approximation of the CD that allows to reduce the computational cost of the algorithm and thus makes it suitable for the coordination of tens of vehicles. To calculate the

shortest path to the CD, in [4] and [5] a complete planning solution has been proposed that provides greater efficiency with respect to an incremental approach. However, since the CD changes each time that a path of a vehicle is modified, an incremental approach is required for the coordination of AGVs. In particular, in [3] the action that maximizes the advancement of the fleet while respecting the constraints imposed by the CD is determined by solving Maximum Independent Set problem. The solver is an heuristic algorithm which has time complexity quadratic in the number of vehicles considered.

In order to test the proposed TMS, a simulation software has been developed using Matlab. The simulations consider AGVs operating on real layouts developed by Elettric80 S.p.A., a manufacturer of AGV. The proposed algorithm is much more flexible than the one currently used by the company. It does not require the manual definition of traffic rules and it allows a significant savings of time ($>$ 30 days / person) in the construction of new plants. The research project presented in this dissertation has been selected as a finalist at the "European Technology Transfer Award 2010" organized by the associations EUROP[1] and EURON[2].

[1]EUROP (www.robotics-platform.eu) is the European Robotics Technology Platform, an industry-driven framework for the main stakeholders in robotics.

[2]EURON (www.euron.org) is the European Robotics Research Network, a community of more than 200 academic and industrial groups in Europe.

Sommario

L'attività di ricerca svolta da Roberto Olmi ha avuto come obiettivo lo sviluppo di un innovativo sistema di gestione del traffico (TMS) per flotte di veicoli a guida automatica (AGV) operanti in ambiente industriale. Tali veicoli sono destinati al trasporto di materiali in ambienti dove è prevista la presenza di persone e macchinari in movimento. Il TMS deve controllare il moto dei veicoli in modo da evitare collisioni e ottimizzare i tempi di trasporto.

L'approccio adottato si basa sulla scomposizione del problema in tre parti: assegnazione delle missioni di trasporto, calcolo dei percorsi, controllo dell'avanzamento di ciascun veicolo lungo il suo percorso. Gli ordini di trasporto vengono assegnati ai veicoli disponibili risolvendo un problema di assegnamento lineare tramite l'algoritmo di Jonker e Volgenant. L'obiettivo è quello di minimizzare la somma dei tempi impiegati da ciascun veicolo per raggiungere la stazione di carico ad esso assegnata. Il moto dei veicoli viene definito incrementalmente da un controllore che periodicamente trasmette agli AGV la lista dei segmenti di traiettoria che essi devono percorrere. La sequenza dei segmenti viene calcolata in modo tale da minimizzare il tempo nominale di viaggio. Allo scopo di definire il moto dei veicoli lungo i loro percorsi è stato applicato uno strumento chiamato Coordination Diagram (CD). Tramite questo strumento il problema di coordinazione del moto di N veicoli viene trasformato in un problema di ricerca del cammino minimo in uno spazio N dimensionale con ostacoli.

Il principale contributo di questa tesi consiste nell'applicazione del CD al problema della coordinazione di una flotta di AGV in ambiente industriale. In [1], [2] e [3], il dottorando ha sviluppato un metodo originale che permette di calcolare in maniera efficiente il CD. Tale metodo si basa su una

approssimazione del CD che permette abbattere il costo computazionale dell'algoritmo e lo rende quindi idoneo per il coordinamento di decine di veicoli. Per il calcolo del cammino minimo all'interno del CD è stata proposta in [4] e [5] una soluzione di pianificazione completa, che garantisce una coordinazione più efficente rispetto ad un approccio incrementale. Tuttavia, la necessità di aggiornare continuamente il CD rende necessario un approccio incrementale nella pianificazione del moto dei veicoli. In particolare, in [3] l'azione di moto che massimizza l'avanzamento della flotta garantendo il rispetto dei vincoli viene determinata risolvendo un problema di Maximum Independent Set, grazie ad un algoritmo euristico con complessità quadratica nel numero di veicoli considerati.

Per verificare i risultati dell'attività di ricerca, è stato sviluppato in Matlab un software in grado di simulare veicoli operanti su gli impianti reali sviluppati da Elettric80 S.p.A., produttore di sistemi AGV. L'algoritmo proposto risulta molto più flessibile di quello usato dall'azienda, in quanto non richiede la definizione manuale di regole di traffico e permette un significativo risparmio di tempo (> 30 gg/uomo) nella realizzazione di nuovi impianti. Il progetto di ricerca qui presentato è stato selezionato come finalista al concorso europeo "Technology Transfer Award 2010" organizzato dalle associazioni EUROP[1] ed EURON[2].

[1]EUROP (www.robotics-platform.eu) è la piattaforma tecnologica europea per la robotica, un'associazione di soggetti operanti a livello insustriale nel settore della robotica.

[2]EURON (www.euron.org) è il network della ricerca robotica europea, una comunità di oltre 200 gruppi accademici e industriali in Europa.

To my family

Acknowledgements

I would like to thank Prof. Cesare Fantuzzi, Dr. Cristian Secchi and Dr. Marcello Bonfè for the opportunity they have given me of doing this work. I also wish to greatly acknowledge the company Elettric 80 s.p.a. (www.Elettric80.com) which supported my research activity and all its staff I have worked with. Their experience has been an essential contribution to the development of this work.

This thesis is dedicated to my family. In particular I want to thank my wife Annalisa, who has always believed in me. Last but not the least, I thank my parents Giuseppe and Paola. This result is mainly due to what they have taught me.

Contents

Glossary

Symbols

$a_1, \ldots, a_7$	Lines defining the forbidden region $\mathcal{F}$
$\mathcal{A}(q_i)$	Portion of space occupied by an AGV at the configuration $q_i \in \mathcal{R}$
b_{k_i,k_j}	Block: a rectangular region $\Delta^i_{k_i} \times \Delta^j_{k_j}$ of the plane $\mathcal{S}_{ij}$
BT_h	Booking table of the segment τ_h (see Sec. 3.3)
$\mathcal{C}$	Configuration space of the AGVs
χ_i	Ratio between the distance covered by $\mathcal{A}_i$ on current segment and the total length of that segment
c_{ij}	Time nominally required by the i-th empty AGV in order to execute the j-th pick mission (see Sec. 2.2.1)
CSS_h	Collision segments set, the set of segments that are colliding with τ_h
$\mathcal{D}(u)$	Cost function for the action choice
δ	Time period between two consecutive calls to the coordinator algorithm (=0.5s for the considered application)
$\Delta^i_{k_i}$	Time interval in which $\mathcal{A}_i$ is expected to travel along the segment k_i of its path
d_i	Binary variable which defines the motion direction of $\mathcal{A}_i$ (see Sec. 5.3)
$d_{\mathbf{ss'}}$	Nominal time required by the fleet to reach the new configuration $\eta'.\mathbf{s}$ from $\eta.\mathbf{s}$ (see Sec. 3.4.3)

$\mathcal{E}$	Enclosing rectangle
$e_1, \dots, e_6$	Lines defining the enclosing rectangle and the blocks grouping
η	Programming object representing a coordination path $\gamma(t)$ (see Sec. 3.4.2)
$\eta.CC$	Coordination components
$\eta.d$	Time duration of the last action
$\eta.ES$	Extension stage
$\eta.P$	Parent path
η_R	Root path (with which the algorithm in Chap. 3 is initialized
$\eta.\mathbf{s}$	Ending point of the coordination path η
$\eta.\mathbf{u}$	Direction of the last coordinated action used for extending η
U_η	Set of directions $\mathbf{u}$ along which the path η can be extended (see Sec. 3.4.3)
$\tilde{U}_\eta$	Subset of the possible extension directions defined in equation (3.10)
$\mathcal{F}$	Forbidden region
$f(\eta)$	Overall cost function associated to a coordination path η (see Sec. 3.5)
$F_{\mathbf{t}}(i)$	Time factor (see equation (4.6))
$\gamma(t)$	Coordination path: defines a coordinated motion of the AGVs for the time $t \in [0, t_{end}]$ (see Sec. 3.4.1)
$g(\eta)$	Cost associated to the execution of η (see equation (3.12))
H	Number of stations
$h(\eta)$	Heuristic underestimate of the minimum cost of the remaining path to the goal from η (see equation (3.14))
K	Number of segments
K_{acc}	Parameter which penalizes the number of accelerations of a coordination path (see Sec. 3.4.1)

κ	Constraint returned by the Alg. 9
k_i	Index of the sequence of segments constituting the path p_i (see Sec. 5.2.2)
l_i	Segment computed by the coordinator algorithm in order to update r_i
L_i	Estimated amount of delay accumulated by $\mathcal{A}_i$ while another AGV advances. (see Sec. 5.3.2)
m_i	Number of segments in which a path π_i is uniformly partitioned (see Sec. 4.1)
N	Number of AGVs of the fleet
n_{acc}^i	Number of times that the vehicle $\mathcal{A}_i$ changes its speed (see Sec. 3.4.1)
n_i	Number of segments of a path p_i
$\pi_i(s_i)$	Mapping of the path assigned to $\mathcal{A}_i$ (see Sec. 3.2.1)
$\mathbf{Q}$	Ordered queue of the paths (see Sec. 3.4)
ρ	Advancing factor
r_i	Last reserved segment for $\mathcal{A}_i$
$\mathbf{s}$	Point of the CD
S	Subset of AGVs for which the coordinator algorithm is updating the list of reserved segments
$\mathcal{S}$	N-dimensional space in which the CD is defined
$\mathcal{S}_{ij}$	Plane of the CD relative to the paths p_i and p_j
$\mathbf{s}_G$	Goal configuration of the fleet (within the CD)
$\mathbf{s}_I$	Initial configuration of the fleet (within the CD)
s_i	Nominal time required by $\mathcal{A}_i$ to reach the position $\pi_i(s_i)$
τ	Segment of the roadmap
$\mathcal{T}$	Set of segments constituting the roadmap

T_i	Nominal time required by $\mathcal{A}_i$ to reach the destination
$\mathbf{u}$	Coordinated action (defines the motion of each AGV)
u_i	Motion action of the vehicle $\mathcal{A}_i$
U_ρ	Subset of coordinated actions which lead to the same advancing factor (see equation (4.1))
u^*	Coordinated action chosen by the algorithm
$W_i(l_i)$	Time required by $\mathcal{A}_i$ in order to reach the segment l_i (see equation (5.4))
$w^i_{k_i}$	Nominal time required by $\mathcal{A}_i$ to cover the segment k_i
X^{coll}_{ij}	Collision region within the CD
(x_f, y_f)	Top right corner of the reference block b_{l_i,l_j} with $i < j$
(x_s, y_s)	Bottom left corner of the reference block b_{l_i,l_j} with $i < j$
x^c_i, y^c_i	Coordinates of the center of $\mathcal{A}_i$ (see Sec. 3.2.3)
x^I_i, y^I_i	Initial coordinates of the center of $\mathcal{A}_i$ (see Sec. 3.2.3)

Acronyms

AGV	Automated Guided Vehicle
AGVS	AGV-Systems
DES	Discrete Event System
FMS	Flexible Manufacturing Systems
LAPJV	Algorithm for Linear Assignment Problem by Jonker and Volgenant ([8])
MHS	Material Handling Systems
MM	Mission Manager
PSLT	Department of Planning and Control of Warehouse and Transport Systems of the University of Hannover

TMS	Traffic Management System
WMS	Warehouse Management System

Chapter 1

Introduction

In modern factories the necessity of increasing production volumes while decreasing costs in order to be more competitive in the global market is becoming more and more crucial. This necessity can be fulfilled by automatizing the production and logistic processes and robotics has contributed and is contributing to this purpose. Today it is possible to find robotic cells and more innovative robotic solutions in many production plants. In particular, Automated Guided Vehicles (AGVs) have been introduced since the 1950s for automatic material handling systems (MHS). Because of their flexibility and efficiency, many AGV systems (AGVS) are used in production lines and today they are more and more exploited also at the end of the production line, namely in automatic warehouses where huge quantities of goods are continuously moved.

Flows of products and materials are central to the primary process of many enterprises and as a consequence, efficient and effective material flow management has drawn much attention. Material handling systems are complex combinations of material, machines and people. Efficiency in space utilization, material handling and AGV control systems was given little consideration. However, rapid development of technology in handling equipment and increasing cost of labor and material, spurs companies to improve the design and management of material handling environments. In order to avoid congestion of goods, AGVS need to never stop during working hours. The number of AGVs that need to be used is growing more and more and their motion need to be controlled in such a way that each AGV reaches its destination as quickly as possible. Thus, traffic management is one of the main issues to be addressed in logistic systems.

This dissertation is focused on the development of a traffic management system (TMS) for AGV-based transportation system. This chapter gives an overview of the AGVS and motivates the objectives of this dissertation in more detail. In Sec. 1.1 an analysis of the AGV market is given in order to highlight the relevant economical interest on the development of AGVS. Next, in order to obtain a better idea of the tasks that have to be performed by an AGVS, an example is reported in Sec. 1.2. In Sec. 1.3, after a brief analysis of the main system requirements, the most important issues concerning the design of an AGVS are reported. The objective and motivations of the presented work are reported in Sec. 1.4. The chapter ends with the outline of the dissertation.

1.1 AGV Market

It has been estimated that between 20 and 50% of the total operating expenses within manufacturing can be attributed to material handling [9]. Furthermore, material handling is estimated to represent between 15 to 70% of the total cost of a product. It is therefore the first place to be considered for cost reduction. However, lowering the product costs is not the only requirement in order to be competitive nowadays. Modern enterprises have also to increase their flexibility. The continuous variations in the product mixes and priorities, require to adopt flexible transportation systems such as AGVs, which enable flexible material routing and dispatching.

The first AGV has been introduced in the market by Barrett Electronics in the 1950s for serving a production line. It was a simple, slow, tow truck following a wire in the floor instead of a rail. Scientific and technological advancements have greatly improved the AGVs. In particular, the navigation system, which was based on following an hardware layer (i.e. wires, nails) that had to be installed in the environment where the vehicles were moving, is now more and more based on laser triangulation.

Because of their flexibility and efficiency, many AGVs are used in production lines and today they are more and more exploited also at the end of the production line, namely in automatic warehouses, where huge quantities of goods are continuously moved. Companies producing convenience goods, like those in the food field or in the paper field, have greatly invested and are still investing in the automation of the logistics. The possibility of covering 24 hours of work using a single AGV increased the market of AGV systems in those countries where the labor cost is high and where, therefore, the return of the investment is quick. In Europe, the countries where AGV systems have been mostly employed are Italy, Spain, France, Germany, the United Kingdom and the Scandinavian countries [10]. According to a survey by the Department of Planning and Control of Warehouse and Transport Systems of the University of Hannover (PSLT) [11], containing data until year 2006, the market of AGV systems in Europe is flourishing. The increasing interest in AGVS is reflected in the sales figures which reached a new peak in 2006 with a volume of 200 mil. EUR. In comparison to the year 2000, about a quarter of the AGV systems manufacturers are emergent vendors. On the other hand, the established vendors offer new and different achievement profiles. Both aspects point out the dynamics of the vendor side which offers, with more than 25 European manufacturers, a large variety of solutions.

A substantial indicator for the market tendency of AGVS is the annual number of produced vehicles. The number of AGVS put into operation worldwide by European AGVS manufacturers sums up to over 2800 new systems and over 700 enlargement systems, with about 27,500 AGV in total. After an intermediate flattening related to the number of the AGVs and AGVS put into operation, a significant increase can be registered since 1999 until 2006. Thus in the three-annual average, the level rose in the meantime over 140 new systems per year. In 2006, a new peak with over 169 AGVS was reached. A similar process is determined for the number of AGVs put into operation. It is also a trend that the average system size measured in vehicles per system rises. The average number of vehicles per system amounts now again over six vehicles. As the complexity systems increase, the requirements on planning, engineering, project management, realization and putting into operation rise. This trend is also pointed out by the fact that the average equipment price allocated on the vehicles increased.

1.2 Description of an AGVS

Modern logistics facilities such as warehouses, distribution centers, production plants and transshipment terminals all have AGV-based material transport systems in common. The next example is used to demonstrate and explain the process of material transport in more detail. Fig. 1.1 shows an example warehouse.

Order requests for loads are received from customers or clients such as other facilities, retail or department stores. These orders are received on-line during the day, and will be considered for shipment. The storage areas are replenished with incoming loads, which are outputted by the production lines or delivered by trucks during the day. The warehouse management system (WMS) then allocates the orders to locations within the storage areas, and order picking routes are determined. AGVs have to travel through the aisles between the racks in the storage area to visit the locations to collect the orders. These routes are determined with objective criteria such as: minimize the travel time or the waiting time caused by traffic congestion.

The sequence of the executed orders depends on their priorities, whether the orders involves single or multiple pallet, etc. Since the pick times and travel times of AGVs are stochastic (due to the acceleration/deceleration effects, failure of equipment, unexpected obstacles along the paths, etc.), the drop off instants of the loads are also stochastic. Although an approximate schedule of the orders can be defined, the exact arrival time and contents of the orders are not known, this fact makes scheduling the transportation of the loads beforehand impossible. Furthermore, determining combinations of delivering loads and retrieving loads at a particular location beforehand will also be impossible since the exact arrival times of vehicles or release times of loads can not be predicted.

In AGVS, the load transfer locations can be programmed into a vehicle control system in advance. Such a control system can be a central controller, which assigns transport tasks to vehicles. The process of selecting and assigning transport tasks to vehicles is called dispatching. The AGVs are in general dispatched on-line, i.e. based on real-time information, since the uncertainty of the load release and delivery times makes vehicle dispatching beforehand (scheduling) impossible. Monitoring vehicle positions and traffic control can be performed by the central computer, or through local controllers.

If the traffic is not properly treated, congestion, deadlocks or even collisions can take place. These situations can block part of the system and they can require to stop the AGVs to allow the intervention of qualified personnel for a manual restart. Besides drastically reducing the performance of the system, manual restarts have a negative impact on the customer perception of the AGV system. The traffic control problem is further complicated by the presence of unpredictable events that can take place in

Figure 1.1: Example of a warehouse

automatic warehouses. For example, a pallet may fall during the transportation or an AGV can suddenly stop because of a fault. These events produce some unplanned static obstacles along the routes tracked by the vehicles. Furthermore, more and more "mixed" (i.e. automatic and manned) warehouse systems are present in the market. Because of economical reasons or of logistic issues, only a portion of a warehouse can be made automatic and part of the goods is still managed by human guided forklifts. In these situations, the AGVs are working in an environment populated by moving obstacles (the forklifts) they cannot absolutely collide with because of safety reasons.

1.3 Quality requirements and design choices

The technology associated with material handling has changed dramatically during the last three decades, mostly due to the introduction of computers and automation. The problem of efficiently managing the traffic of AGVs is getting more and more relevant. In particular, the traffic management is recognized as one of the main issues for the development of an AGV system both by the industrial and by the scientific communities (see e.g. [12]). The main quality requirements for an AGVS are performances, configurability and robustness [13]. Performance is a major quality requirement, customers expect that transports are handled efficiently by the transportation system. The number of transportation tasks per time unit that can be done by a given number of vehicles have to be maximized. Coordinating the motion of the AGVs in such a way that traffic congestion is minimized is essential in order to attain good performances. Configurability is important, it allows installations to be easily tailored to client-specific demands. From a control software point of view, this property is related to the easiness of installation on many different plant with different needs. The less are the software components that have to be customized when a new AGVS is deployed, the more configurable is the system. In current plants tens of AGVs are circulating at the same time and humans and manually guided forklifts can be present in the same environment. This makes the robustness another primary concern. The AGVs must be able to navigate autonomously without colliding and intervention of service operators have to be avoided since it is costly and time consuming.

In order to meet these requirements a lot of choices have to be done in the design and control processes of an AGVS. They belong to three different levels of the decision-making process which are denoted as:

- Strategic decisions level
- Tactical decisions level
- Operational decisions level

At the level of the strategic decisions the relevant long-term constraints are fixed. They regard the design of the facility layout and the definition of the material flows (the interested reader may refer to [9]). Issues at tactical level include estimating the number of vehicles, guide-path design, positioning idle vehicles and managing battery charging scheme. The decisions at this stage have a strong impact on decisions at other levels. Finally, problems such as AGV scheduling, AGV routing, conflict avoidance, deadlock resolution and/or prevention are addressed at operational level. Operational level involve the decisions which have to be taken online (i.e. while the system is working) by the TMS. Some recent surveys regarding tactical and operational issues are [12], [14] and [15]. There is a high interaction between the strategic, tactical and operational decisions. For example, the number of AGVs is dependent on the control of the vehicles, and the control of the vehicles is dependent on the constraints, the performance criteria and the layout. An integrated approach to these levels of decision seems impossible and one often uses a nested approach, where first the strategic decisions are made based on rough tactical and operational ideas, followed by fine-tuning the tactical and operational decisions. Each decision influences the others and have to be done considering the kind of application (e.g. container terminals rather than automatic warehouses) for which the system is designed.

1.4 Motivation and objectives of the dissertation

The requirement of producing a vast and ever-changing number of goods at a lower and lower production costs puts under pressure many companies. In this scenario, a lot of manually operated systems have been replaced by intelligent machines offering higher efficiency and reliability. An evidence of this technological evolution is given by the widespread application of AGVS as a flexible transportation systems in harbors, warehouses, storages, and product distribution centers. For AGVS manufacturers, the design of a TMS that is collision-and deadlock-free requires a lot of engineer time and it needs to be heavily re-adapted when installing the AGV system in another warehouse. Furthermore, unexpected obstacles and/or faults often require to stop the system for manual recovery. This leads to a decrease of performance and to a bad impact on the customer perception of the system.

This dissertation shows an efficient, fault-tolerant traffic control strategy that has been successfully applied in a simulated AGVS. The main objective is to make the traffic management of the AGVs automatic and efficient. No tuning depending on the topology of the warehouse needs to be done and good performance have to be guaranteed also in presence of faults and mobile obstacles as human guided forklifts. This will drastically reduce the engineer time required for each installation and the number of required stops of the system leading to better performance, to a significant reduction of installation costs and to an increase of customers satisfaction.

The main functionalities required to the TMS are tasks dispatching, path computation and AGVs coordination. Since the vehicles can not exit the system when no tasks

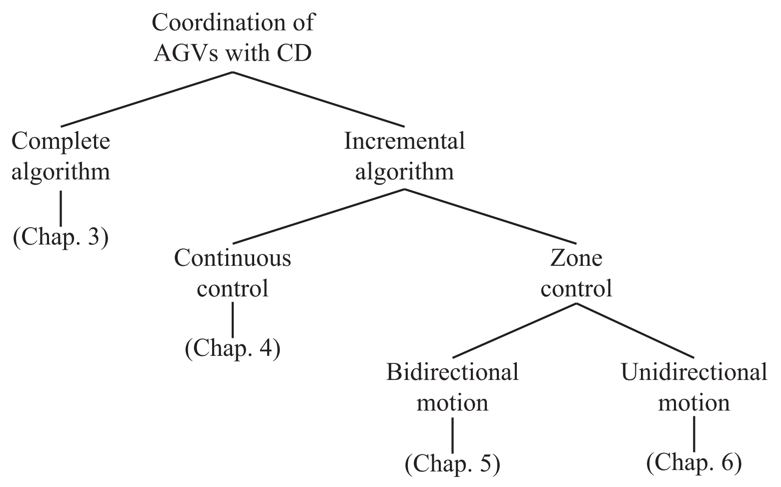


Figure 1.2: Taxonomy of the algorithms for the coordination of AGVs presented in this thesis

are required, the system is also responsible for handling idle AGVs.

1.5 Outline of the dissertation

The overall objective of the dissertation is the development of a TMS for industrial AGVS. The next chapter gives a description of the considered framework along with the functionalities that are expected from the TMS. In particular the adopted solutions for the dispatching, routing and managing idle AGVs are reported. The next four chapters are devoted to the development of innovative algorithms for solving the coordination problem. The four works can be classified as reported in Fig. 1.2. The common feature these works is the application of the CD for determining the way in which the AGVs have to move along their precomputed paths.

In Chap. 3 we first give introduction to the CD. Then an algorithm for efficiently computing the CD is developed. Subsequently, exploiting an approximated representation of the CD, we develop a complete coordination planning algorithm. The speed profile of each vehicle from its start point to its destination is computed before the vehicles start moving. A complete approach allows to plan an optimal coordination at the expense of an high computational complexity.

In order to coordinate the motion of AGVs within a dynamic industrial environment, where a lot of unexpected events may happen, an incremental coordination is more suitable. In Chap. 4 an algorithm which determines the motion of the AGVs step by step is designed. The motion of the AGVs is not planned in advance. At each iteration the algorithm defines the motion of the AGVs considering the actual configuration reached by the fleet. Unexpected events can be considered without the need of replanning.

Both the above mentioned approaches does not take into account the vehicles dynamics. Moreover they can not be easily implemented into the considered industrial framework (see next chapter). These issues are considered by the approach described in Chap. 5 and Chap. 6. The first consider the case in which the AGVs are allowed to move both in forward and in backward direction while the second considers only the forward motion. By considering only one possible motion direction, the latter algorithm is characterized by a computational complexity which is significantly lower than the previous ones.

In the last chapter some conclusions are drawn.

Chapter 2

Considered framework for TMS

This chapter presents a TMS which gives an efficient solution to the operational problems outlined before. The application scenario that has been considered is an automatic material handling system for warehouse and production plants. The TMS has to guarantee an efficient and conflict-free motion of a fleet of AGVs that move over a paths network in a dynamic industrial environment. The main functionalities required are tasks dispatching, path computation and AGVs coordination. Since the vehicles can not exit the system when no tasks are required, the system is also responsible for handling idle AGVs.

In Sec. 2.1 a description of the considered framework is given along with the functionalities that are expected from the TMS. The solutions adopted for the dispatching, routing and management of idle AGVs are presented in Sec. 2.2. The last two sections introduce to the AGV coordination problem. In Sec. 2.3 a review of the literature for solving the coordination problem of autonomous wheeled vehicles is given while in Sec. 2.4 the approaches proposed in this dissertation are outlined.

2.1 Considered framework and main objectives of the TMS

Operational problems have gradually gained an important role in the research on AGVS, as documented in recent surveys [12], [14] and [15]. An AGVS is often integrated into a larger framework which is referred as flexible manufacturing system (FMS). As pointed out by [16], recent literature offers integrated solutions for the overall operational control of an FMS. However, integrated approaches are based on the assumption that the

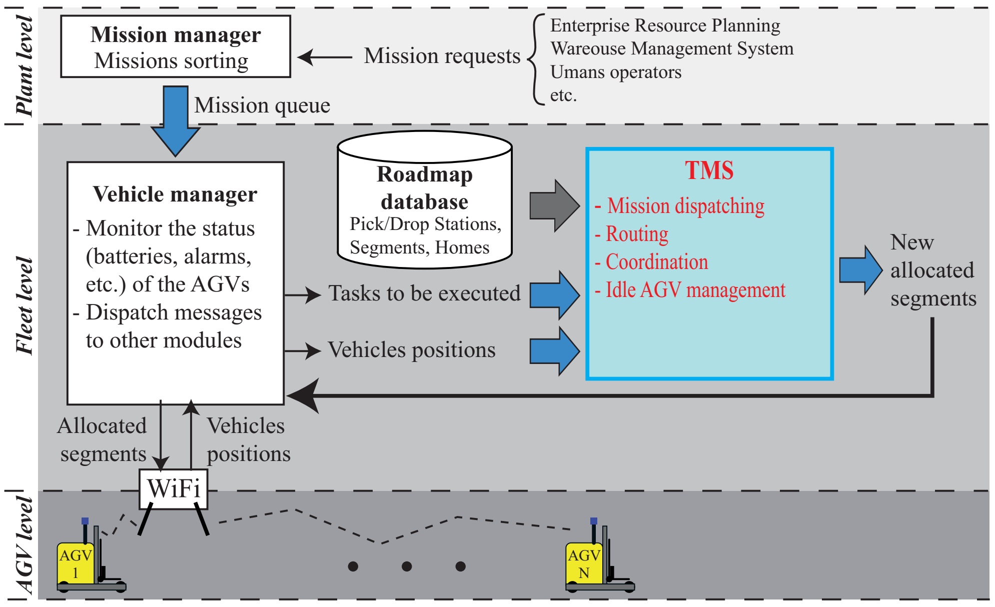


Figure 2.1: Overall control framework for AGVS considered in this dissertation.

entire control system is developed by the same manufacturer. In practice this assumption is seldom verified since FMS are complex systems composed of even more complex modules which are rarely produced by the same company. These facts motivates the development of modular control architectures. The work presented in [13], for example, studies the architectural design of a decentralized control system for industrial AGVS. They elaborate, in particular, on the integration of different software modules of the system. In this dissertation, the main focus is on the development of a TMS to be integrated into an existent industrial control framework.

The proposed TMS is designed to be applied into the existing framework developed by Elettric 80 S.p.A., a company producing end-of-line automation solutions and AGVSs for warehouses and production plants. In the considered framework, a network of paths covering the overall plant is defined considering the layout of the plant and the location of the pick-up and delivery stations. This network, called *roadmap*, is statically defined and it is stored as a database of trajectory segments and points. The paths of the vehicles are defined by sequences of roadmap segments. The overall control system for the fleet of AGVs is made up of three layers (see Fig. 2.1). The two higher layers are centralized, while the third is executed asynchronously on each vehicle. At *plant* *level*, the incoming transportation requests are collected from the FMS. The *mission manager* (MM) is responsible to define the sequence in which the missions have to be passed to the TMS so as to be executed by the AGVS. The *fleet level* where the TMS is located, monitors the status of the fleet and incrementally determines the trajectory segments which are reserved to any AGV in order to efficiently execute the transports while avoiding collisions with other vehicles. The main functionalities that have to be implemented in the TMS are:

- Dispatching: the goal is to determine the assignment of the transportation tasks to the AGVs.
- Routing: once that a mission has been assigned, the path that a vehicle has to track has to be determined.
- Coordination: the motion of the AGVs along their paths must be coordinated so as to avoid collisions and deadlocks. Once the path is computed, an algorithm, called *coordinator*, periodically computes the list of the segments that each vehicle is allowed to track, called *reserved segments*. The stream of the allocation of these lists determines the coordinated motion of the fleet.
- Idle AGV management: AGVs that do not have tasks to execute must be located at some parking positions defined in order to react as efficiently as possible to new missions. It is also important to avoid that an idle AGV block the motion of another vehicle.

At last, the *AGV level* implements the low level control of the AGV for the trajectory tracking. This level is also responsible for stopping the vehicle if an unforeseen obstacle is detected and for the loading and unloading operations. A vehicle tracks the list of trajectory segments which are reserved to it and stops when it reaches the last. In this way, even if the communication with some vehicle is lost, the safety is guaranteed by the fact that each vehicle will remain within its set of reserved segments. The TMS does not have any real time data about the vehicle thus, for safety reasons the insertion of a segment into the reserved list can not be undone. For that reason, each new path that has to be assigned to an AGV starts with the sequence of the segments which are in the reserved list when the path is computed. This approach is rather common in industrial layouts (see e.g. [17]) and it is called *zone control*.

Remark 1 Deadlock detection and solution: *The TMS is also responsible for solving the deadlocks that may occur by applying a given coordination strategy. In the case a deadlock is predicted, one of the blocked AGVs is rerouted along a new path. Many deadlock prediction techniques have been presented for AGV systems. In this thesis the approach presented in [18] is applied in order to solve the deadlocks that may blocks the AGVs.*

2.2 Dispatching, routing and management of idle AGVs

In this section we describe the methodologies applied for the implementation of three important functionalities of the TMS: dispatching of the transportation requests to the AGVs, routing (i.e. definition of the paths), and management of idle vehicles.

2.2.1 Dispatching

A *mission* is a request for an AGV to execute an operation at a given point, called *station*. When a transportation request is issued, two missions are generated: one for picking the load and one for dropping it at destination. The exact time a mission is generated is not known in advance. A picking mission is generated only at the time the load becomes available for the transfer while the drop mission is defined after the pick operation. All the missions are collected by the MM (see Sec. 2.1). The association of the missions to the vehicles is done in two stages. First, the MM decides which mission has to be executed, and subsequently the selected mission is passed to the TMS which is responsible to select the AGVs that have to execute the missions. The drop missions are executed as soon as they have been received by the MM whereas a pick mission is selected (if any is present in the MM queue) and passed to the TMS as soon as a drop mission has been completed. The criteria applied by the MM in order to select the pick missions are outside the scope of the presented work. The interested reader is addressed to [14] and [12] for a good literature review on this topic.

The TMS stores the missions released by the MM and defines which AGV has to execute each mission. A drop mission is trivially reserved to the AGV that has picked the load at which it is related. Thus, the TMS is only responsible to dispatch the pick missions to the empty AGVs. This problem can be modeled as an assignment problem in which all empty AGVs and all pick missions are considered simultaneously so as to minimize the total cost due to empty moves of the AGVs. The algorithm was originally proposed by [19] and, recently it has been adapted to the case of internal AGVS by [20]. The approach is based on the definition of a cost matrix in which each element c_{ij} corresponds to the cost of executing the mission j with the AGV i. Once that the cost matrix has been defined, the assignment problem is solved by applying the LAPJV (Linear Assignment Problem by Jonker and Volgenant) algorithm by [8].

In our implementation the cost c_{ij} is the time nominally required by the i-th empty AGV in order to execute the j-th pick mission. The nominal time is computed by considering the path length and the nominal speed of the vehicle. By applying the offline paths computation the nominal travel time can be retrieved in constant time (see Sec. 2.2.2). A mission is removed as soon as it has been completed by the corresponding AGV. In this way the number of missions that have to be assigned by the TMS is always less or equal than the number of vehicles.

2.2.2 Routing

The path that each vehicle has to track in order to execute the assigned mission is computed (using Dijkstra algorithm) without taking into account the presence of the other vehicles. A path is defined by a sequence of segments into which the roadmap covering the overall plant is partitioned. Since the roadmap is statically defined, it is possible to use the Dijkstra algorithm in order to compute offline all the minimum time paths between the segments and the pick/drop stations. In this way, for each segment, the next segment to be covered in order to reach a given station can be stored along with the minimum travel time between the segment and the station. All this data can be easily stored in a matrix data structure. This takes $O(2HK)$ space where H is the number of stations and K is the number of segments of the roadmap. By using this approach the length of the path to reach any station from any segment of the roadmap can be determined in constant time while the sequence of segments composing the path is retrieved in time $O(n)$ where n is the number of segments of the path. Allowing for a fast determination of the path length is important for the dispatching algorithm since it needs to know the distance between each AGV and the station points of the missions.

2.2.3 Management of idle AGVs

An AGV becomes idle if it has delivered a job at its destination and it is not immediately assigned to a new job. This task is indicated as *homing* and the locations are denoted as *home*. One of the decisions to make is where to locate idle AGVs so that they can react as efficiently as possible to a new assignment. [14] observes that most literature on this topic involves selecting home locations of vehicles in roadmaps composed of only one cycle. The research of [21] is the only one that takes conventional layouts into account. Their algorithms minimize the system response time to vehicle demand points when dispatched from home locations. However, their model does not consider possible traffic congestions induced by such choice. In the considered framework the home locations are statically defined and stored within the roadmap database. In general these places are defined so as to minimize the system response time to the requests for new transports. Another aspect to be taken into account is the minimization of the traffic congestion that can be generated when an AGV start moving from a particular home location. However the home locations are taken as a static input data by the TMS, thus these criteria their definition will not further analyzed. The criteria considered in order to define these locations is outside the scope of the thesis (the interested reader may refer to [12] and [14] for further details). The TMS is only responsible to dispatch idle vehicle to the home locations. In the considered framework the number of homes is approximately equal to the number of AGVs. In this way, the idle vehicles are distributed within the roadmap regardless of the dispatching criterion applied. In order to dispatch the idle AGVs to the home locations we propose to apply the same approach used for the mission assignment (see Sec. 2.2.1). Each time that a vehicle becomes idle the LAPJV algorithm is used in order to assign each idle AGV to an home location. The cost for assigning an AGV to a given home corresponds to the time that the vehicle takes to reach that location. In this way, our approach minimizes the total travel time of the idle AGVs and, as a consequence, also the traffic caused by the moving idle AGVs is minimized. Due to space restrictions, it is seldom possible to reserve an home location so that an idle AGV can stop without obstructing the way of other vehicles, especially near the pick stations. If an idle AGV waiting at home location blocks another vehicle, the idle AGV has to be dispatched to another location so as to free the way for the other vehicle. To this aim, the assignment of the homes is computed again. In this case, the cost for assigning this AGV to its location is set to infinity so that the idle AGV is dispatched to a different home location.

2.3 Coordination of wheeled autonomous vehicles

In this section a collection of the approaches presented in literature is given following the classifications given in [22] and [23].

2.3.1 Centralized versus decoupled

Focusing on mobile robots systems, the coordination strategies can be mainly classified in: *centralized approaches* and *decoupled approaches*. Centralized approaches search the solution of the motion planning problem in a composite coordination space, which is formed by the Cartesian product of the configuration spaces of the individual vehicles. In [24] a genetic algorithm is exploited in order to generate optimal (or near-optimal) paths for a set of non-holonomic vehicles, moving in a environment cluttered with static and moving obstacles. A Mixed Integer Non-Linear Programming problem is formulated in [25] in order to obtain the optimal velocity profile for each vehicle. Except from their very high complexity, this kind of approaches are non-robust to any contingencies arising during the system operation. They require to discard the plans every time the coordination problem must be updated (e.g., a new mission is assigned to a vehicle). In [26] an exact algorithm for computing Pareto optima for collision-free coordination is presented. These approaches are generally characterized by a significant computational burden so that their application is often limited to simple problem settings involving two or three vehicles. A time window based method have been developed and implemented in industrial applications for dynamic routing of AGVs [27]. Each time that a new mission is assigned to a vehicle, the approach is to select the shortest feasible path of each vehicle from among a set of candidate paths. Though this approach allows to avoid collisions and deadlock, its benefit can be poor since in many industrial layouts only a few possible candidate paths for a given mission exist.

Decoupled approaches face the complexity of the coordination, by breaking the problem into two distinct phases: path planning and motion coordination. During the first phase a path for each vehicle is planned without considering the presence of other vehicles. In the second phase the velocity profile of each vehicle along its path is computed. These approaches are more suitable than centralized ones for dealing with coordination problems involving a big number of vehicles. An approach for the time optimal coordination that can be considered somewhere between centralized and decoupled is presented in [28]. In this case however the computation time is still too high for real applications.

2.3.2 Centralized versus decentralized

A further characterization can be found in [23], where tree aspects are considered: the architecture of the system, the temporal scope of the planning and the decomposition of the coordination problem. The architecture can be *centralized* or *decentralized* (or *distributed*). In the first case, a central unit gathers the data of all the AGV and then it decides the motion of each AGV. The main advantage of these approaches is that they allow to obtain a very efficient coordination and, in principle, even the optimal one (with respect to a chosen functional). The main drawback of these algorithms is the computational burden. Traffic problems have been also treated using distributed architectures. In decentralized architectures, each vehicle decides its motion based on local information. Some interesting works, in which real implementations are reported, are [29], [13] and [30]. This approach is successful if the environment is populated by cooperative vehicles that aim at solving traffic jams. Unfortunately, this is not the case in industrial AGVS. In fact a traffic problem can be caused by an unexpected obstacle or by a human guided forklift which are not cooperative entities. These situations can severely limit the efficiency of the system. Another issue is the communication overhead required in order to share information among agents. The experiments in [13] show that the maximum bandwidth usage is almost directly proportional to the number of AGVs and with only five vehicles the usage is up to 40% of a 11Mbps network.

2.3.3 Complete versus incremental

Considering the temporal scope of the planning the algorithms can be subdivided into *complete* and *incremental*. Complete planners try to find a coordination for the whole mission of each vehicle in one shot while incremental algorithms coordinate the vehicles continuously and slightly ahead. With a complete approach the coordination is computed before the robots start moving and, if a vehicle deviates from the planned motion profile (e.g., due to an unexpected obstacle), the computation must be repeated from scratch. The main drawback of complete approaches is that they require to discard the plans every time the coordination problem must be updated (e.g. a new mission is assigned to a vehicle). This could require to stop the AGVs for a significant amount of waiting time for each new re-planning. Incremental techniques are convenient when the vehicles work in a dynamic environment in which some unexpected events may require the plans to be discarded. Moreover the computational complexity of an incremental algorithm is independent of the length of the plans, thus it is more suitable when real time requirements have to be satisfied. A particular class of incremental approaches is based on the theory of Discrete Event System (DES). It is also possible to model the AGVS as a Discrete Event System (DES) and the traffic control unit as a controller for this DES. Some relevant examples are [31] and [32] and [33] [34]. The main drawback of DES-based approaches is that the main focus is the development of a collision and deadlock free routing for the AGVs. Performance of the fleet is not often taken into account and it can be rather low in some situations. Furthermore, it is not clear how to consider unexpected events that could block some vehicles. In [31] for example, a condition that guarantees the existence of a solution to the problem of scheduling AGVs on roads is provided, considering that all vehicles move at constant speed. Exploiting this condition the authors develop an incremental algorithm that is deadlock free, and that runs in polynomial time (in the number of vehicles). For industrial AGVs however the assumption of constant speed is rarely satisfied, thus a more general model must be considered.

2.3.4 Global versus local

The third aspect depends on the total amount of resources (i.e. routes) needed for each mission. If this is much less than those available in the plant, the conflicts remain local and the solution can be found considering only a restricted number of vehicles (see, for instance, [35]). Differently the conflict resolution may, by propagation, involve the whole fleet. In this case a *global* technique is required. In AGVS the transportation tasks require all the vehicles to traverse long paths within the layout thus a global approach has to be considered.

2.3.5 Standard approaches based on traffic rules

The standard approaches for the coordination of industrial AGVs are based on a set of traffic rules manually defined during the construction of the roadmap. Some examples of this approach are [36], [37]. This requires a lot of engineering work when an AGVS must be deployed or modified since several exceptions have to be handled both for production and safety reasons.

2.4 The proposed CD-based approach

The coordination algorithms presented in the next chapters are based on the CD (coordination diagram). This tool, which is also called task completion diagram, was first introduced by [38] in order to coordinate two robot manipulators. The CD allows to map a coordination problem into a planning problem. Previous works which apply the CD for coordinating the motion of mobile robots are, [26], [28], [35], [39], [40]. These approaches try to plan a coordination path by means of standard search algorithms. For an introduction to this topic see [41]. In [35] the cylindrical structure of the CD is exploited in order to give an implicit cell decomposition of the diagram. Then the A* algorithm is applied to find a coordination path. A distributed approach is presented in [39]. In this case the CD is partitioned into a regular grid over which the D* algorithm searches for a coordination path. However, by minimizing the length of the coordination path, the obtained solution does not minimize the average time-to-goal. An algorithm that minimizes the average time-to-goal is presented in [28] where the optimal coordination path is found by applying the dynamic programming principles. All these approaches however suffer from a very high computational complexity (see Sec. 3.1) so that they can not be applied on a real AGVS composed many vehicles.

Our approach is characterized by a centralized architecture. This choice is due to the fact that in the industrial environment, mainly for safety reasons, a centralized coordination is still a requirement. Since the transportation tasks require all the vehicles to traverse long paths within the roadmap, a global algorithm is chosen. In Chap. 4 a complete approach is proposed for systems that are characterized by a low level of uncertainty (see also [4] and [5]). However, since the system that we are considering is characterized by an high level of uncertainty, the next chapters are focused on incremental approaches. To the best of the author's knowledge, the CD-based coordination algorithms proposed since now are complete. The works presented in Chap. 3, Chap. 5 and Chap. 6 is the first application of the CD for the development of an incremental coordination approach (see also [1], [2] and [3]).

Chapter 3

Complete coordination planning

In this chapter we propose a complete planning algorithm for computing a coordination plan for a fleet of vehicles moving along predefined paths. We first classify the possible collisions that can take place and the induced geometry of the resulting CD. This analysis justifies the approximated representation of the CD which is used next for developing the coordination algorithm. Subsequently, we show how to exploit the structure of the roadmap in order to develop a technique to build the CD online with a very limited computational effort. Loosely speaking, when the roadmap is defined we generate a set of offline sub-CDs. When a group of vehicles starts moving over the roadmap, the CD is composed by putting together the proper sub-CDs which are identified by the paths tracked by the vehicles. Exploiting the approximated representation of the CD, the algorithm defines the speed profile with which each vehicle must cover the assigned path. The algorithm generates a set of possible coordination plans and then gives the optimal one. Previous works apply traditional path planning techniques (such as D* or A*) to compute a coordinated motion. However these techniques do not take into account the fact that the CD has a cylindrical structure. This features enables our algorithm to explore the CD in a more efficient way. The coordination approach proposed in this chapter has been validated through experiments on real plants layouts. We present an example in which the coordinated motion of 10 vehicles is computed in only 12.4 sec. on a common PC.

The chapter is organized as follows: the first section introduces the proposed complete approach and gives an overview of some related literature. In Sec. 3.2 a formal definition of the problem is reported along with a classification of the possible kinds of collisions that can take place between two AGVs. In Sec. 3.3 the technique for computing the CD is presented. In Sec. 3.4 the algorithm for planning a coordinated motion of the AGVs is reported. In Sec. 3.5 the heuristic cost function is defined. In Sec. 3.6 some experiments are presented. Finally, in Sec. 3.7 some conclusions are drawn.

3.1 Introduction

Given a fleet of vehicles to be coordinated along assigned paths, the CD is a representation of all configurations of the fleet where mutual collisions might occur. A path through the CD, indicated as *coordination path*, defines a coordinated motion for the vehicles. Some approaches try to plan a coordination path by means of standard search algorithms. For an introduction to this topic see [41]. Examples are [35], [39] and [28]. These techniques however require the CD to be partitioned into a grid in which the search algorithm is used in order to find a coordination path. At each iteration step the number of grid elements explored by the algorithm can be exponential in the number of vehicles. To avoid this problem the algorithm proposed in [35] explores only along Manhattan paths, but then a smoothing technique is applied over the output path. Moreover the partition of the entire CD requires an enormous number of cells. As far as the problem instance is simple (i.e. the obstacle within the CD are small and sparsely distributed), the algorithm explores a small subset of cells. Differently the computation time will grow very quickly.

In this chapter we propose a new search strategy that explores the CD by incrementally expanding a set of coordination paths. Starting from a null length path placed at the the origin of the CD, at each iteration step the algorithm selects a path to be extended. This operation generates new paths that are identical to the extended one except for the last added segment. The exploration continues until an optimal path, among all the possible paths that can be explored by the algorithm, is found.

The main feature of our algorithm is that it drastically reduces the directions to be evaluated by explicitly considering the cylindrical structure of the CD. The set of directions used to extend each path is computed by considering two things: the direction of the last segment of the path and the position of the last point of the path compared to the obstacles. The directions will be generated by taking the last direction of the path and modifying only the components that are relevant in order to avoid the obstacles.

The second contribution of the chapter is the definition of an heuristic estimate of the cost function which takes into account the number of times that a vehicle has to stop and start its motion during the coordinated motion. This brings two benefits. The first one is that we define a trade-off between time optimality and smoothness of the motion of each vehicle. The second, but more important, benefit is that the number of explored states is further reduced thanks to the fact that the algorithm tends to extend only the smoother paths.

The procedure used for the expansion of a path is similar to that reported in [26] where an exact algorithm for computing Pareto optima paths is given. The input for this algorithm is a collision-free path; the output is the Pareto optimal path homotopic to the input. However the output path is only guaranteed to be a local minimum of the cost function considered. The proposed algorithm does not focus on a particular homotopy class but it searches among all possible homotopy classes.

3.2 Overview of the problem

3.2.1 Roadmap and missions

In the application that we are considering, a fleet of N AGVs have to move in the same environment sharing the same configuration space $\mathcal{C}$ (e.g. $SE(2)$). For a given plant to be served, a network of paths which the AGVs can follow is defined. This network is designed through a dedicated CAD program developed by Elettric 80 so as to avoid collisions between AGVs and static obstacles. We can model this network as a *roadmap* $\mathcal{R}$, that is a one-dimensional connected subset of $\mathcal{C}$. The roadmap is formed by a collection $\mathcal{T}$ of regular curves called *segments*. Each segment $\tau \in \mathcal{T}$ is a mapping $\tau : [0, 1] \to \mathcal{R}$. A *route* is defined as a sequence of adjacent segments.

We indicate with $\mathcal{A}(q_i)$ the portion of space occupied by an AGV at the configuration $q_i \in \mathcal{R}$. Two segments $\tau_i, \tau_j \in \mathcal{T}$ represents a pair of *colliding segments* if there exists a pair of scalars $(\alpha, \beta) \in [0, 1]^2$ such that

$$\mathcal{A}(\tau_i(\alpha)) \cap \mathcal{A}(\tau_j(\beta)) \neq \emptyset \tag{3.1}$$

This means that when two AGVs are moving through τ_i and τ_j it can happen that a collision takes place.

A centralized planning system plans a set of missions to be executed by the AGVs. In case no more missions are scheduled, a homing mission is assigned to the AGV (see Sec. 2.2.3) which is taken to a garage position. It is possible that the planning system can decide to change a mission previously assigned to an AGV(see Sec. 2.2.1). Furthermore, vehicles can be blocked by unexpected events (such as a person standing on its trajectory) for an unpredictable amount of time (see Sec. 1.2). Each AGV has to execute a mission, namely to reach a goal configuration. The path of each vehicle $\mathcal{A}_i$ is computed without considering the presence of other vehicles and it can be expressed as a mapping

$$\pi_i : s_i \to \mathcal{R}, \quad s_i = \left[0, T^i\right] \tag{3.2}$$

where s_i can be any scalar parameter that allows to express the position of the AGV along its trajectory. In particular $\pi_i(T^i)$ corresponds to the final configuration. In order to find a coordination that reduces the delay accumulated by each vehicle, it is convenient to choose a parameterization based on the expected traveling time. The scalar parameter $s_i \in [0, T_i]$ is the time that the vehicle $\mathcal{A}_i$ would take to reach the position $\pi_i(s_i)$ considering that it travels following the given nominal velocity profile.

Our goal is to determine the velocity profile by which each AGV has to move through the assigned path in order to avoid collisions among the AGVs and to minimize the total time required by the fleet for reaching the goal configurations. In this (and the next) chapter we do not take into account vehicle dynamics (we consider that each vehicle can instantaneously change its velocity). In order to guarantee that the coordination problem admits a solution, we assume that, for each AGV, the initial and the goal configurations do not belong to an already planned path.

3.2.2 Coordination diagram

The coordination strategy that we are going to develop in this chapter is based on the *coordination diagram* (CD). This tool gives a compact representation of the configuration of the AGVS. Given N paths $\pi_1, \dots, \pi_N$, parametrized by $s_1 \in [0, T_1], \dots, s_N \in [0, T_N]$, the CD represents all the configuration set of the vehicles along their paths and, therefore, it is given by $\mathcal{S} = [0, T_1] \times \cdots \times [0, T_N]$.

For each pair of paths, a *collision region* is defined as:

$$X_{ij}^{coll} = \{(s_1, \dots, s_N) \in \mathcal{S} \,|\, \mathcal{A}(\pi_i(s_i)) \cap \mathcal{A}(\pi_j(s_j)) \neq \emptyset\} \tag{3.3}$$

This region defines all the possible configurations of the fleet such that two vehicles collide moving along paths π_i and π_j. Since the collision region depends only on the configuration of two vehicles, this region can be completely characterized by its 2D projection onto the (s_i, s_j) plane of the CD (that we will denote shortly with $\mathcal{S}_{ij}$).

A point $\mathbf{s} = (s_1, \ldots, s_N)$ within the CD represents a possible configuration of the vehicles along their paths. We denote as $\mathbf{s}_I = (0, \ldots, 0) \in \mathcal{S}$ the initial configuration of the fleet and with $\mathbf{s}_G = (T_1, \ldots, T_N)$ the goal configuration. The motion of the vehicles is computed by searching for a path within the CD which avoids all the collisions among vehicles. Since the collisions depend only on the configurations of the pairs of vehicles, such a path can be found by considering just all the planes $\mathcal{S}_{ij}$ of the CD.

3.2.3 Taxonomy of the CDs

In this section we provide a classification of the possible collisions that can take place between two vehicles and the induced geometry of the corresponding collision diagram. For the sake of simplicity, we consider circular vehicles of the same size. We consider roadmaps composed only by straight segments, spaced enough so that vehicles on parallel segments don't collide. Under these assumptions, the condition for which there is a collision is given by:

$$\sqrt{(x_1^c - x_2^c)^2 + (y_1^c - y_2^c)^2} \leq d_{min} \tag{3.4}$$

where x_i^c and y_i^c, $i = 1, 2$, represent the coordinates of the center of the circle, and d_{min} is the diameter of the circle.

The regions of the CD corresponding to collision configurations of the fleet can be approximated with a rectangular region called *enclosing rectangle* (see Fig. 3.1) having two sides parallel to the bisector of the diagram.

There are only three possible kinds of collisions between two AGVs:

- *Intersection collision:* it takes place when the paths intersect each other (see Fig. 3.1). By the paths parameterizations (using notations in Fig. 3.1, where x_i^I, y_i^I are the initial coordinates of vehicle $\mathcal{A}_i$) the condition (3.4) becomes:

$$((y_2^I - y_1^I) - s_1)^2 + ((x_1^I - x_2^I) - s_2)^2 \leq d_{min}^2 \tag{3.5}$$

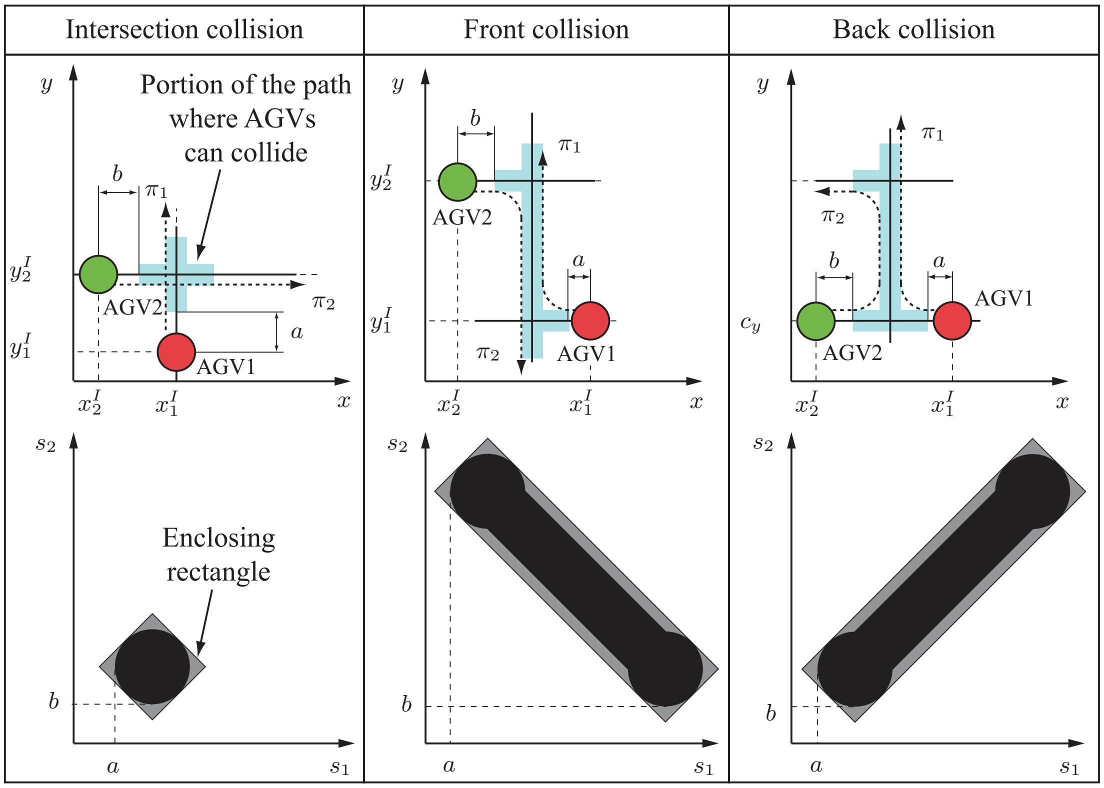


Figure 3.1: Three kinds of collision and the corresponding CDs

- *Front collision:* it takes place when the paths have some common segments and the AGVs move on them in opposite senses. Using the path's parameterizations (with notations in Fig. 3.1, where we have posed $\beta = y_2^I - y_1^I$, and where $x_1^I \equiv x_2^I$), the condition in (3.4) becomes:

$$\beta - d_{min} \leq s_1 + s_2 \leq \beta + d_{min} \tag{3.6}$$

Thus, the enclosing rectangle is defined by all points between the lines of equation
$s_2 = (\beta - d_{min}) - s_1$ and
$s_2 = (\beta + d_{min}) - s_1$.

- *Back collision:* it takes place when the paths have some common segments and the AGVs move in the same sense (see Fig. 3.1). By computations analogous to the ones for front collision, we obtain that the enclosing rectangle is defined by all points between the lines of equations $s_2 = (\beta + d_{min}) + s_1$ and $s_2 = (\beta + d_{min}) + s_1$.

3.3 Construction of the CD

In this section we propose a strategy for building the CD corresponding to N AGVs moving along paths over a predefined roadmap $\mathcal{R}$. When an AGV completes its last mission, a new one is assigned to it. This implies that the CD should be modified. The construction of the CD is, in general, a computationally demanding task [35]. In order to avoid to stop all the AGVs for computing the CD the algorithm must require a small computational effort. In [35] the path of each vehicle are considered to be not known a priori and the CD is computed online subdividing the path into straight line segment and arc of a circle. Our approach exploits the knowledge of the roadmap (recall Sec. 3.2.1) in order to split this computation in an offline phase and an online phase. This will reduce the time needed for the computation of the CD.

The offline phase is executed once after the definition of the roadmap. For each $\tau_h \in \mathcal{T}$ we define the *collision segments set* CSS_h (function DETERMINE CSS_h in Alg. 1) as

$$CSS_h = \{\tau \in \mathcal{T} \mid \exists(\alpha, \beta) \in [0,1]^2 \mathcal{A}(\tau_h(\alpha)) \cap \mathcal{A}(\tau(\beta)) \neq \emptyset\} \tag{3.7}$$

namely the set of segments that are colliding with τ_h.

For each pair of colliding segments (τ_h, τ_k) we compute and store their relative two dimensional sub-CD $s\mathcal{S}_{hk}$ using standard collision checking algorithms [41] (function COMPUTE $s\mathcal{S}_{hk}$ in Alg. 1). Furthermore, for each segment $\tau_h \in \mathcal{T}$, an empty *Booking Table* BT_h is created. This object contains the list of paths including that segment. The procedure for creating the sub-CDs is illustrated in Alg. 1.

Algorithm 1 Sub-CDs Computation

```
Require: Roadmap as a collection $\mathcal{T}$ of segments $\tau_i$
1: for all $\tau_h \in \mathcal{T}$ do
2:     DETERMINE $CSS_h$
3:     for all $\tau_k \in CSS_h$ do
4:         COMPUTE $s\mathcal{S}_{hk}$
5:     end for
6:     CREATE $BT_h$
7: end for
```

In summary, the main output of the offline phase, is a set of sub-CDs stored in a database as pieces of a puzzle that will be used in the online phase for building the global CD.

When a new path π_i is assigned to an AGV, all the $s\mathcal{S}_{ij}$ describing the collision regions between the AGV along π_i and the other vehicles must be computed. For each segment τ_k of the path, we make a reservation on its corresponding booking table; in this way we specify that the segment τ_k is contained in the path π_i. We then check the set of colliding segments CSS_k. For each colliding segment $\tau_h \in CSS_k$, we check its booking table and in case it has been booked by another path π_j, we fetch the corresponding sub-CD $s\mathcal{S}_{hk}$ from the database that we have built before. The path π_j can be on its turn split into a sequence of segments and τ_h will be part of this sequence. The decomposition of π_i and π_j into sequences of segments, induces a partition on the coordination plane. Each block of the partition is identified by a segment in the sequence of π_i and a segment of the sequence of π_j. Thus, the sub-CD $s\mathcal{S}_{hk}$ is inserted (function INSERT($s\mathcal{S}_{hk}$) in Alg. 2) in the plane $\mathcal{S}_{ij}$ of the overall CD in correspondence of the partition block that is identified by τ_h and by τ_k.

Remark 2 *The sub-CD computed in Alg. 1 are computed assuming a certain default travel direction over the two segments. During the composition phase, the real direction along which the colliding segments are crossed by the AGV is considered and the sub-CD is properly reversed before being inserted in the CD.*

In summary, the composition of the diagram is the result of picking the right piece from a database that has been defined once the roadmap has been defined. The algorithm for composing the CD is reported in Alg. 2:

3.4 Coordination planning algorithm

In this section we give an heuristic algorithm for the computation of a near-optimal solution to the coordination problem.

3.4.1 Coordination path

A *coordination path* is a continuous map $\gamma : t \to \mathcal{S}$ where $t \in [0, t_{end}]$ is the time, that defines a coordinated motion of the vehicles along their predefined paths. A path that avoids all collision regions within the CD is said to be *collision-free*. The *coordination problem* requires to define a collision-free coordination path $\gamma : [0, t_{end}] \mapsto \mathcal{S}$ whose

Algorithm 2 CD composition

```
Require: Paths currently covered by moving vehicles
Require: New path $\pi_i$
 1: for all $\tau_h \in \pi_i$ do
 2:     $BT_h \leftarrow \pi_i$
 3:     for all $\tau_k \in CSS_h$ do
 4:         if $BT_k \neq \emptyset$ then
 5:             for all $\pi_j \in BT_k$ do
 6:                 $\mathcal{S}_{ij} \leftarrow$Insert$(s\mathcal{S}_{hk})$
 7:             end for
 8:         end if
 9:     end for
10: end for
```

components $s_i(t)$ define a motion plan for each vehicle $\mathcal{A}_i$ so that all the paths are completed (i.e. $\gamma(t_{end}) = \mathbf{s}_G$, $t_{end} < \infty$) without mutual collisions.

Our goal is to solve the coordination problem problem by seeking to minimize the sum of the mission time of all vehicles. In [25] a mathematical programming formulation in which the task completion time (i.e. the time taken by the last vehicle to reach the goal) is taken as objective function and where dynamic constraints are considered is given. In many applications, however, considering the average time-to-goal as objective function would be more appropriate since this allows minimize the number of vehicles required to serve a given plant ([42]). The minimum of such a function corresponds to a Pareto optimal coordination among the vehicles ([26]). We consider that each vehicle can not backtrack along its path and that, at each position s_i, the vehicles are able to switch instantaneously between their nominal speed and halting. The first assumption is only used in order to reduce the computational effort of the algorithm. Therefore the motion of a vehicle can be described using only a binary variable and the coordination paths are piecewise linear curves. The direction of each linear segment of $\gamma(t)$ can be defined by a vector $\mathbf{u} = (u_1, \dots, u_N)$, called *coordinated action*, where $u_i \in \{0, 1\}$ for $i = 1, \dots, N$. Each variation of the direction $\mathbf{u}$ of $\gamma(t)$ corresponds to an instantaneous change of the motion of some vehicles. We denote with n^i_{acc} the number of times that the component u_i switches its value along the path $\gamma(t)$. In other words, n^i_{acc} corresponds to the number of times that the vehicle $\mathcal{A}_i$ changes its speed while executing the coordinated motion $\gamma(t)$. Given a path $\gamma(t)$ with end points at $\mathbf{s}_I = (0, \ldots, 0)$ and $\mathbf{s}_G = (T_1, \ldots, T_N)$, we denote as t_i^* the time in which the vehicle $\mathcal{A}_i$ reaches its destination (i.e. $s_i(t_i^*) = T_i$). We propose this cost function:

$$C(\gamma) = \sum_{i=1}^{N} (t_i^* + n_{acc}^i \cdot K_{acc}) \tag{3.8}$$

where K_{acc} is a parameter that is used to penalize the paths that require many vehicles to be stopped during the execution of their path.

Since the measure of t_i^* is computed without considering the vehicle dynamics, K_{acc} can be used to approximately take into account the delay accumulated by the vehicles each time that a component of $\mathbf{u}$ switches its value (considering the same delay for a start or a stoppage).

3.4.2 Forbidden regions

The proposed algorithm exploits an explicit representation of the CD that is obtained by approximating each collision region with a convex polygonal region as shown in Fig. 3.2. The polygon corresponds to the region enclosed by a set of lines: two horizontal (a_1 and a_5), two vertical (a_2 and a_6), two parallel to the bisector of the plane (a_3 and a_4) and one orthogonal to the bisector of the plane (a_7); all tangent to the obstacle and non coincident with each other. We refer to this polygon as *forbidden region*, denoted $\mathcal{F}$. This approximation is justified by the observation that frequently, in AGV applications, the collision regions have a strip shape (see Sec. 3.2.3). Thanks to this definition a coordination path that avoids all the forbidden regions is also collision-free. The algorithm that we propose finds a collision-free path by efficiently exploiting the explicit representation of the forbidden regions.

3.4.3 Construction of the coordination path

The approach consist of building a set of piecewise linear coordination paths by iterative expansions. The algorithm terminates when the best path of the set is better than all those that can be generated. At each iteration step a path is selected from the set and one or more paths are generated by concatenating it with a set of linear segments (for each segment a new path is generated). These segments will be referred as *actions* since they represent a coordinated motion of the fleet from a configuration to another.

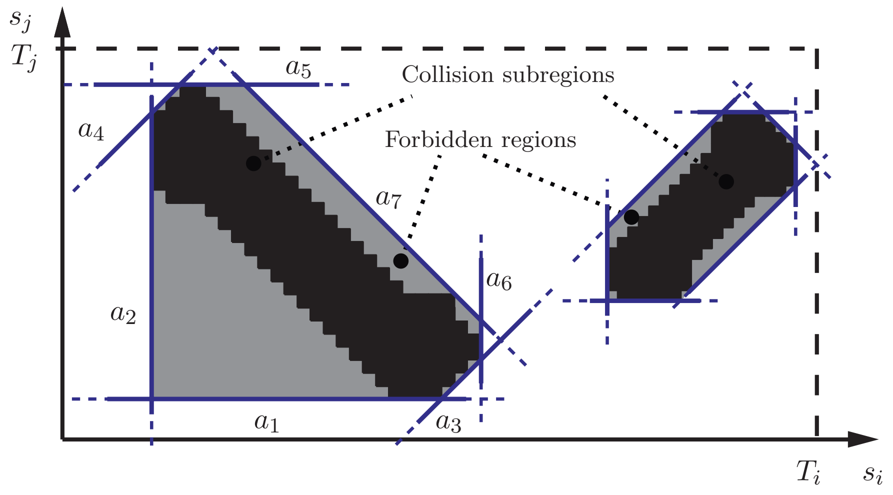


Figure 3.2: Two collision subregions and the relative forbidden regions.

This process induces a hierarchical structure over the set of paths such that each path can be defined as an extension of another one.

The extension of a path is based on the explicit representation of the CD obtained by defining the forbidden regions. For each forbidden region a set of segments, called *critical segments*, can be defined. Given a plane $\mathcal{S}_{ij}$, the set of critical segments associated with the forbidden regions of this plane is composed by (see Fig. 3.3):

- all the boundary edges of the forbidden region but the one coincident with a_7.
- two segments coincident with a_3 and a_4 outgoing from the boundary of the forbidden region towards the axes of the plane. These are interrupted when they encounter another forbidden region.
- two segments coincident with a_1 and a_2 from the forbidden region towards a_3 and a_4. These are terminated when they encounter another forbidden region.
- two segments coincident with the rays $s_i = T_i$ and $s_j = T_j$, from $\mathbf{s}_G$ towards the axes of the plane.

Loosely speaking the subset of critical segments defined in $\mathcal{S}_{ij}$ represents some configurations that are critical for the coordination of the vehicles $\mathcal{A}_i$ and $\mathcal{A}_j$. This means that when a coordination path $\gamma(t)$ reaches a critical segment on some $\mathcal{S}_{ij}$, the different alternatives for the coordination of $\mathcal{A}_i$ and $\mathcal{A}_j$ should be explored.

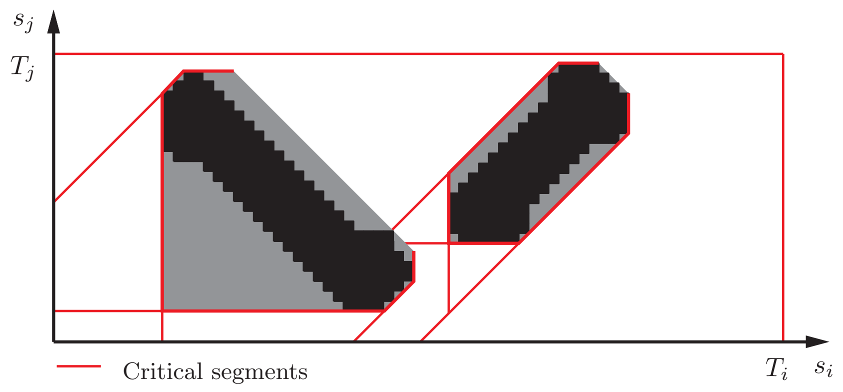


Figure 3.3: Critical segments for two forbidden regions

A coordination path $\gamma(t)$ is represented as an object η characterized by some properties (indicated by using the syntax "$\eta.Property$"). The properties are:

- $\eta.\mathbf{s}$ denotes the ending point of the path
- $\eta.P$ indicates the parent path
- $\eta.\mathbf{u}$ direction of the last action
- $\eta.d$ time duration of the last action
- $\eta.CC$ coordination components
- $\eta.ES$ extension stage

Every path has the starting point at $\mathbf{s}_I \in \mathcal{S}$ while the ending point is defined by the property $\eta.\mathbf{s} \in \mathcal{S}$. A path has zero or more children paths. The children represent the paths that are generated, during one iteration of the algorithm, by extending a given path, called the parent path ($\eta.P$). The only exception is the root path, denoted as η_R, for which no parent is defined. The root path is the null length path with which the algorithm is initialized $\eta_R.\mathbf{s} = \mathbf{s}_I$. The extension of a path η, generates a new path η' that is obtained by adding to η an action connecting the point $\eta.\mathbf{s}$ to a point $\mathbf{s}' \in \mathcal{S}$. The new path η' is the piecewise linear curve connecting the sequence of points $\eta_0.\mathbf{s}, \ldots, \eta_k.\mathbf{s}, \eta_{k+1}.\mathbf{s}$ where $\eta_0 = \eta_R$, $\eta_k = \eta$, $\eta_{k+1} = \eta'$ and $\eta_i.P = \eta_{i-1}$ for every $i \in [1, k+1]$. The point $\mathbf{s}'$ is the nearest point at which a ray outgoing from the point

$\eta.\mathbf{s}$ reaches a critical segment in some $\mathcal{S}_{ij}$. In Fig. 3.4 an example of the paths generated after two expansion steps within a two dimensional CD is reported. More formally:

$$\mathbf{s}' = \eta.\mathbf{s} + d_{\mathbf{ss}'}\mathbf{u} \tag{3.9}$$

where $\mathbf{u} = (u_1, \dots u_N)$, $u_i \in \{0, 1\}$, is the vector that defines the direction of the action with which the path is extended. This vector defines the vehicles of the fleet that have to advance in order to reach the new configuration $\mathbf{s}'$ from $\eta.\mathbf{s}$. Each component u_i defines whether a vehicle must be moving or not. The scalar value $d_{\mathbf{ss}'} > 0$ represents the time required by the fleet to reach the new configuration $\mathbf{s}'$ considering that each vehicle is traveling at its nominal velocity. This parameter is stored as a property of the path, denoted $\eta'.d$, since it will be used in the next section for the computation of the cost function. A path η can be extended by using a set of directions $\mathbf{u}$. For each direction in which the path is extended a new path η' is added as a child of η. The direction $\mathbf{u}$ along which the new path is created is stored in $\eta'.\mathbf{u}$. The i-th component of $\eta.\mathbf{u}$ is referred as $(\eta.\mathbf{u})_i$. All the pair of axes (i, j) such that $(\eta.\mathbf{u})_i = 1$ or $(\eta.\mathbf{u})_j = 1$ and where the new point $\eta'.\mathbf{s}$ is coincident with a critical segment defined in $\mathcal{S}_{ij}$ are called *coordination components*. These axes are stored in $\eta'.CC$. Note that the point $\eta'.\mathbf{s}$ may lie on a critical segment in more than one plane, thus the cardinality of set is $2 \leq |\eta.CC| \leq N$. The set of directions $\mathbf{u}$ along which a path can be extended is denoted with U_η. The definition of U_η takes into account the direction $\eta.\mathbf{u}$ of the last action of η and the set of coordination components $\eta.CC$. Instead of all the 2^N possible directions, the set U_η contains only the directions defined by substituting in $\eta.\mathbf{u}$ any possible assignment of the components indicated by $\eta.CC$. Since, generally, $|\eta.CC|$ is only a fraction of N, this definition of U_η leads to a drastic reduction of the search directions. When the point $\eta.\mathbf{s}$ is located on the boundary of some forbidden region, all directions in U_η that enter the forbidden region must be eliminated in order to ensure that the paths extended are all collision free. Moreover, when a path is chosen to be extended the algorithm does not consider all the directions in U_η. The extension of a path is done in different stages, called *extension stages*. This is obtained by defining a partition of U_η so that at each extension stage a different subset $\tilde{U}_\eta \subseteq U_\eta$ is used for the extension. The partition is defined by considering the sum of the components u_i of the coordination components $\eta.CC$. The extension stage of a given path, denoted as

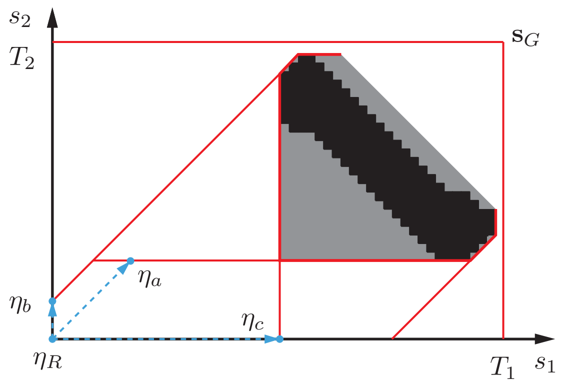


Figure 3.4: Coordination paths after two expansion steps in a two dimensional CD.

$\eta.ES$, indicates which subset of directions must be considered for the next extension of the path. Formally, the subset $\tilde{U}_\eta$ is defined as:

$$\tilde{U}_\eta = \left\{ \mathbf{u} \in U_\eta \,\middle|\, \sum_{i \in \eta.CC} u_i = |\eta.CC| - \eta.ES \right\} \tag{3.10}$$

where $|\cdot|$ denotes the cardinality of the set. Note that $\eta.ES \in \{0, 1, \ldots, |\eta.CC|\}$ thus the number of subsets defined by the partition of U_η is equal to $|\eta.CC| + 1$.

The algorithm is summarized in Alg. 3. The paths that are created at any iteration step are inserted into an ordered queue Q. The elements of Q are sorted in ascending order of the cost function defined in the next section. At the beginning of the algorithm Q is initialized with the root path η_R (lines 1-2 in Alg. 3). The algorithm then runs in a while loop (line 4) in which at each step, the first element from Q, denoted as η^*, is considered for the extension. If $\eta^*.ES = |\eta^*.CC|$, all the subset of U_{η^*} have already been evaluated, the path η^* can not be further extended and it is removed from Q (line 5). The function *DirectionSet*(η^*) (line 7) returns the subset $\tilde{U}_{\eta^*}$ of the directions corresponding to the extension stage reached by η^*. After that, the extension stage of η^* is incremented (line 8). All the directions of $\tilde{U}_{\eta^*}$ that enter any forbidden region are removed (line 9). For each direction in $\tilde{U}_{\eta^*}$, a new path (along with all its properties) is computed (lines 11-13). Then the cost function is evaluated so that the new path can be inserted in the correct position within the ordered queue Q (lines 14-15). When $\eta^*.\mathbf{s} \equiv \mathbf{s}_G$ the search terminates and the path represented by η^* is the optimal path among all those that can be explored by the algorithm. This path is then used to specify the coordinated motion of the fleet.

Algorithm 3 Coordination path search

```
Require: Critical segments $\mathcal{R}$ and forbidden regions $\mathcal{F}$
 1: $\eta_R.\mathbf{s} = \mathbf{s}_I$; $\eta_R.P = \emptyset$; $\eta_R.\mathbf{u} = (0,\dots,0)$; $\eta_R.d = 0$; $\eta_R.CC = \{1,\dots,N\}$; $\eta_R.ES = 0$
 2: $\mathrm{Q} = \eta_R$
 3: $\eta^* = \mathit{GetFirst}(\mathrm{Q})$
 4: while $\eta^*.\mathbf{s} \neq \mathbf{s}_G$ do
 5:     if $\eta^*.ES = |\eta^*.CC|$ then $\mathrm{Q}.\mathit{Remove}(\eta^*)$
 6:     else
 7:         $\tilde{U}_{\eta^*} = \mathit{DirectionSet}(\eta^*)$
 8:         $\eta^*.ES = \eta^*.ES + 1$
 9:         $\tilde{U}_{\eta^*} = \mathit{RemoveCollDir}(\tilde{U}_{\eta^*}, \eta^*.\mathbf{s}, \mathcal{F})$
10:         for all $\mathbf{u} \in \tilde{U}$ do
11:             $\eta'.\mathbf{s} = \mathbf{s}' = \mathit{NewPoint}(\eta^*.\mathbf{s}, \mathbf{u}, \mathcal{R})$
12:             $\eta'.P = \eta^*$; $\eta'.\mathbf{u} = \mathbf{u}$; $\eta'.d = d_{\mathbf{ss}'}$; $\eta'.ES = 0$
13:             $\eta'.CC = \mathit{FindCoordAxes}(\mathbf{s}', \mathbf{u}, \mathcal{R})$
14:             Evaluate $f(\eta')$                                    ▷ See Sec. 3.5
15:             $\mathrm{Q} \leftarrow \eta'$
16:         end for
17:     end if
18:     $\eta^* = \mathit{GetFirst}(\mathrm{Q})$
19: end while
```

3.5 Heuristic cost function

In this section we describe the function used by the algorithm to determine the order in which the paths are extended. Thanks to this function the algorithm extends first the paths that are more likely to optimize the objective function defined in (3.8). When the algorithm terminates the best path among all those that can be created is returned.

Like for the A* algorithm this function, denoted $f(\eta)$, is defined as the sum of two terms. The first, denoted $g(\eta)$, is directly related to the shape of the path while the second, denoted $h(\eta)$, is an heuristic underestimate of the minimum cost of the remaining path to the goal point $\mathbf{s}_G \in \mathcal{S}$. Formally the cost function is defined by this expression:

$$f(\eta) = g(\eta) + h(\eta) \tag{3.11}$$

Recall that each path can be described as a sequence of actions (i.e. linear segments). Each new path is obtained from a previous one by adding a new action. Given a path η' the value of $g(\eta')$ is computed by adding to the cost of its parent path $\eta = \eta'.P$ the cost of the new action:

$$g(\eta') = g(\eta) + c(\eta, \eta'), \qquad g(\eta_R) = 0 \tag{3.12}$$

where $c(\eta, \eta')$ is the additional cost of the new added action. This cost is defined as:

$$c(\eta, \eta') = K_{acc} \cdot N_{acc}(\eta, \eta') + N_{run}(\eta) \cdot \eta'.d \tag{3.13}$$

where, in the first term, $N_{acc}(\eta, \eta')$ is the number of vehicles that have to change their motion at the configuration $\eta.\mathbf{s}$ while executing the coordination path η'. This quantity corresponds to the number of non null components of the vector $\eta'.\mathbf{u} - \eta.\mathbf{u}$. In the second term, $N_{run}(\eta)$ is the number of vehicles that have not reached the goal at the configuration $\eta'.\mathbf{s}$. The value $\eta'.d$ (defined in Sec. 3.4) represents the time required by the fleet to reach the configuration $\eta'.s$ from the configuration $\eta.s$.

Remark 3 *The parameter K_{acc} is the additional cost accumulated each time that a vehicle has to change its motion (Sec. 3.2.2). By inflating the value of K_{acc} it is possible to reduce the computational burden of the algorithm since only the paths that require less changes of velocity are extended. The time taken by the complete algorithm and the values chosen for the parameter K_{acc} are reported in Tab. 4.1 (see Sec. 4.2). We have adopted high values of K_{acc} so that the algorithm extends only the coordination paths that require few accelerations for the vehicles. In particular the solution found will produce a coordinated motion in which all vehicles avoid collisions by starting their motion at different times, and travel at the nominal velocity till the end of their mission.*

The heuristic estimation of the cost-to-go must be an underestimate of the actual cost in order to obtain an optimal plan ([41]). The estimate of this cost is defined as:

$$h(\eta) = \sum_{i=1}^{N} (T_i - \eta.s_i) + K_{acc} \cdot J(\eta) + 2K_{acc} \cdot P(\eta) \tag{3.14}$$

The first term is the sum of the times that each of the vehicles takes to reach the goal position if the collisions with other vehicles are ignored. The last two terms are an

underestimate of the amount of costs that will be accumulated each time that a vehicle will have to stop or start its motion. In particular $J(\eta)$ is the number of vehicles $\mathcal{A}_i$ such that $(\eta.\mathbf{u})_i = 0$ and $(\eta.\mathbf{s})_i \neq T_i$. These vehicles will have to start their motion in order to reach their goal. The term $P(\eta)$ represents the minimum number of vehicles that, from the configuration $\eta.\mathbf{s}$, will have to stop before reaching the goal. This term is multiplied by two because each time that a vehicle has to stop then it will also have to restart its motion. Consider for example the point $\eta_a.\mathbf{s}$ in Fig. 3.4. From this configuration the vehicles $\mathcal{A}_1$ and $\mathcal{A}_2$ will approach a collision if they advance simultaneously. Thus if both are advancing it can be stated that one of the two vehicles will have to stop (thus $P(\eta_a) = 1$). The value of $P(\eta)$ can be computed by solving a binary integer program (BIP). We denote with $z_1, \ldots, z_N$ the variables of the program. The value $z_i = 1$ means that the vehicle $\mathcal{A}_i$ will have to give the way to another vehicle (thus u_i will have to be set to 0) while $z_i = 0$ means that $\mathcal{A}_i$ is stopped or it does not have to give the way to other vehicles. The objective function is $\sum_{i=1}^{N} z_i$, subject to the constrains defined as follows. For each plane $\mathcal{S}_{ij}$ such that $(\eta.\mathbf{u})_i = 1$ and $(\eta.\mathbf{u})_j = 1$, if the projection of $\eta.\mathbf{s}$ onto $\mathcal{S}_{ij}$ is between the rays a_3, a_4, a_5, a_6 and a_7 (Fig. 3.2), the constraint $z_i + z_j \geq 1$ is imposed. This constraint means that, since the pair of vehicles is approaching a collision, at least one of the two vehicles will have to stop. The minimization of the objective function under these constraints gives an underestimate of the number vehicles that will have to stop in order to avoid all forbidden regions. A solver for the BIP can be found within the Optimization Toolbox of MATLAB.
Given the solution $z_1^*, \ldots, z_N^*$, $P(\eta)$ is computed as:

$$P(\eta) = \sum_{i=1}^{N} z_i^* \tag{3.15}$$

Consider the example reported in Fig. 3.4 for two vehicles in which $\eta_R.\mathbf{s} = (0,0)$, $\eta_a.\mathbf{s} = (10,10)$, $\eta_b.\mathbf{s} = (0,5)$, $\eta_c.\mathbf{s} = (30,0)$ and $\mathbf{s}_G = (60,40)$. We report the computation of the cost function for $K_{acc} = 10$. By applying (3.11)-(3.14):

- For η_R: $g(\eta_R) = 0$, $h(\eta_R) = 120$, thus $f(\eta_R) = 120$. But this path will not further extended because $\eta_R.ES = |\eta_R.CC|$, thus it will be removed from Q.
- For η_a: $g(\eta_a) = 40$, $h(\eta_a) = 100$, thus $f(\eta_a) = 140$.
- For η_b: $g(\eta_b) = 20$, $h(\eta_b) = 105$, thus $f(\eta_b) = 125$.

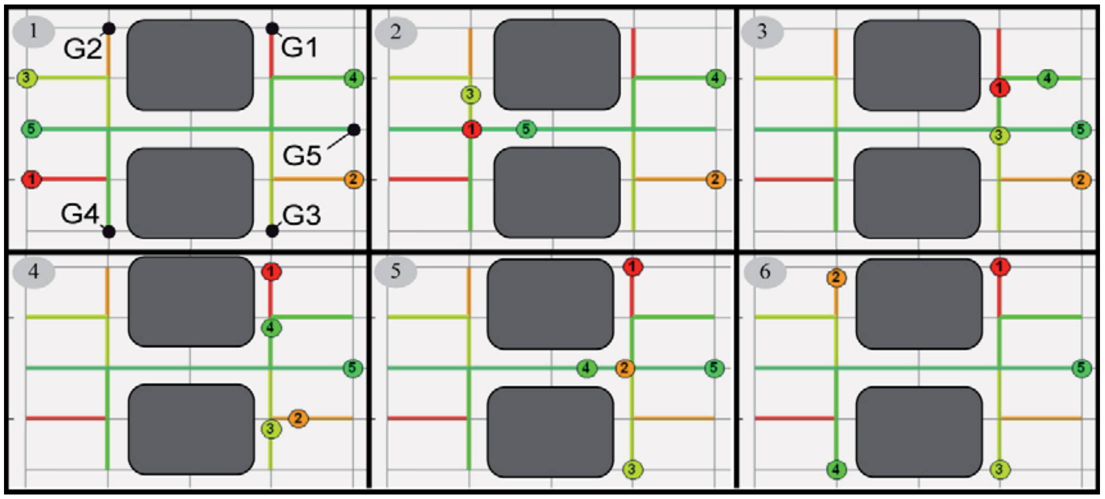


Figure 3.5: Snapshots of the simulation of the complete algorithm (test no. 5 in refsec3.5)

- For η_c: $g(\eta_c) = 70$, $h(\eta_c) = 80$, thus $f(\eta_c) = 150$.

Thus at the next step the algorithm will extend path η_b.

In Fig. 3.5 some snapshots illustrating an example of coordinated motion obtained with the proposed complete algorithm are reported. The goal positions are marked with $G1, \ldots, G5$. The vehicles are supposed to move on a Manhattan like roadmap and to have the same maximum velocity such that horizontal segments are covered in 7.5s and vertical segments in 5s. In Fig. 3.6 the planes $\mathcal{S}_{ij}$, relative to a problem instance with 5 AGVs are displayed. For each plane, the projections of the collision-free coordination path are displayed along with the set of paths explored by the algorithm.

3.6 Experiments on a real layout

We have tested our algorithm running a simulation with up to 10 vehicles in MATLAB on a Intel P8400 2.26 GHz (see Fig. 3.7). The total time required by the algorithm in order to compute the coordination path is 12.4 seconds. The vehicles are supposed to move on a roadmap used in a real industrial plant. In its real implementation this roadmap is composed both of curvilinear an straight line segments. In order to simplify the simulation program, all the curvilinear segments have been replaced by straight line segments. However, this is only a graphical approximation since the underlying coordination algorithm considers the effective trajectory of the vehicles. The nominal

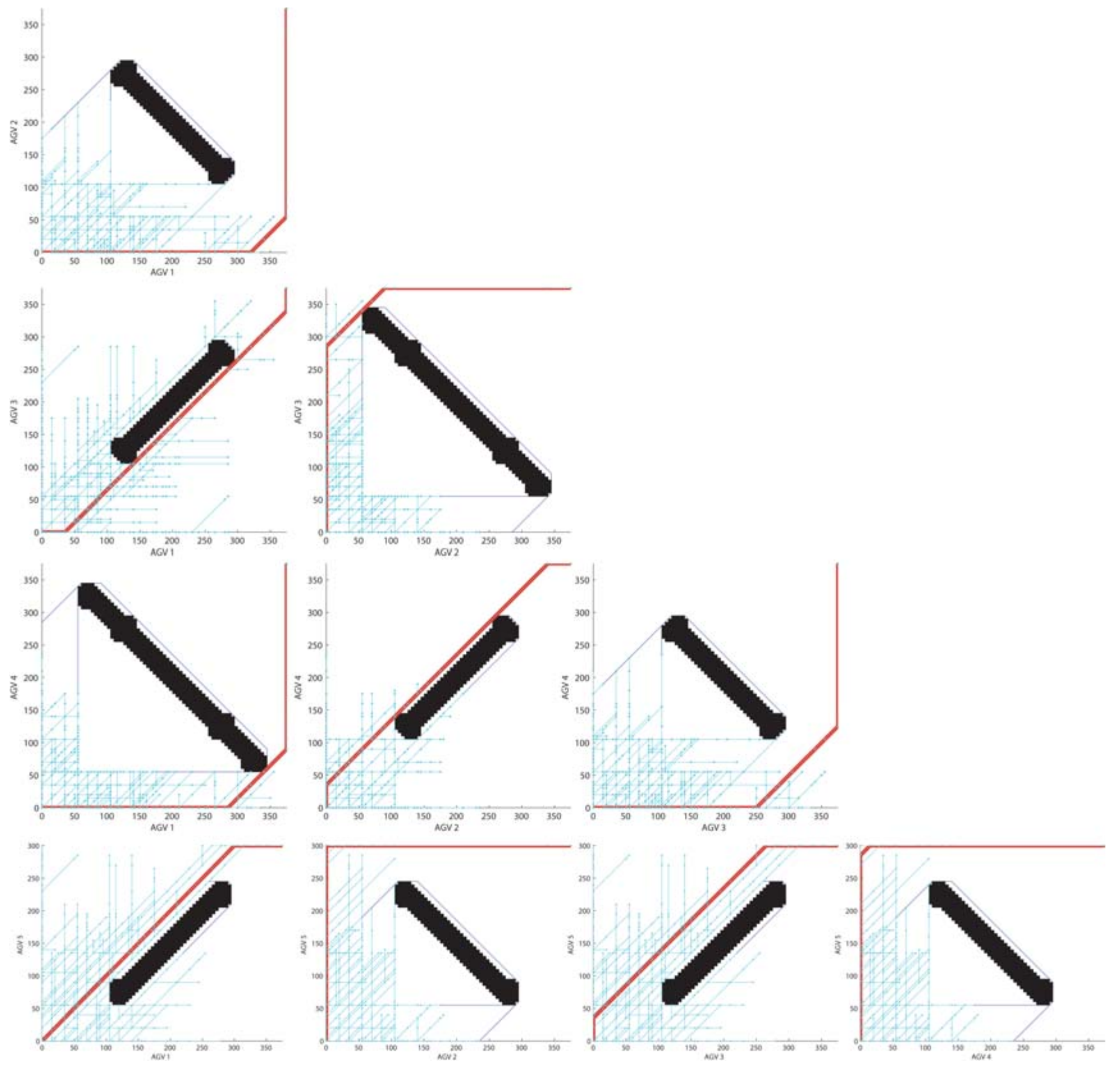


Figure 3.6: Projections of the best coordination path within $\mathcal{S}_{ij}$ (thick red line) and the paths explored by the algorithm (light gray lines). Axis units in 10^{-1}s

Table 3.1: Timing for each vehicle.

Vehicle	1	2	3	4	5	6	7	8	9	10
$T_{advance}$ [s]	144.2	118.6	115.2	127	73.4	151	123	107	121.6	136.8
T_{stop} [s]	26.8	0	1.6	0	17	0	0	1.4	0	0

speed with which a vehicle has to cover each segment is defined during the roadmap design and it can be different according to the type of vehicle and the load that it carries. The missions that are assigned to each vehicle are representative of the real working condition of the system. Each mission is defined by four way-points: initial position, pick-up position, drop off position and rest position. In Fig. 3.7 the test is illustrated. The initial, pick-up, drop-off and rest positions of the vehicles are marked with $S1, \ldots, S10$, $P1, \ldots, P10$, $D1, \ldots, D10$, $H1, \ldots, H10$ respectively. See also the attached video of the simulation test. The timings of each vehicle are reported in Tab. 3.1. For each vehicle the total time of advancement ($T_{advance}$) and the waiting time (T_{stop}) are reported. For the proposed experiment we have adopted $K_{acc} = 10$ so that the algorithm extends only the coordination paths that require few accelerations for the vehicles. In particular the solution found will produce a coordinated motion in which all vehicles avoid collisions by stopping their motion only once. Although this could produce a suboptimal solution, inflating K_{acc} allows a substantial reduction of the computation times. In Fig. 3.6 the planes $\mathcal{S}_{ij}$, relative to an experiment conducted with 5 vehicles are displayed.

3.7 Conclusions

In this chapter we have presented a complete algorithm for planning a coordinated motion of a fleet of AGVs moving along predefined paths. Using the particular structure of the considered application we have been able to considerably reduce the time required for computing the CD. We propose an algorithm that computes a complete coordination plan for the overall missions that each vehicle has to execute. The algorithm generates a set of possible coordination plans and then gives the optimal one. In real world planning problems, time for deliberation is often limited. The computational complexity of the algorithm is still too high for application on real AGVS where the paths assigned to

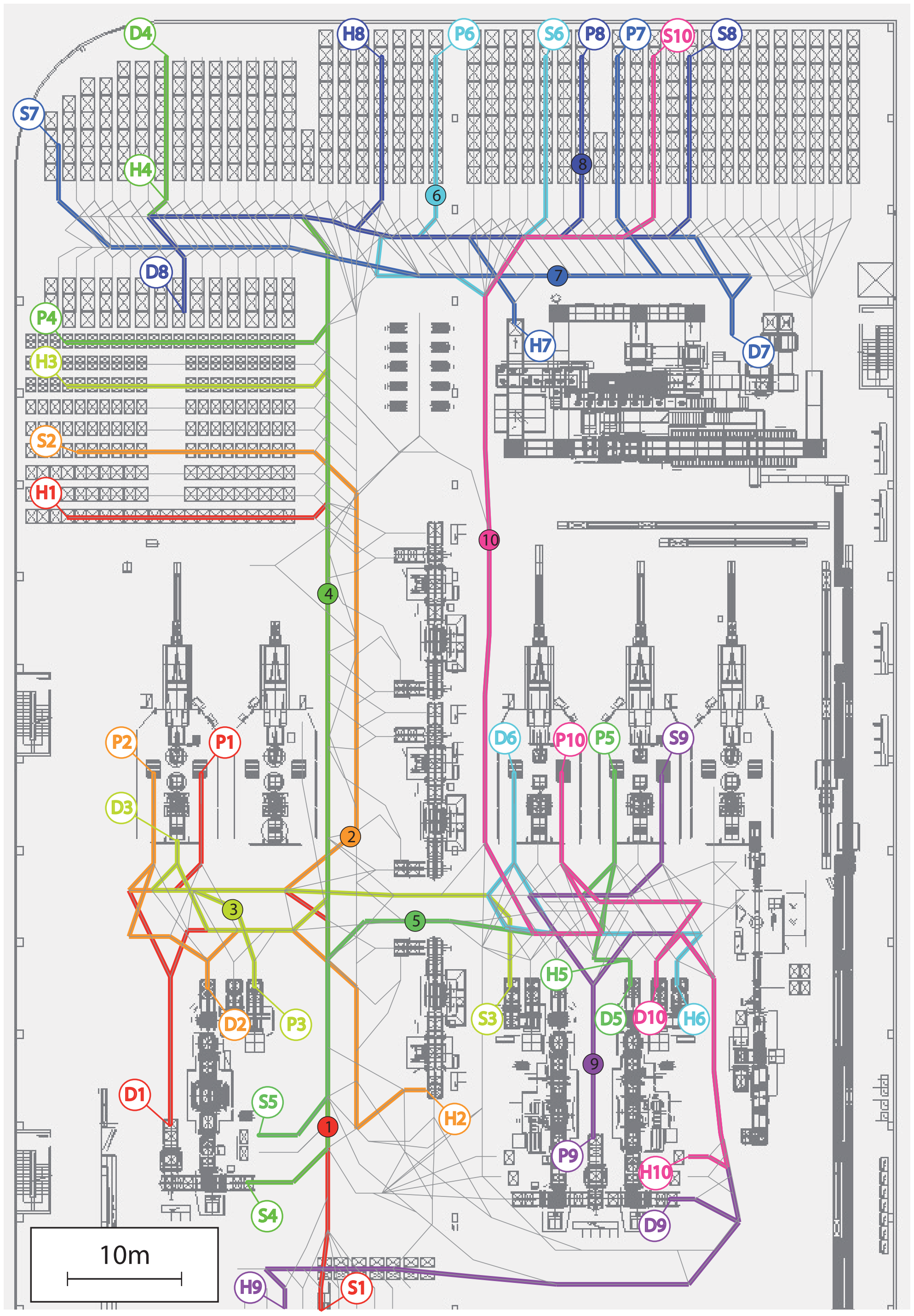


Figure 3.7: A snapshot of the simulation with 10 vehicle running in a real industrial plant.

the AGVs are frequently changed. The algorithm presented in this chapter can be applied in systems where the number of vehicles is limited and the paths that have to be executed by the vehicles are almost always the same. A possible improvement could be obtained by modifying the algorithm so that it can find a feasible solution quickly and then refine it while vehicles are in motion.

Chapter 4

Incremental coordination

In this chapter a methodology for coordinating a group of mobile vehicles following predefined paths is presented. The CD are used for representing the possible collisions among the vehicles. We exploit this information in order to define a mapping between the configuration space of the fleet and a set of motion constraints that the AGVs must satisfy in order to avoid mutual collisions. Then, a centralized and incremental planning algorithm which defines the motion of the AGVs step by step is developed. At each step, the algorithm defines the motion of the AGVs considering the actual configuration reached by the fleet. The main advantage of an incremental approach is that it allows to take into account unexpected events that can occur in an industrial environment without the need of replanning. Due to the unpredictability of the industrial environment (e.g. temporary malfunctioning of the AGVs, emergency stops), this this feature is fundamental for the considered application.

For an overview of the problem considered in this chapter, as well as for the definition of CD, the reader is addressed to Sec. 3.2. In Sec. 4.1 the incremental algorithm for coordinating a fleet of AGVs is presented and in Sec. 4.2 some simulations are developed in order to compare the proposed incremental algorithm with the complete algorithm developed in Chap. 3. Finally, in Sec. 4.3 some conclusions are drawn.

4.1 Incremental coordination algorithm

As reported in Sec. 3.2.2, in order to find a solution to the coordination problem it is sufficient to find a collision-free path with end points $\gamma(0) = \mathbf{s}_I$ and $\gamma(t_{end}) =$

$\mathbf{s}_G$. We want to design an incremental algorithm which determines the coordination action step by step (like in [23]) rather than an algorithm that defines the overall coordination strategy in one shot (like in [35]). This choice is due to the fact that in factory applications a lot of unexpected events could prevent some AGVs from performing the pre-planned action.

The problem of finding an optimal coordination path has an exponential complexity [35, 41] and, therefore, in case the number of AGVs is big, it could be necessary to stop the AGVs for a significant amount of time waiting for each new re-planning. An incremental algorithm decides which action to implement when the vehicles are at a given configuration by looking at all the possible collisions (due both to the presence of collision regions in the CD and to unexpected events) and decides the motion that the AGVs should make. Thus, the algorithm that we are proposing, allows to take into account also unexpected events without the need of re-planning each time the coordination of the vehicles.

In order to build the incremental coordination algorithm, we first need to impose a grid structure over the CD $\mathcal{S}$. Thus, for each axis, we split the interval $[0, l(\pi_i)]$ of the CD into m_i segments. This induces a grid structure over each $\mathcal{S}_{ij}$ plane. The granularity of the partition (i.e. m_i) depends on the particular application.

Remark 4 *Notice that the partition of the axis for the planning algorithm is different from that considered in Sec. 3.3 since the granularity required is, in general, different from that induced by the segments in* $\mathcal{T}$.

When moving over a path π_i, an AGV, executes an action u_i. It can either move forward to the next segment ($u_i = 1$) or move backward to the previous segment ($u_i = -1$) or remain motionless ($u_i = 0$). The role of the coordination algorithm is to tell to all the vehicles which action has to be executed. We define the *action set* as $U = \{(u_1, \ldots, u_N) \mid u_i \in \{-1, 0, 1\}\}$. The problem of choosing the right coordinated action has, in general, exponential complexity. In order to decrease the computational effort, we will not consider generic collision regions, but we will limit ourselves to the three kinds of collision regions outlined in Sec. 3.2.3.

It is clear that, locally, the best coordinated action is the one which leads to the major advancement of the fleet. Thus, the first criterion by which the actions are chosen is the maximization of overall fleet advancement. Since all vehicles can take only three actions, we can identify $2N+1$ subsets $U_\rho \subseteq U$ that contain actions which lead to the same *advancing factor* ρ:

$$U_\rho = \{(u_1, \ldots, u_N) \mid \sum_{i=1}^{N} u_i = \rho\} \tag{4.1}$$

Loosely speaking, the advancing factor, provides a measure of the advancement of the overall fleet. For example, the action subset U_{N-1} contains all the actions that make advancing all vehicles but one that remains motionless.

The algorithm evaluates, in decreasing order, the actions belonging to each subset U_ρ, starting from $\rho = N$, until it is found a subset that contains a valid action. In Alg. 4, we have denoted with Action Set(ρ) the function that gives the subset to U_ρ.

Thanks the cylindrical structure of the collision regions the algorithm can realize a coordination path considering just all the $\mathcal{S}_{ij}$. As we have seen in Sec. 3.2.3, collision regions on the two dimensional planes have a well defined shape. Thanks to this particular structure, we can define over each plane some regions that we call *shadow zones*. These regions are defined between the collision region, the axes of the plane and the two rays a_1 and a_2 tangent to obstacle borders and parallel to the bisector of the plane (see Fig. 4.1). The goal is to partition each plane in a set of zones in which some actions are forbidden since they would lead to collision or they would delay the completion time of the missions.

The algorithm has to choose an action that is allowed by all planes (a valid action). We denote by s_{ij} the point in $\mathcal{S}_{ij}$ that denotes the configuration of two AGVs considered along their paths π_i and π_j (see Sec. 3.2.1). For all possible collision region (Sec. 3.2.3) we can define (see as an example Fig. 4.1):

- *Light region:* When s_{ij} falls in this region there are no actions that have to be discarded.

- *Antumbra region:* s_{ij} falls in this region when the vehicles are approaching to collision. To escape from the shadow zone, one AGV has to advance while the other has to stop or go backward. As long as s_{ij} remains in this region, there are no action that penalize the possibility of escaping from the shadow zone. Thus, all the actions remain valid.

- *Penumbra region:* This corresponds to the situation in which one vehicle has reached a segment that belongs to the path of another vehicle. To escape from this region the following constraint between the actions has to be satisfied:

$$u_i + u_j \leq 1 \tag{4.2}$$

- *Umbra region:* This corresponds to the situation in which a couple of AGV has reached a common portion of path. Thus one of the two vehicles has to move backward. The constraints imposed by this region are:

$$u_i + u_j \leq 0 \tag{4.3}$$

- *Obstacle border:* This is the border of the collision regions. In this region, all the directions that enter into the collision region are forbidden. Referring to Fig. 4.1, the following constraints on the control actions have to be satisfied:

$$\begin{array}{ll} u_i - u_j \geq 0 & s_{ij} \in \overline{AB} \\ u_i + u_j \geq 0 & s_{ij} \in \overline{BC} \\ u_j - u_i \geq 0 & s_{ij} \in \overline{CD} \\ u_i + u_j \leq 0 & s_{ij} \in \overline{DA} \end{array} \tag{4.4}$$

For each action set U_ρ to be evaluated, the algorithm reads in which zone the coordination point projection falls (ReadRegion($\mathcal{S}_{ij}, s$) in Alg. 4) and discards all the actions of U_ρ that do not respect the constraints imposed by the region. In Alg. 4 we refer to this operation with Prune($Region, U_\rho$). Further actions are forbidden in case they imply a movement of an AGV that has to implement an emergency stop. In this way, the emergency handling is embedded online in the coordination controller. If, after pruning, all the actions contained in U_ρ are discarded, the next subset $U_{\rho-1}$ is considered. The evaluation of the action subsets terminates when, after pruning, the action set contains at least one action that satisfies the constraints induced by the shadow zones of each coordinate plane. We refer to this subset as the *valid action set*.

In general, when the evaluation stops, the valid action set contains more than one action. The final choice of the action to execute is made evaluating a cost function $\mathcal{D}(u)$. This cost is an indicator of the total time spent by the vehicles to avoid the collision with the others and it depends on the way chosen to bypass each collision region on the CD. Consider a given configuration of the vehicles and a valid action set.

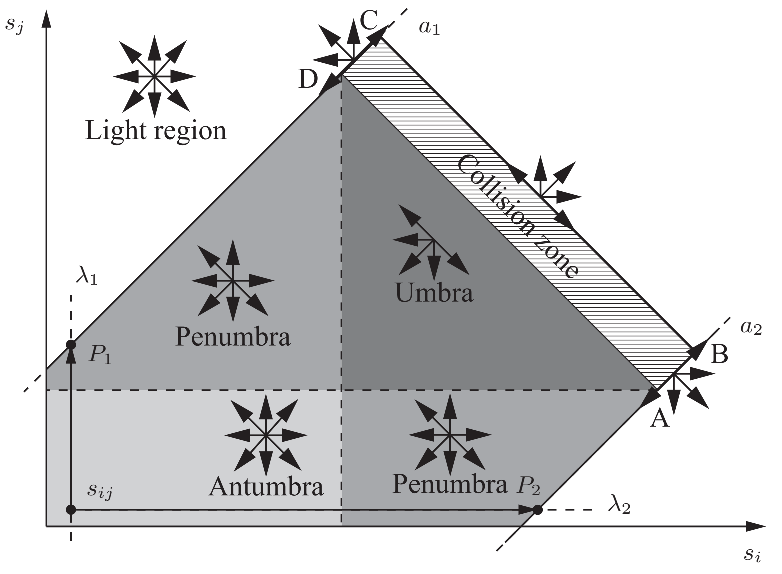


Figure 4.1: Shadow sub-zones and corresponding actions allowed

For each s_{ij} falling in a shadow zone, two points in the plane are individuated. These points, indicated as P_1 and P_2 in Fig. 4.1, are the intersection between the two half-lines λ_1 and λ_2 (outgoing from the actual position in the coordinate plane, parallel to and directed in the positive direction of the coordinate axis) and the two rays a_1 and a_2 as depicted in Fig. 4.1. The distance between s_{ij} and the points P_1 and P_2 is proportional to the time required by two AGVs for bypassing the collision region traveling in a given direction. Thus, to each action moving toward P_1 (P_2) is associated a cost equal to the distance between s_{ij} and P_1 (P_2). On the other hand, to each action moving away from P_1 and P_2 (or standing in the same position) is associated a cost equal to the sum of the distances between s_{ij} and the points P_1 and P_2. This means that actions that don't tend to resolve the collision condition are penalized with respect to those that tend to escape from the shadow zones. Finally, a cost equal to the minimum distances between s_{ij} and P_1 and between s_{ij} and P_2 is associated to actions which yield an advancement of both vehicles. This means that the choice of bypassing the obstacle is postponed but, since both AGVs are moving toward their goals, these action are not penalized. In case the coordination point in the plane doesn't fall into a shadow zone, a zero cost is associated to all actions.

These calculations in Alg. 4 are indicated by the function $\textsc{Cost}(u, \mathcal{S}_{ij})$. The total cost of the action u is defined as the sum of the costs associated to u on each coordinate plane. Once that each valid action has been assigned to a cost, the algorithm picks a minimum cost action. Thus, the chosen action u^* is given by:

$$u^* = \operatorname*{argmin}_{u \in U_\rho} \mathcal{D}(u) \tag{4.5}$$

Algorithm 4 Incremental coordinator

```
Require: Current positions of the vehicles $s_i$
 1: $\rho \leftarrow N$                                  ▷ $N$ active vehicles
 2: $U_\rho \leftarrow \emptyset$
 3: while $U_\rho = \emptyset$ do
 4:     $U_\rho \leftarrow$ ACTION SET($\rho$)
 5:     for all $\mathcal{S}_{ij}$ do
 6:         $Region \leftarrow$ READREGION($\mathcal{S}_{ij}, s$)
 7:         $U_\rho \leftarrow$ PRUNE($Region, U_\rho$)
 8:     end for
 9:     $\rho \leftarrow \rho - 1$
10: end while
11: for all $\mathcal{S}_{ij}$ do
12:     if $Region = Shadow$ then
13:         for all $u \in U_\rho$ do
14:             $\mathcal{D}(u) \leftarrow \mathcal{D}(u) +$ COST($u, \mathcal{S}_{ij}$)
15:         end for
16:     end if
17: end for
18: $u^* \leftarrow$ ARGMIN($\mathcal{D}(u)$)
19: return $u^*$
```

In Fig. 4.2 some snapshots illustrating an example of coordinated motion obtained with the proposed incremental algorithm are reported. In the first one we have marked the goal positions $G1, \ldots, G5$. In Fig. 4.3 we show some planes of the CD in which are displayed the shadows and the projection of the coordination path computed.

4.2 Complete versus Incremental: comparative tests

Some tests have been developed in order to compare the incremental algorithm proposed in this chapter with the complete algorithm presented in Chap. 3. We have tested our

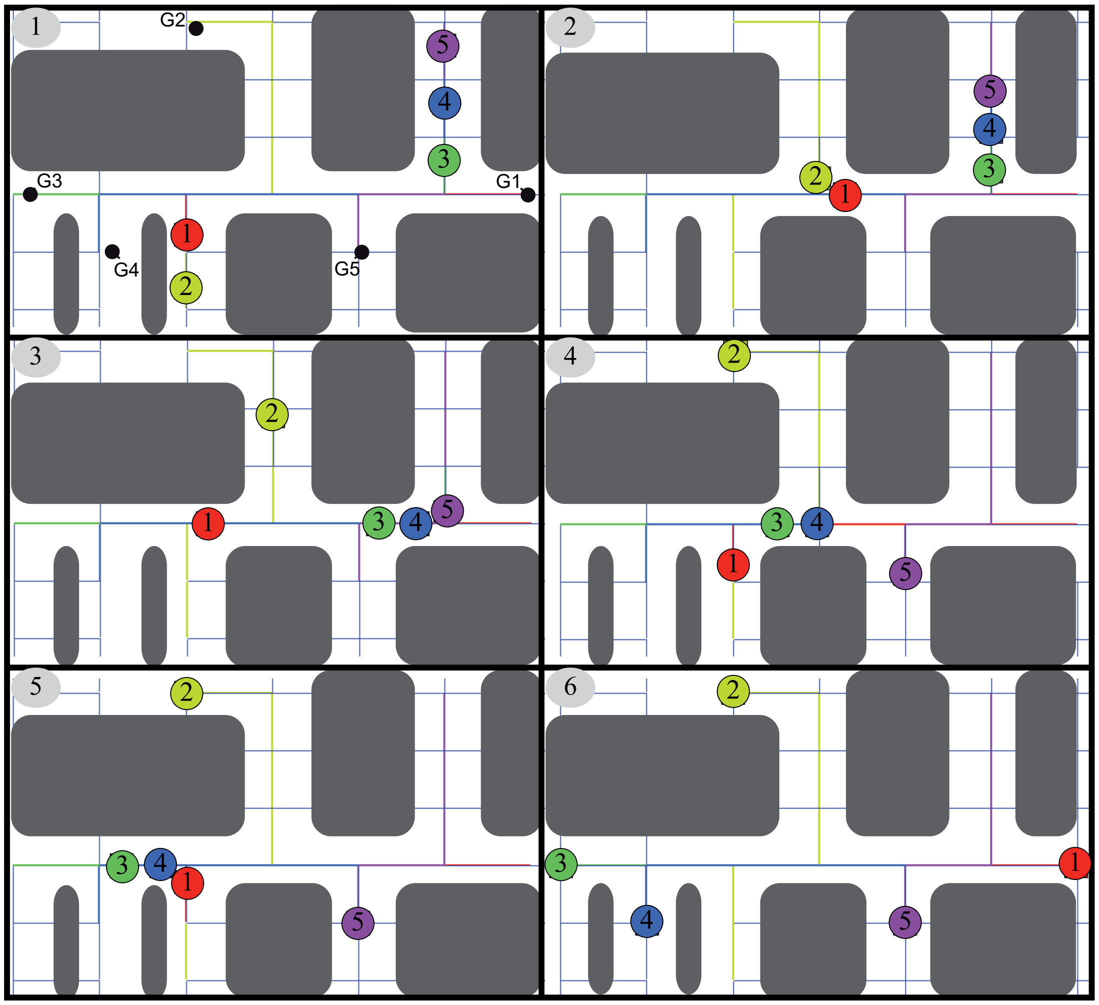


Figure 4.2: Snapshots of the simulation with the incremental algorithm

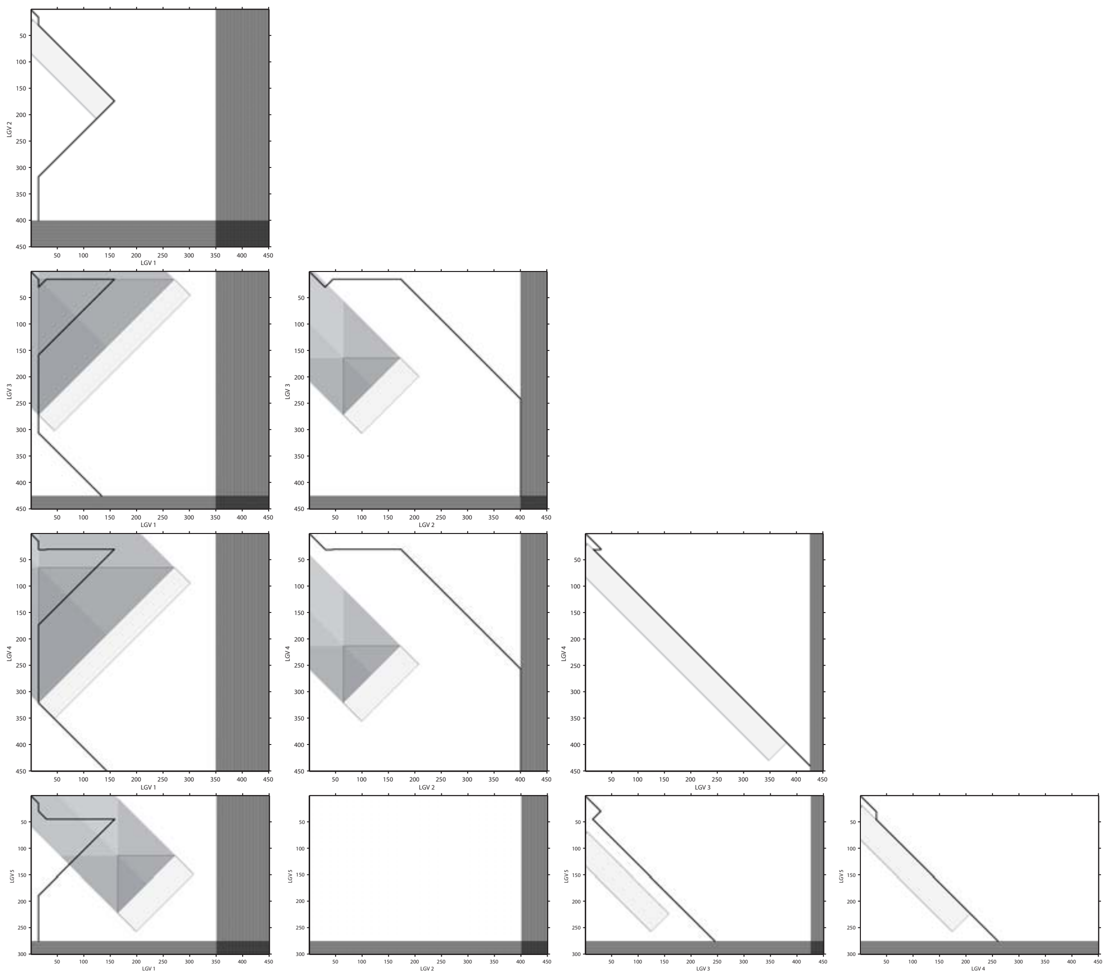


Figure 4.3: The solid black line describes the projections of the coordination path within the CDs.

Test no.	1	2	3	4	5	6	7	8	9	10	11	12
Comp. time [s]	1.1	140	13.8	14	49.9	6.3	44.3	4.6	2.7	4.3	19.4	4.4
K_{acc}	10	10	10	10	10	50	10	50	10	50	10	50

Table 4.1: Computation time and K_{acc} values.

algorithm running a simulation with 5 vehicles in MATLAB on a Intel P8400 2.26 GHz. The vehicles are starting at random positions along given paths in order to consider the fact that the mission can be released asynchronously. In Fig. 4.5 some tests are illustrated. The vehicles are supposed to move on a Manhattan like roadmap and to have the same maximum velocity such that horizontal segments are covered in 7.5s and vertical segments in 5s.

Time factor ([42]) has been used to measure system performance in the tests:

$$F_{\mathrm{t}}(i) = \frac{t_i^*}{T_i} \tag{4.6}$$

where t_i^* is the time required by $\mathcal{A}_i$ to reach its destination, T_i is the time that is required by the AGV to reach its destination considering that it continuously travel at its nominal velocity. Thus, a small value of the time factor indicates a good coordination of the fleet. The average time factor over all vehicles from 12 simulation runs are reported in Fig. 4.4. The complete algorithm works far better than the incremental one in 6 experiments while for the other cases the performances are almost the same.

For the construction of the CD both the algorithms exploit the strategy described in Sec. 3.3. The online computation of the CD requires on average takes 260 ms.

The incremental algorithm, for the computation of a single action takes at maximum 8 ms (while the average is 2 ms). The times required by the complete algorithm for computing a coordination plan are reported in Tab. 4.1. Even if these values are far higher than the ones required by the incremental algorithm, they could be acceptable if the missions were known in advance. However, unexpected events may prevent the AGVs to execute the planned motion (e.g. a manual driven vehicle standing on the trajectory of the AGV). In this case the coordination plan has to be discarded and all the AGVs have to wait for the new plan. In the considered application, events that can block the motion of AGVs are frequent, thus an incremental algorithm is more suitable.

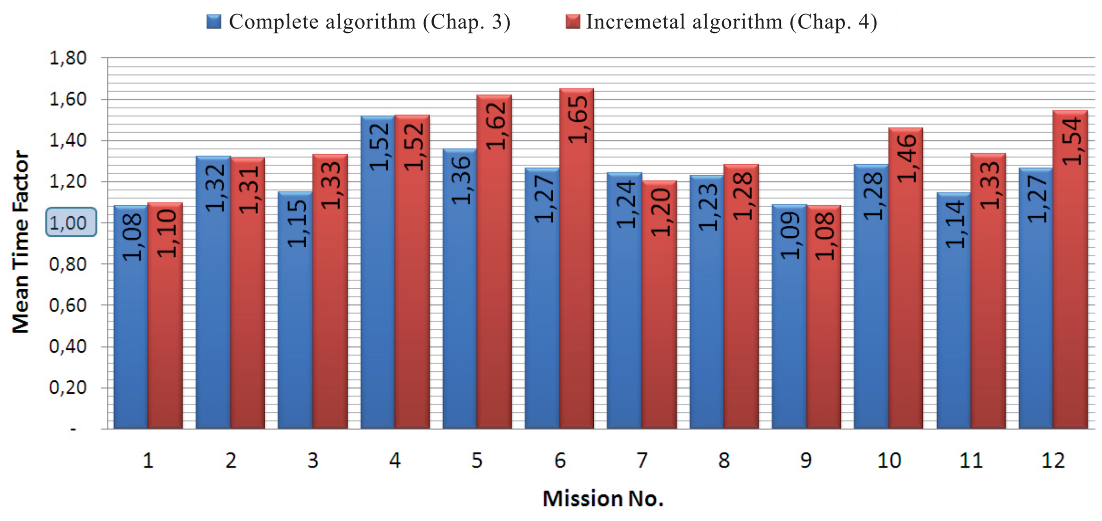


Figure 4.4: Results of the comparative tests

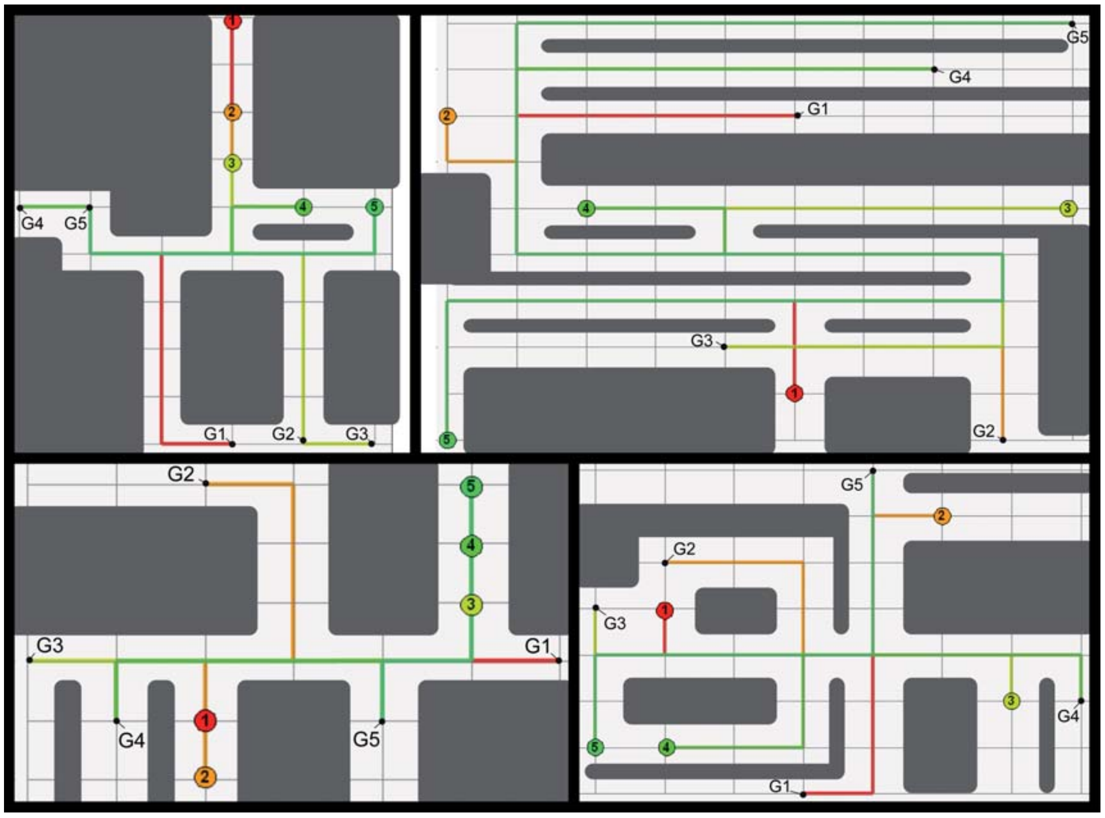


Figure 4.5: Top-left: test no. 1, Bottom-left: test no. 2, Top-right: test no. 11, Bottom-right: test no. 12.

4.3 Conclusions

In this chapter we have proposed an algorithm for coordinating multiple AGVs moving on a predefined roadmap for an industrial application. Since only some kinds of collisions can take place (see Sec. 3.2.3), we have been able to develop a incremental coordinator with a limited complexity and which can handle both collisions avoidance and emergency stops that can take place in industrial plants.

The simulation tests executed for both the complete coordinator (Chap. 3) and the incremental coordinator (presented in this chapter) show that the incremental planner has to be chosen for the considered application because of the possible unexpected events that may occur in the industrial environment.

On the experimental sides, it is necessary to implement the proposed control strategy on a real industrial setup. In order to do this, we need to relax some assumptions made since now. In particular, in the next chapter we will handle the vehicle dynamics by developing an algorithm that plans always some steps ahead with respect the current configuration.

Chapter 5

Zone control approach

In this chapter, we propose an algorithm for coordinating a group of mobile vehicles that go through predefined paths in a dynamic industrial environment. The approach is based on a centralized traffic control unit that defines the coordination of the fleet incrementally during the motion of the vehicles. The AGVs move according to the zone control approach (see Sec. 2.1). We have modified the approach presented in Chap. 4 in order develop a coordinator algorithm that can be applied within the zone control approach. A partitioned CD has been defined and the constraints introduced in Sec. 4.1 have been adapted in order to consider the new structure of the CD. Simulation tests have been executed in order to compare the performance of our algorithm with the one currently implemented by the company.

The chapter is organized as follows: in the first section the main assumptions are recalled. In Sec. 5.2 a formal definition of the problem is reported. In Sec. 5.3 the coordination algorithm is presented. In Sec. 5.4 some experiments are presented. Finally, in Sec. 5.5 some conclusions are drawn and some future work is addressed.

5.1 Introduction

Our goal is to develop an algorithm that polls the status of the AGVs while they are executing their missions, and plans the segments that each AGV is allowed to track in order to avoid collisions with other AGVs and to minimize the total time required by the fleet for reaching the rest configurations.

In the framework developed by Elettric 80 (see Chap. 2), the coordination algorithm can control the motion of vehicles only by reserving to each AGV the segments that it has to cover. However it is not possible to ensure that the segments will be covered by each vehicle at the nominal velocity. The nominal speed defined for each segment is only a setpoint for the motion controller of the vehicle. The actual speed can be different due to dynamic constraints and/or to unexpected events that could occur during the motion of the vehicle (e.g a vehicle stops to avoid a person on its way). For that reason the collisions between vehicles can be avoided only reserving at each vehicle a list of segments that are not colliding with other segments already reserved.

This chapter is an extension of the incremental approach presented in Chap. 3. The approach considers:

- The actual velocity of the AGVs can be different from its nominal value; this allows to easily take into account the effects of the dynamics as a velocity errors.
- The safety is guaranteed also if the communication with some vehicles is lost.
- Real industrial roadmaps are considered, where the dimension of the segments is pre-assigned.

The assumptions under which our algorithm is designed to work are:

- The paths must start on segments that are not colliding (see Sec. 5.2) with the segments of other paths.
- All vehicles are geometrically identical.
- The vehicles must be allowed to move in both the directions (forward and backward) over all segments.

The first two assumptions are required only to simplify the discussion of the algorithm, but they can be removed following the considerations in Remark 5 and Remark 10. The third requirement is not always satisfied in industrial plants, however it is non strictly necessary in order to make the algorithm working. The last assumption is considered in order to increase the performance of the system by giving more flexibility to motion of the vehicles.

5.2 Overview of the problem

5.2.1 Roadmap and missions

In this section we modify some of the definitions given in Sec. 3.2.1. Recall that the paths of the AGVs are defined within a collection $\mathcal{T}$ of regular curves called segments. In this chapter a *path* p_i is defined as a sequence of n_i adjacent segments and it can be represented by the following mapping

$$p_i : k \rightarrow \mathcal{T}, \quad k = 1, 2, \ldots, n_i \tag{5.1}$$

We will refer with $\mathcal{A}_i$ to the vehicle that is assigned to the path p_i. Each segment τ of the roadmap has to be covered by each vehicle $\mathcal{A}_i$ at a specified nominal speed which depends on the type of vehicle (and other factors like the payload, the priority, etc.) and on the shape of the segment. Recall that $\mathcal{A}(q)$ is the volume occupied by an AGV at the configuration $q \in \mathcal{R}$ (in order to simplify the description of the proposed algorithm we consider that all the vehicles are identical). A segment $\tau \in \mathcal{T}$ is a *colliding segment* for another segment $\tau' \in \mathcal{T}$ (and vice versa) if there exists a pair of scalars $(\alpha, \beta) \in [0, l_\tau] \times [0, l_{\tau'}]$ such that

$$\mathcal{A}(\tau(\alpha)) \cap \mathcal{A}(\tau'(\beta)) \neq \emptyset \tag{5.2}$$

This means that when two AGVs are moving through τ and τ' it can happen that a collision takes place. The concept of colliding segments can be extended by introducing a new parameter denoted *release distance*. For any pair of colliding segments (τ, τ') the release distance is a value $\gamma_{\tau,\tau'} \in [0, l_\tau]$ so that, for any $(\alpha, \beta) \in [\gamma_{\tau,\tau'}, l_\tau] \times [0, l_{\tau'}]$ the following holds

$$\mathcal{A}(\tau(\alpha)) \cap \mathcal{A}(\tau'(\beta)) = \emptyset \tag{5.3}$$

This means that if an AGV has crossed the release distance $\gamma_{\tau,\tau'}$, along the segment τ, the segment τ' can be safely covered by another vehicle (see the example in Fig. 5.1).

Remark 5 *By giving a more general definition of colliding segments, it is possible to consider also the case of a fleet composed of different types of vehicles. In practice, for each pair of segments it is possible to determine the list of vehicle shapes such that the segments are colliding.*

When the mission is released (the sequence of missions is defined as described in Chap. 2), an AGV has to execute a path, namely to reach a final configuration within the roadmap, starting from its current configuration. The path that each AGV has to track is computed without taking into account the collisions with other vehicles. Our goal is to define the motion of each vehicle along its path in order to avoid mutual collisions between AGVs.

5.2.2 Partitioned coordination diagram

For any pair of AGVs, a CD is given by $\mathcal{S}_{ij} = \left[0, T^i\right] \times \left[0, T^j\right]$. A point $(s_i, s_j) \in \mathcal{S}_{ij}$ represents a configuration of the vehicles along their paths. Recall that s_i is the time that the vehicle $\mathcal{A}_i$ would take to reach the position $\pi_i(s_i)$ considering that it travels at the nominal speed on each segment (Sec. 3.2.1).

Since each path can be defined by (5.1), the domain $\left[0, T^i\right]$ of its parameterization (3.2) can be partitioned into a sequence of adjacent intervals $\left[\Delta^i_1 \dots \Delta^i_{n_i}\right)$. Each $\Delta^i_{k_i}$ (with $k_i = 1, \dots, n_i$) corresponds to the time interval in which $\mathcal{A}_i$ is expected to travel along the segment k_i of its path if traveling at its nominal speed. When a new path is assigned to a vehicle, these intervals are computed according to the nominal speed at which each segment has to be covered by the vehicle. By picking a particular interval $\Delta^i_{k_i}$ from each path, we define a *block* as $b_{k_i,k_j} = \Delta^i_{k_i} \times \Delta^j_{k_j} \subseteq \mathcal{S}_{ij}$. A block identifies a rectangular region $\Delta^i_{k_i} \times \Delta^j_{k_j}$ of the plane $\mathcal{S}_{ij}$. The set of all blocks constitutes a partition of the CD (see Fig. 5.1).

Given two paths p_i and p_j, a block b_{k_i,k_j} corresponding to a pair of colliding segments $(p_i(k_i), p_j(k_j))$, is defined a *collision block*. By taking into account the release distances (see Sec. 5.2.1) the area of the collision blocks can be reduced. An example of $\mathcal{S}_{ij}$ is reported in Fig. 5.1 in which the collision blocks are represented with black filled rectangles.

The set of all collision blocks in a plane of the CD is called *collision region*. A collision region can be composed of many disjoint subregions (see Fig. 5.2). However, in order to simplify the description of the proposed approach we discuss this case only in the next chapter (see Sec. 6.1.3).

When an AGV completes its mission, a new one is assigned to it. An idle vehicle that is waiting at an home location (see Chap. 2) is considered to be assigned to a path composed of only the last segment covered by the vehicle before stopping.

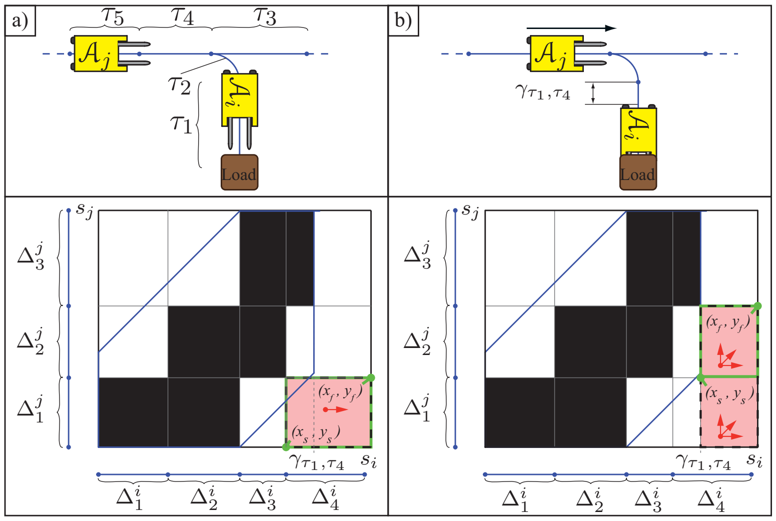


Figure 5.1: Example of motion coordination of two vehicles in which the concept of release distance is applied. The path of $\mathcal{A}_i$ is the sequence $\tau_5, \tau_4, \tau_2, \tau_1$ while the path of $\mathcal{A}_j$ is τ_5, τ_4, τ_3. In a) $\mathcal{A}_j$ can not enter the segment τ_4 because that segment is colliding with τ_1, which is currently occupied by $\mathcal{A}_i$; in b) the $\mathcal{A}_i$ is beyond the release distance γ_{τ_1,τ_4}, thus the $\mathcal{A}_j$ is allowed to cover the segment τ_4. At bottom, the CD, which is used to determine the motion constraints is reported. The grid partition of the CD, induced by the sequence of segments composing the paths is represented with gray lines. Black rectangles are the collision blocks (representing the pairs of colliding segments). The pink area represents the pair of segments which are reserved to the vehicles. In a) $x_s < \gamma_{\tau_1,\tau_4}$ because $\mathcal{A}_i$ has not already passed the release distance. In b) $x_s > \gamma_{\tau_1,\tau_4}$, thus the constraint that blocks the motion of $\mathcal{A}_j$ is removed.

This implies that the CD must be modified in order to consider the possible collisions introduced by the new path. In Sec. 3.3 we have exploited the a priori knowledge of the roadmap in order to compute offline the pairs of colliding segments. This reduces the computational cost. The definition of the CD allows to give a geometrical representation of the coordination problem. Through this representation all information about the possible collisions among the vehicles can be extracted in a way that is independent of the roadmap considered. A classification of the possible collisions that can take place between vehicles is given in Sec. 3.2.3 by using the CD.

A solution to the collision free coordination of the N AGVs is to incrementally select the sequence of segments that each vehicle has to cover in order to reach the destination. We are going to present an algorithm that controls the motion of the vehicles by searching for a path within the CD which avoids all the collisions among vehicles.

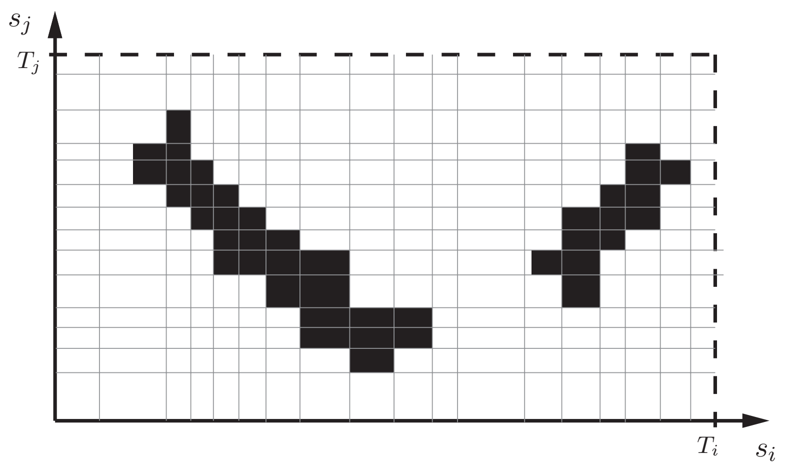


Figure 5.2: Two groups of collision blocks.

Exploiting the geometry of the CD, it is possible to associate at each configuration of the fleet a set of constraints on the actions of each vehicle such that no collisions occur.

5.3 Coordinator

The coordination problem can be solved by finding a proper path in the CD (see Chap. 3). In this section an incremental algorithm (see Alg. 5) which determines the reserved segments (i.e., the segments that the AGV is allowed to track) for each vehicle in such a way that no collisions take place is presented.

5.3.1 Segment reservation

The mapping defined in (5.1) specifies the sequence of segments that an AGV has to cover. In order to simplify the notation, the k-th segment of a path p_i, is indicated as $k_i \in \{1, \ldots, n_i\}$. Each vehicle $\mathcal{A}_i$ is associated to a list of segments, called *reserved segments*, that it is allowed to cover. The first segment of the list is the segment on which an AGV is located and it is denoted as $c_i \in \{1, \ldots, n_i\}$. The last segment is indicated with $r_i \in \{1, \ldots, n_i\}$. The current segment c_i is automatically updated each time that an AGV enters a new segment of its path. The motion of an AGV along the list of reserved segments is controlled at AGV layer (see Chap. 2). A vehicle covers all the reserved segments and stops at the end of the last. The list of reserved seg-

Algorithm 5 Coordinator algorithm

```
function COORDINATOR($\mathrm{CD}, r_1, \ldots, r_N, S$)
    $\mathcal{E} \leftarrow$ UPDATEENCLOSINGRECTANGLE($CD$)
    $(l_1, \ldots, l_N) \leftarrow (r_1, \ldots, r_N)$
    while $S \neq \emptyset$ do
        $\mathbf{u}^* \leftarrow$ ACTIONCOMPUTATION($\mathcal{E}, S, l_1, \ldots, l_N$)
        for all $i \in S$ do
            $l'_i \leftarrow$ LASTALLOCATED($u^*_i, l_i, r_i, d_i$)
            if $u^*_i \neq 0$ then
                if $\exists\ k_j \in [c_j, \ldots, l_j]$ such that $b_{l_i,k_j}$ is a collision block then
                    return $l_i$                                   ▷ Return previous value
                    $S = S \setminus i$
                end if
                if $u^*_i = 1 \wedge (l'_i = n_i \ \vee \ W_i(l'_i) > \bar{W})$ then          ▷ Not starving
                    return $l'_i$
                    $S = S \setminus i$
                else if $u^*_i = -1 \wedge l_i = 0$ then
                    return $l_i$
                    $S = S \setminus i$
                end if
            else                                                   ▷ If no segments are reserved
                return $l'_i$
                $S = S \setminus i$
            end if
        end for
    end while
end function
```

ments $[c_i, \dots, r_i]$ can be updated by the coordinator with the addition of new reserved segments.

In this chapter we consider that every AGV is allowed to cover each segment both in the same or in opposite direction with respect to the nominal orientation of the segment. The motion direction of $\mathcal{A}_i$ is denoted with the binary variable d_i. If $d_i = 1$ the vehicle covers the segments of the path in their nominal direction (i.e. toward the destination), if $d_i = 0$ the vehicle moves in opposite direction. The sequence of reserved segments has to be consistent with the motion direction of the vehicle, i.e. r_i has to greater or equal than c_i when $d_i = 1$, and less or equal otherwise.

The width of the interval $\Delta^i_{k_i}$ (i.e. the nominal time required by $\mathcal{A}_i$ to cover the segment k_i) is denoted with $w^i_{k_i}$. The time required by an AGV in order to reach the segment $l_i \in [1, \dots, n_i]$ is given by

$$W_i(l_i) = -\chi_i w^i_{c_i} + \sum_{k_i=c_i}^{l_i} w^i_{k_i} \tag{5.4}$$

where $\chi_i \in [0, 1]$ is the ratio between the distance covered by $\mathcal{A}_i$ on current segment and the total length of that segment. The coordinator algorithm is executed periodically, with a time period of $\delta = 0.5s$. Each time it updates the lists of reserved segments so that the AGVs can advance without colliding each other. If the time $W_i(r_i)$ (i.e. the time required by $\mathcal{A}_i$ in order to reach the last reserved segment r_i) is less than a value $\bar{W}$ (a parameter which has been fixed to 5s) the AGV $\mathcal{A}_i$ is defined *starving*. This condition indicates that a vehicle will reach the end of the segment r_i before the next execution of the algorithm and thus it will stop unless new segments are added to the reserved list. At each time period the algorithm starts a while loop (see line 4 in Alg. 5) in which, for each starving vehicle, the last reserved segment r_i is updated to a new value $l_i \in [1, \dots, n_i]$. A vehicle which still is starving after the execution of coordinator (e.g. because a segment $l_i = r_i$ is returned) will stop before the next execution of the algorithm (or it will remain stationary if already stopped). During the iterations of the loop the algorithm updates the segments l_i corresponding to a subset S of vehicles. At the first iteration the set S contains the starving vehicles. The condition $W_i(l'_i) \geq \bar{W}$ (see line 13 in Alg. 5) indicates that the AGV $\mathcal{A}_i$ is no more starving. In this case, as well as the case in which an AGV has reached the last segment of its path ($l'_i = n_i$), the vehicle is removed from S and the new last reserved segment l'_i is returned. An

Algorithm 6 Update last allocated segment

```
1:  function LastAllocated($u_i^*, l_i, r_i, d_i$)
2:      if $u_i^* = 0$ then
3:          $l_i = r_i$; return $(l_i, d_i)$
4:      end if
5:      if $l_i \neq r_i$ then
6:          $l_i = l_i + u_i$
7:      else
8:          if $(u_i = 1 \wedge d_i = 1) \vee (u_i = -1 \wedge d_i = 0)$ then
9:              $l_i = r_i + u_i$
10:         else
11:             if $u_i = 1$ then
12:                 $d_i = 1$
13:             else
14:                 $d_i = 0$
15:             end if
16:             $l_i = r_i$
17:         end if
18:     end if
19:     return $l_i$
20: end function
```

AGV that has to implement an emergency stop is automatically removed from S. In this way, the emergency handling is embedded online in the coordination controller.

At the first iteration (see while loop in Alg. 5), the segments l_i are initialized to r_i for each vehicle $\mathcal{A}_i$ (also those that are not in S).

At each iteration the function ActionComputation (see line 5 in Alg. 5) evaluates the l_i relative to every AGV (also those that are not in S) and returns an *action* $u_i \in \{1, 0, -1\}$ for the AGV in S. The while loop (line 4 in Alg. 5) terminates when the set S is empty. The routine LastAllocated, defined in Alg. 6, updates l_i according to the action u_i^* (see line 7 in Alg. 5). The list of reserved segments can be either in forward ($d_i = 1$) or in backward ($d_i = 0$) direction. The action $u_i^* = 0$ does not change l_i and causes the AGV $\mathcal{A}_i$ to be removed from S. The code at lines 5-18 in Alg. 6 guarantees that the segment l_i is consistent with the motion direction of the AGV. Namely, if $d_i = 1$ the segment l_i can not be less than r_i and if $d_i = 0$, l_i can not be grater than r_i.

Remark 6 *Each list of reserved segments must not contain segments that are colliding with reserved lists of other AGVs. The action chosen by the algorithm* ACTIONCOMPUTATION *(line 5 in Alg. 5) guarantees only that all the segments $l_1, \ldots, l_N$ are not colliding (pairwise). Thus, at each iteration it must be checked that the last added segment l_i, for each vehicle $\mathcal{A}_i$ in S, is not colliding with any segment $k_j \in [c_j, \ldots, l_i]$ (see line 9 in Alg. 5). If this is the case, l_i is set to its previous value and the $\mathcal{A}_i$ is removed from S. See Fig. 6.1 as an example.*

Remark 7 *At each iteration the algorithm evaluates the allocation of new segments according to the last added segment l_i of each vehicle. In order to avoid situations in which the algorithm ends up on a loop over the same blocks (i.e. livelock situations), a trace of the visited segments is stored and at each iteration the algorithm checks if the configuration it is evaluating has been already visited. If this is the case the changes applied to each l_i are discarded and the loop cycle is terminated. For seek of simplicity this function is not reported in the pseudocode.*

In Fig. 5.5 some snapshots illustrating the coordination of two vehicles are reported as a running example.

5.3.2 Action computation: forward, backward or stop

This section describes the algorithm ACTIONCOMPUTATION executed at each iteration of the COORDINATOR algorithm (see line 5 in Alg. 5) for defining how to update the lists of reserved segments. Each vehicle is allowed to move forward, backward or stop. For any pair of AGVs in which at least one of the two is in S, this function evaluates the position of the block b_{l_i,l_j}, called *reference block*, with respect the collision regions of the CD $\mathcal{S}_{ij}$.

The set S represents the AGVs for which the number of reserved segments has to be incremented. The function ACTIONCOMPUTATION returns a vector of actions $\mathbf{u} = u_1, \ldots, u_M$ (also called *coordinated action*) that will be used by the coordinator algorithm to update the reserved segments of the AGVs in S. Even if S might in general be constituted by any subset of vehicles, we consider (to simplify the notation) that $S = \mathcal{A}_1, \ldots, \mathcal{A}_M$ $(M \leq N)$.

We consider the best coordinated action $\mathbf{u}$ as the one which leads to the major advancement of the fleet. Given M vehicles, the set of all 3^M possible vectors $\mathbf{u}$ is denoted as U. This set can be partitioned (as already seen in Sec. 4.1) into $2M+1$ subsets $U_{rho} \in U$ according to the following definition:

$$U_\rho = \left\{ (u_1, \dots, u_M) \mid \sum_{i=1}^{M} u_i = \rho \right\} \tag{5.5}$$

where ρ is called *advancing factor*. Loosely speaking, the advancing factor, provides a measure of the advancement of the starving vehicles related to the coordinated action $\mathbf{u}$.

For each $\mathcal{A}_i$ that is not in S the algorithm takes an action $u_i = 0$. The algorithm evaluates, in decreasing order, the actions belonging to each subset U_ρ, starting from $\rho = M$, until it is found a subset that contains a valid action.

Remark 8 *Note that, if the constraints imposed by the CD are too restrictive, the action computation algorithm needs to evaluate a number of action which is exponential in the number of vehicles. However, differently from Chap. 4 where all the N AGVs of the fleet are always considered, here only the subset S of M ($M \leq N$) vehicles is evaluated by the coordination algorithm. Even if it does not change complexity of the algorithm in the worst case (S contains all vehicles), this fact reduces the mean computational effort.*

The reference block b_{l_i,l_j} represents the configuration that the vehicles will reach, once that all reserved segments have been covered. When new segments have to be reserved, the position of this block is evaluated in order to avoid the possible collision regions.

All collision regions in $\mathcal{S}_{ij}$ are represented with a rectangular region having two sides parallel to the bisector of the diagram. This region has been defined as *enclosing rectangle* $\mathcal{E}$ (see Sec. 3.2.3) and it is computed at line 2 in Alg. 5. In Fig. 5.3 we have represented the enclosing rectangle with the blue lines e_1, e_2, e_3, e_4.

On each plane $\mathcal{S}_{ij}$, the position of the reference block b_{l_i,l_j} is evaluated, with respect to the enclosing rectangle $\mathcal{E}$. As a convention, if $i < j$, the position of the bottom left and top right corners of b_{l_i,l_j} are indicated respectively with the coordinates (x_s, y_s) and (x_f, y_f) (see Fig. 6.1). In the following we consider $i < j$. If $l_i > c_i$ (i.e. the AGV has more than one reserved segment), the values of x_s and x_f are respectively the lower bound and the upper bound of the interval $\Delta^i_{l_i}$. If $l_i = c_i$ (i.e. the vehicle is on its last reserved segment) the value x_f is the upper bound of the interval $\Delta^i_{c_i}$ while

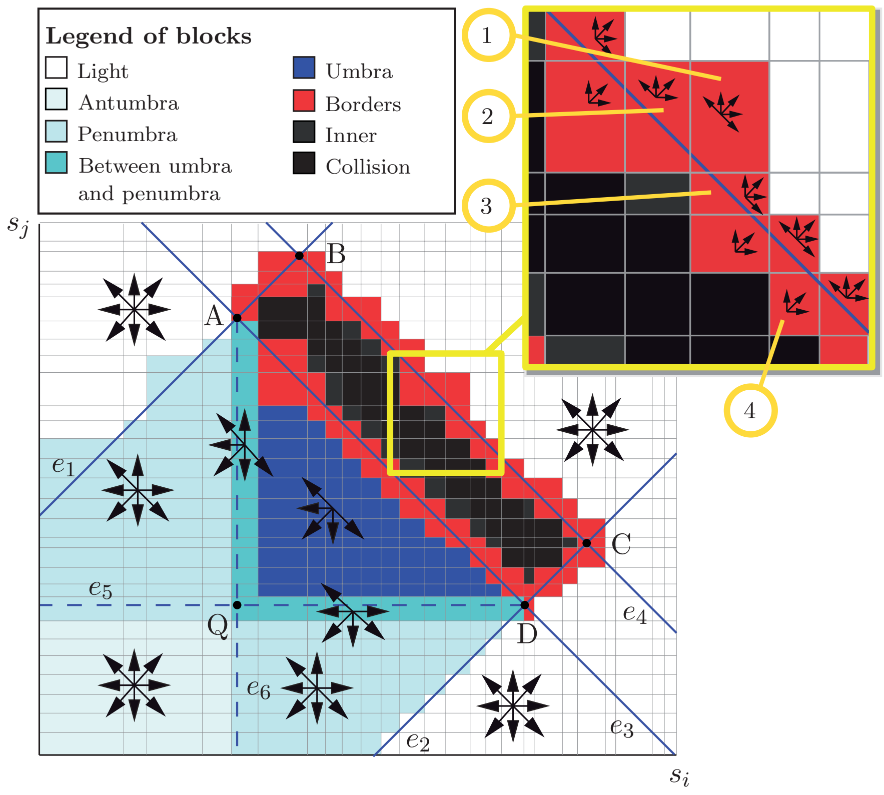


Figure 5.3: The colors denote the block grouping and the arrows the corresponding constraints. The upper right box reports a detail of the borders blocks (red filled): block 1 has only one corner inside enclosing rectangle, 2 and 3 have two corners inside, 4 has three corners.

$x_s = x'_s + \chi_i(x_f - x'_s)$ where x'_s is the lower bound of $\Delta^i_{c_i}$ and $\chi_i \in [0, 1]$ is the ratio between the distance covered by $\mathcal{A}_i$ on segment c_i and the total length of that segment. The values of y_s and y_f are computed analogously.

At each iteration the function ActionComputation (line 5 in Alg. 5) extracts the coordinates (x_s, y_s) and (x_f, y_f) and determines the position of b_{l_i,l_j} with respect to the enclosing rectangle $\mathcal{E}$.

Remark 9 *Note that, when a vehicle passes the release distance, the constraints imposed by the CD may change. Consequently, also the action chosen by the function* ActionComputation *may vary. An example about this detail is reported in Fig. 5.1.*

The blocks in which a plane $\mathcal{S}_{ij}$ is partitioned (recall Sec. 5.2) can be grouped according to their position with respect the enclosing rectangle. At each group of blocks is associated a constraint that restricts the set of actions that can be chosen by the algorithm. These associated constraints ensure that the two vehicles will reach their goals without colliding while minimizing the completion time of the two missions.

Given an enclosing rectangle (see the example Fig. 5.3) we give the definition of the groups of blocks and the associated constraints:

- *Inner blocks:* those that have all the corners inside the enclosing rectangle. Thanks to the constrains imposed b_{l_i,l_j} will never be in this group of blocks.

- *Antumbra blocks:* those that have the upper right corner inside the region enclosed by the lines e_5, e_6 and the axes of the plane. This region means that the vehicles are approaching to collision. As long as b_{l_i,l_j} is in this group, there are no actions that penalize the possibility of escaping from this zone. Thus, all the actions remain valid.

- *Penumbra blocks:* those that have the upper right corner inside one of the two regions enclosed by the lines e_1, e_5 and e_6, as well as e_2, e_5 and e_6. This corresponds to the situation in which one vehicle has reached a segment that belongs to the path of another vehicle. To escape from this region the following constraint between the actions has to be satisfied:

$$u_i + u_j \leq 1 \tag{5.6}$$

- *Umbra blocks:* those that have all the corners inside the region enclosed by the lines e_3, e_5 and e_6. This corresponds to the situation in which a couple of AGVs has reached a common portion of path. The constraints imposed by this region are:

$$u_i + u_j \leq 0 \tag{5.7}$$

- *Between umbra and penumbra blocks:* those that are traversed by the segments $\overline{QA}$ and $\overline{QD}$. We have imposed these constraints:

$$\begin{array}{ll} u_i \leq \max(-u_j, 0) & \text{for } b_{l_i,l_j} \text{ across } \overline{QA} \\ u_j \leq \max(-u_i, 0) & \text{for } b_{l_i,l_j} \text{ across } \overline{QD} \end{array} \tag{5.8}$$

- *Borders blocks:* those that have some corners inside the enclosing rectangle. The associated constraints are different according to the number of corners that are inside the enclosing rectangle: see Tab. 5.1.

Table 5.1: Border constraints

	Corners inside enclosing rectangle			
Borders	1	2 (case a)	2 (case b)	3
$\overline{AB}$	$u_j \geq u_i$	$u_j \geq \max(u_i, 0)$	$u_i \leq \min(u_j, 0)$	$\begin{cases} u_i \leq 0 \\ u_j \geq 0 \end{cases}$
$\overline{BC}$	$u_i \geq -u_j$	$u_j \geq \max(-u_i, 0)$	$u_i \geq \max(-u_j, 0)$	$\begin{cases} u_i \geq 0 \\ u_j \geq 0 \end{cases}$
$\overline{CD}$	$u_i \geq u_j$	$u_j \leq \min(u_i, 0)$	$u_i \geq \max(u_j, 0)$	$\begin{cases} u_i \geq 0 \\ u_j \leq 0 \end{cases}$
$\overline{DA}$	$u_j \leq -u_i$	$u_j \leq \min(-u_i, 0)$	$u_i \leq \min(-u_j, 0)$	$\begin{cases} u_i \leq 0 \\ u_j \leq 0 \end{cases}$

"case a" and "case b" distinguish the cases in which respectively the horizontal edge and the vertical edge is inside.

- *Light blocks:* All other blocks. There are no constrained actions.

The algorithm evaluates the block b_{l_i,l_j} on each plane $\mathcal{S}_{ij}$, identifies at which region it belongs, and discards all the $\mathbf{u} \in U_\rho$ that do not respect the constraints imposed by the region. Further actions are forbidden in case they imply a movement of an AGV that has to implement an emergency stop. In this way, the emergency handling is embedded online in the coordination controller. If all the coordinated actions contained in U_M are discarded, the next subset U_{M-1} is considered and so on. The evaluation of the action subsets terminates when is found an action set which contains at least one coordinated action that satisfies the constraints imposed. We refer to this subset as the *valid action set.*

In general, when the evaluation stops, the valid action set contains more than one coordinated action. The final choice of the action to execute is made evaluating a cost function $\mathcal{D}(\mathbf{u})$. We denote with s_{ij} the upper right corner of the block b_{l_i,l_j} (i.e. the point (x_f, y_f)). For each block having s_{ij} between lines e_1, e_2 and e_4 two points in the plane are individuated. These points, indicated as P_1 and P_2 in Fig. 5.4, are the intersection between the two half-lines λ_1 and λ_2 (outgoing from s_{ij}, parallel to and directed in the positive direction of the coordinate axis) and the two rays e_1 and e_2. Consider the distances L_i and L_j between the corner s_{ij} and respectively the points P_1

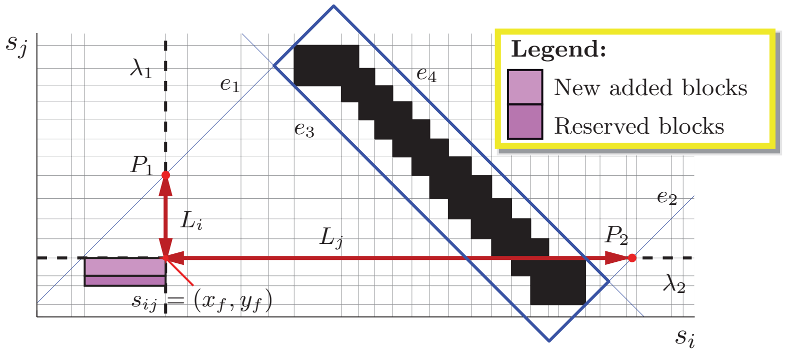


Figure 5.4: Evaluation of the distances L_i and L_j

Table 5.2: Cost computation

Actions	u_i	0	-1	-1	-1	0	1	0
	u_j	1	1	0	-1	0	$\forall$	-1
Cost	$\mathcal{D}_i(\mathbf{u})$	L_i	L_i	L_i	0	0	0	0

and P_2. Consider two vehicles at the configuration s_{ij}, L_i (or L_j) corresponds to the amount of delay (recall that the parameterization of the paths is based on time) that $\mathcal{A}_i$ (or $\mathcal{A}_j$) accumulates, due to its stop or backward motion, while the other one advances. Obviously, no delay is accumulated for the vehicle that can advance. The same delay is accumulated also if $\mathcal{A}_j$ (or $\mathcal{A}_i$) move backward, and the other remain motionless. If $u_i = u_j$, no delay is accumulated since this choice does not tend to resolve the collision. In Tab. 5.2 is reported the delay $\mathcal{D}_i(\mathbf{u})$ accumulated by $\mathcal{A}_i$ according to the coordinated action $\mathbf{u}$ considered.

For all actions $\mathbf{u} = (u_1, \ldots, u_M)$ in the valid action set, the algorithm computes for each vehicle $\mathcal{A}_i$, the maximum $\mathcal{D}_i(\mathbf{u})$ evaluated over all planes for which it is defined (recall that the distances L_i and L_j, which determine the cost $\mathcal{D}_i(\mathbf{u})$, are defined for each plane in which s_{ij} between lines e_1, e_2 and e_4). The overall cost $\mathcal{D}(\mathbf{u})$ of the coordinated action considered is the sum of all the costs $\mathcal{D}_i(\mathbf{u})$ associated to each vehicle:

$$\mathcal{D}(\mathbf{u}) = \sum_{i=1}^{M} \mathcal{D}_i(\mathbf{u}) \tag{5.9}$$

Once that each valid action has been assigned to a cost, the algorithm picks a minimum cost action $\mathbf{u}^*$:

$$\mathbf{u}^* = \operatorname*{argmin}_{\mathbf{u} \in U_\rho} \mathcal{D}(\mathbf{u}) \tag{5.10}$$

This is the action returned by Action choice. Minimizing (5.9) allows to reduce the idle time of the starving vehicles. This allows to reduce the time factor which has been proposed as a performance measure for the evaluation of an AGV system.

Remark 10 *If the initial segment of a path p_i is colliding with a segment of another path p_j, the plane $\mathcal{S}_{ij}$ has a collision block adjacent to the line $s_i = 0$. This situation can lead to a deadlock situation that can be avoided by defining some additional constraints which the valid action set must satisfy.*

5.4 Experiments

Some simulation tests have been developed in MATLAB to evaluate the improvement of the new algorithm described by this chapter. The positions of the vehicles are updated every 0.5s. A speed error in the simulated motion of the vehicles is introduced in order to take into account all possible factors that may affect the real speed of the vehicles (e.g. AGV dynamics). The vehicles are supposed to move on a roadmap used in a real industrial plant. In its real implementation this roadmap is composed both of curvilinear and straight line segments. In order to simplify the simulation software, all the curvilinear segments have been replaced by straight line segments. However, this is only a graphical approximation since the underlying coordination algorithm considers the effective trajectory of the vehicles. The nominal speed with which a vehicle has to cover each segment is defined during the roadmap design and it can be different according to the type of vehicle and the load that it carries. As described in Chap. 2, the framework required to manage an AGV system is composed of many different functionalities. The purpose of the experiments is not to evaluate how the proposed algorithm interplays with all those functionalities. Thus concerns like deadlocks which are determined by the interactions between different functions of the framework, is not considered here. Nevertheless, the transportation tasks that are assigned to each vehicle are representative of a real working condition of the system.

Time Factor ([42]) has been used to measure system performance in the tests (see (4.6) in Sec. 4.2). The average time factors over all vehicles from 29 simulation tests are reported in Fig. 5.8.

Aiming at simulating different traffic situations, we have tested our algorithm on three different parts of the overall roadmap: “Area A”, “Area B” and “Area C” (see Fig. 5.6).

The maximum number of vehicles that can be employed depends on the amount of pick/drop stations within each part. We have also performed some tests using the overall roadmap and deploying up to ten vehicles (see Fig. 5.7).

Each mission is defined by four way-points: the position of the vehicle when a new transportation task is assigned to it, the pick-up and drop off positions and the final rest position. In Fig. 5.6 and Fig. 5.7 the tests are illustrated. The starting, pick-up, drop off and rest positions are marked with $S1, \ldots, S10$, $P1, \ldots, P10$, $D1, \ldots, D10$ and $R1, \ldots, R10$ respectively.

The experiments 1-9 have been conducted with a fleet of 4 vehicles running in Area A. The missions are randomly generated and assigned to each vehicle. The aim of this group of experiments is to analyze the traffic management of the fleet when vehicles have to traverse a narrow passage. To this end, the tests consider only the motion of the vehicles between the pick-up and drop off stations which are placed at opposite sides of the narrow passage. The results are reported in the first row of Fig. 5.8. In the rest of the experiments the whole mission of each AGV is considered. The groups of the experiments 10-12, 13-17 and 18-23 have been conducted in Area B, Area C and in the overall layout respectively. The tests of each group have been defined incrementally. For example, considering the first group, test 10 considers only two missions (i.e. two vehicles), test 11 has been defined from test 10 by adding a new randomly generated mission and so on. The incremental construction of the groups of experiments is the reason for which the time factor increases with the increasing of the number of vehicles considered (see rows 2-4 of Fig. 5.8). The last group of tests (from 24 to 29) is defined by randomly generating and assigning a set of ten missions spanning the overall layout.

From this set of experiments it is clear that there is not a clear advantage of one algorithm over the other. The average overall test results put in evidence that the performance of the proposed algorithm in some cases is worse than the algorithm that has been customized by the company in order to serve the considered plant. However, we should consider that the original requires up to fifteen days of engineer work to be fine tuned. Differently, the proposed algorithm is able to coordinate vehicles on different roadmaps without the need of any additional engineering work.

As claimed in the chapter, our algorithm considers the possibility that vehicles can be blocked by some unexpected events or rerouted, due to a mission reassignment. This property has been verified by adding to our simulation software a functionality that allows to manually block a set of vehicles or to change their paths. Unfortunately the simulator used to run the algorithm of the company does not have these functionalities, thus we cannot compare the two algorithms in these situations.

5.5 Conclusions and Future Work

In this chapter we propose an algorithm for coordinating a group of AGVs developed to be implemented into the real industrial framework considered in Sec. 2.1. The main advantage of our approach is that it saves several days of engineering work each time new plant must be deployed since it is applicable to any kind of roadmap without the need of specific traffic rules. Simulation tests on a real roadmap have been executed in order to compare the performance of our algorithm with the one currently implemented by the company. The results shows that the implementation of our algorithm into the existent framework can slightly improve the performance of the system.

From a computational complexity point of view, in the worst case the proposed algorithm is still exponential (see Remark 8) in the number of AGVs. Since in real applications up to 50 AGVs have to be considered, in the next chapter a polynomial time coordinator will be presented.

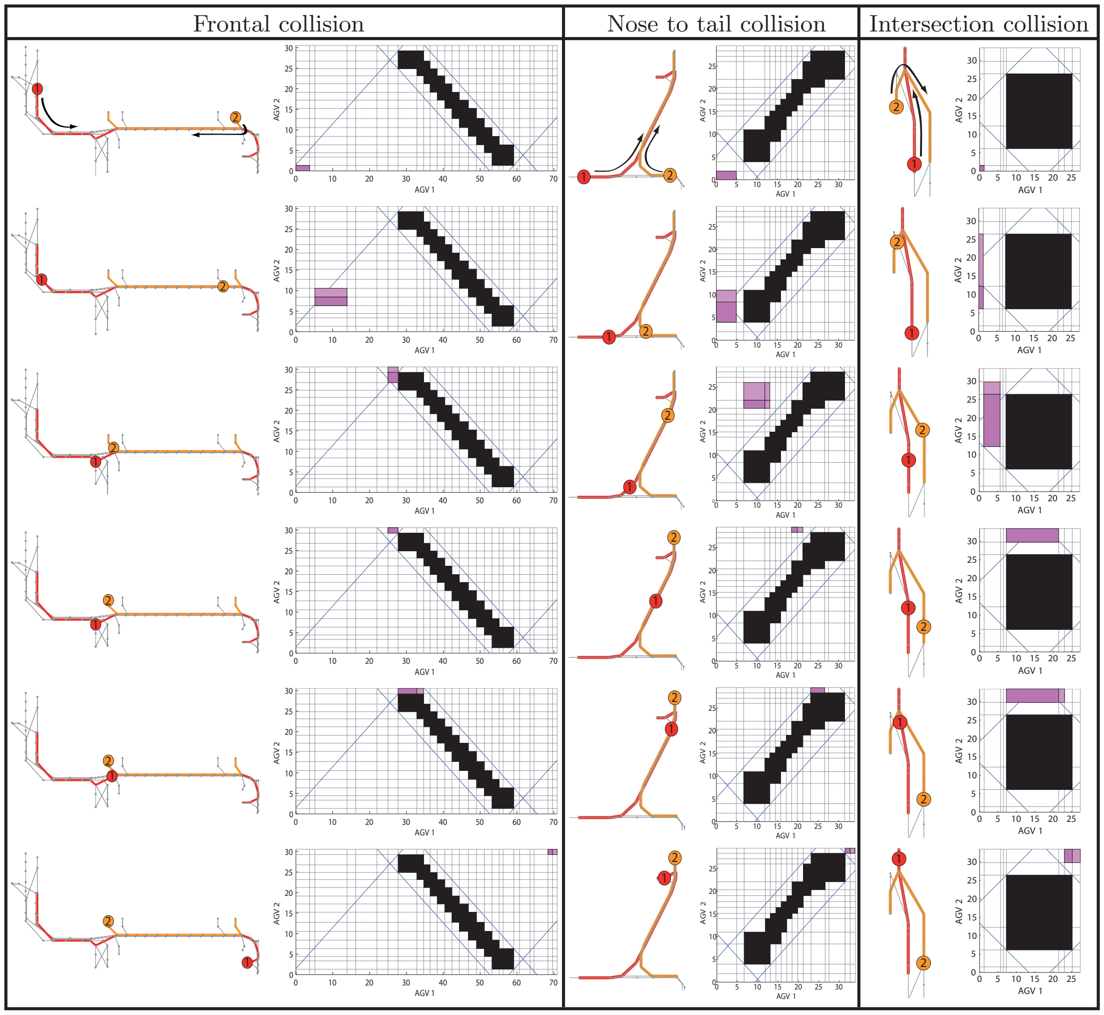


Figure 5.5: Some snapshots of three running examples and the corresponding coordination diagrams are reported. For simplicity (with M vehicle the CD would be M-dimensional) only two vehicles are employed. The first two examples illustrate a potential collision in bottleneck segments (recall that the coordination algorithm cannot change the assigned path).

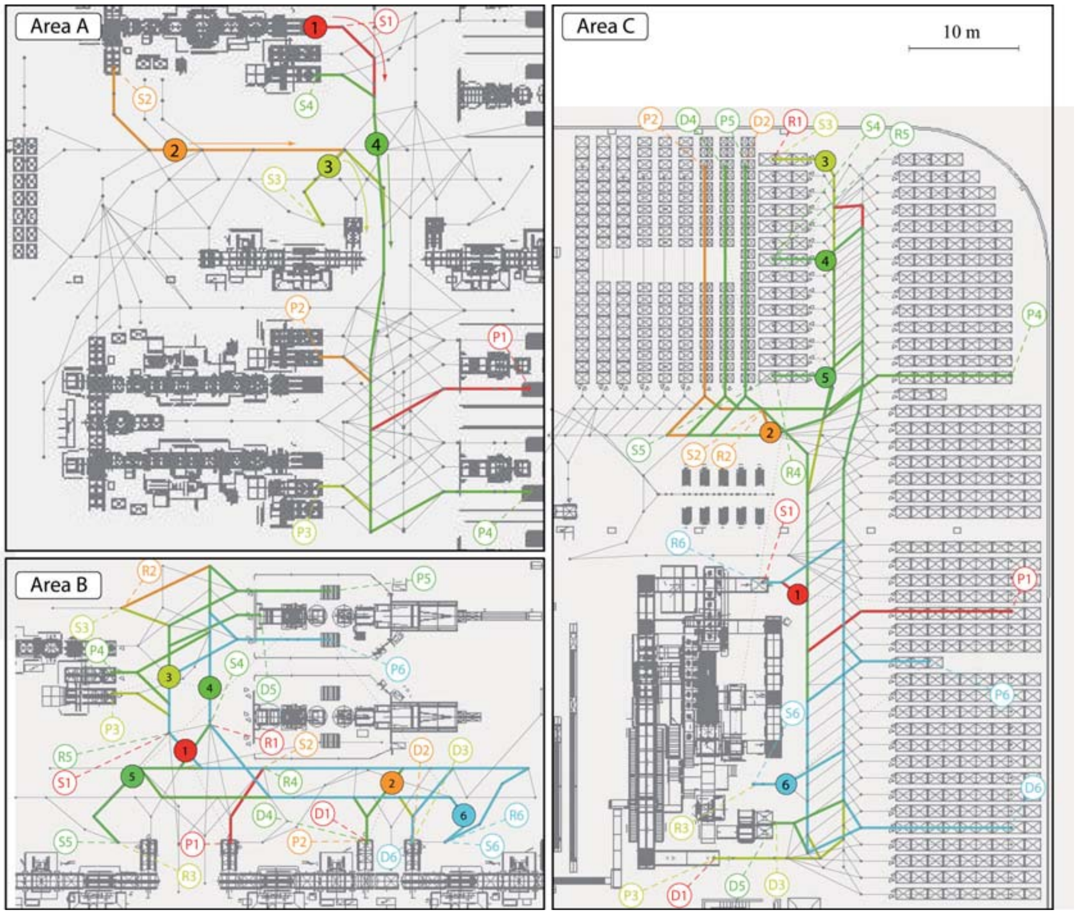


Figure 5.6: Snapshots of the experiments conducted in the three parts of the layout.

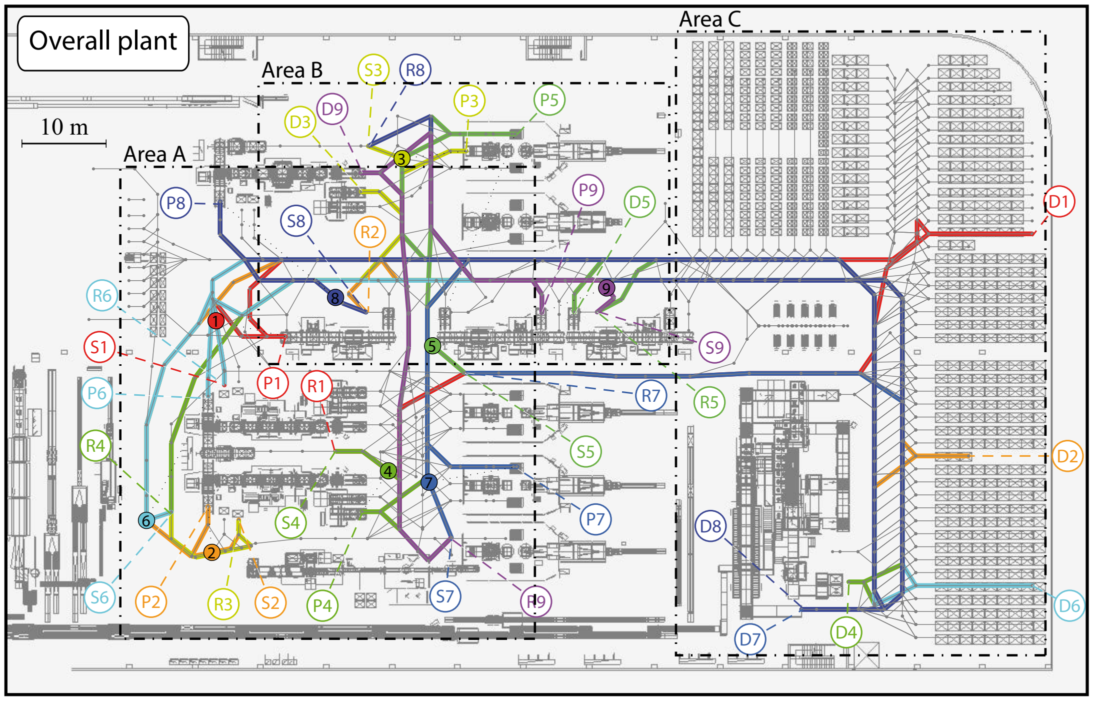
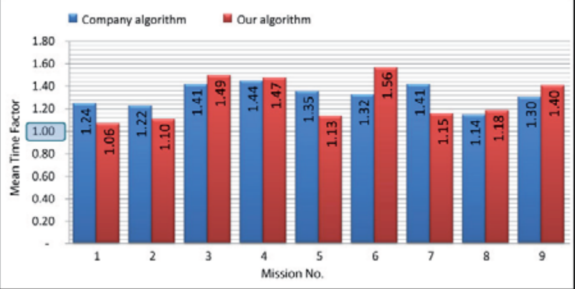


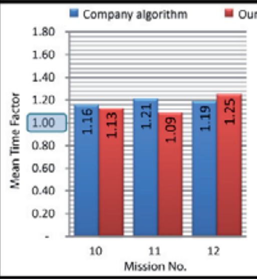
Figure 5.7: Snapshot of one experiment conducted considering the overall layout.

Area A



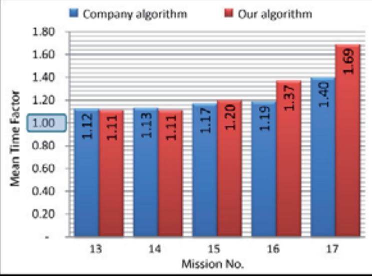
Test No	F_t comp. alg.	F_t our alg.	# of AGV
1	1.24	1.06	4
2	1.22	1.10	4
3	1.41	1.49	4
4	1.44	1.47	4
5	1.35	1.13	4
6	1.32	1.56	4
7	1.41	1.15	4
8	1.14	1.18	4
9	1.30	1.40	4
Average	1.31	1.28	

Area B



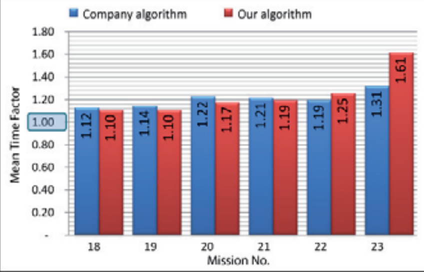
Test No	F_t comp. alg.	F_t our alg.	# of AGV
10	1.16	1.13	2
11	1.21	1.09	3
12	1.19	1.25	4
Average	1.19	1.16	

Area C



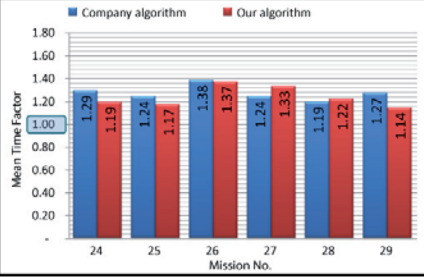
Test No	F_t comp. alg.	F_t our alg.	# of AGV
13	1.12	1.11	2
14	1.13	1.11	3
15	1.17	1.20	4
16	1.19	1.37	5
17	1.40	1.69	6
Average	1.20	1.30	

Overall layout



Test No	F_t comp. alg.	F_t our alg.	# of AGV
18	1.12	1.10	4
19	1.14	1.10	5
20	1.22	1.17	6
21	1.21	1.19	7
22	1.19	1.25	8
23	1.31	1.61	9
Average	1.20	1.24	

Overall layout



Test No	F_t comp. alg.	F_t our alg.	# of AGV
24	1.29	1.19	10
25	1.24	1.17	10
26	1.38	1.37	10
27	1.24	1.33	10
28	1.19	1.22	10
29	1.27	1.14	10
Average	1.27	1.24	

Figure 5.8: Results of the comparative tests (considering up to 10 vehicles in the overall layout).

Chapter 6

Zone controlled without backtracking

This chapter presents an improved version of the coordinator algorithm described in Chap. 5. The main improvement is the reduction of the time complexity. While for the previous solution the complexity is exponential with respect to the number of vehicles, the current coordinator has a polynomial time complexity. This improvement has been achieved mainly by applying an heuristic algorithm for the action selection problem. Differently from before, each vehicle is only allowed to wait or to move forward along its path. The coordinator computes the maximum set of AGVs that are allowed to advance while respecting the constraints.

A formal analysis of the computational time complexity is given. The performances obtained with the proposed TMS are evaluated through some tests involving up to 25 vehicles on a paths layout used in a real industrial plant.

The chapter is organized as follow. In Sec. 6.1 the coordination algorithm is presented. In Sec. 6.2 provide some simulation results. Finally, in Sec. 6.3 some conclusions are drawn and some future work are mentioned.

6.1 Coordination algorithm

6.1.1 Segment reservation

In this section an incremental algorithm (see Alg. 7) which determines the reserved segments (i.e., the segments that the AGV is allowed to track) for each vehicle in such

Algorithm 7 Coordinator algorithm

```
1:  function COORDINATOR(CD, $l_1, \dots, l_M, S$)
2:      $\mathcal{F} \leftarrow$ UPDATE FORBIDDEN REGIONS($CD$)
3:      while $S \neq \emptyset$ do
4:          $\mathbf{u}^* \leftarrow$ ACTION COMPUTATION($\mathcal{F}, S, l_1, \dots, l_M$)
5:          for all $i \in S$ do
6:              $l'_i = l_i + u^*_i$
7:              if $u^*_i = 1$ then
8:                  if $\exists\ k_j \in [c_j, \dots, l_j]$ such that $b_{l_i,k_j}$ is a collision block then
9:                      return $l_i$                              ▷ Return previous value
10:                     $S = S \setminus i$
11:                 end if
12:                 if $l'_i = n_i$ or $W_i(l'_i) > \bar{W}$ then                      ▷ Not starving
13:                     return $l'_i$
14:                     $S = S \setminus i$
15:                 end if
16:             else                                  ▷ If no segments are reserved
17:                 return $l'_i$
18:                 $S = S \setminus i$
19:             end if
20:         end for
21:     end while
22: end function
```

a way that no collisions take place is presented.

Remark 11 *In this chapter we exploit again the definitions given in Sec. 5.3.1 about the list of reserved segments and the starving condition.*

At each period the algorithm starts a while loop (see line 3 in Alg. 7) in which, for each starving vehicle, the last reserved segment l_i is updated. During the iterations of the loop the algorithm updates only the reserved segments lists corresponding to a subset S of vehicles. At the first iteration the set S contains the starving vehicles. A vehicle $\mathcal{A}_i$ is removed from S (i.e., l_i is no more updated) if $W_i(l_i) \geq \bar{W}$, which means that it is no more starving. An AGV that has to implement an emergency stop is automatically removed from S. In this way, the emergency handling is embedded online in the coordination controller. When the S becomes empty the loop terminates.

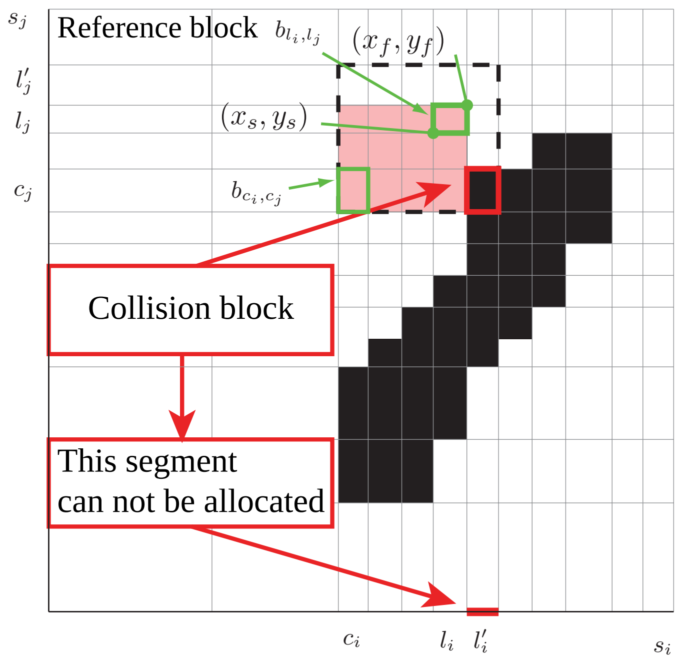


Figure 6.1: Example of the CD for two vehicles. The segment l'_i can not be added to the reserved list since the block $b_{l'_i,c_j}$ is a collision block.

At each iteration the algorithm evaluates the last reserved segment l_i of each vehicle $\mathcal{A}_i$ (also those that are not in S) in order to chose how the reserved list of each vehicle has to be updated. This computation is done by Action Computation (Alg. 8) which returns a binary *action* $u_i \in \{0, 1\}$ for each AGV. If $u_i = 1$ a new segment is added to the reserved list, i.e. l_i is incremented by 1, (see line 6 in Alg. 7). Otherwise, if $u_i = 0$ no segments are added and the $\mathcal{A}_i$ is removed from the set S. The block b_{l_i,l_j} identified by this set of segments is called *reference block*.

6.1.2 Action computation: forward or stop

In this section the algorithm for the selection of the coordinated action $\mathbf{u}$ is presented. In order to reduce the computational complexity of the algorithm, we have limited the possible actions for each vehicle to only two values (forward and stop).

The proposed algorithm exploits an explicit representation of the CD that is obtained by approximating each collision region with a convex polygonal region (line 2 in Alg. 7), as shown in Fig. 6.2. We refer to this polygon as *forbidden region* and we

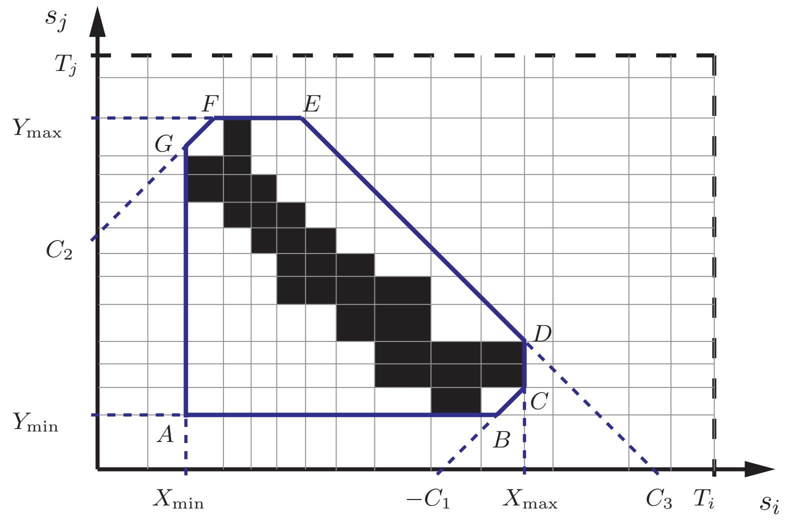


Figure 6.2: A collision region and the forbidden region with which it is approximated.

denote it with $\mathcal{F}$. The polygon corresponds to the region enclosed by seven edges: two horizontal ($\overline{AB}$ and $\overline{FE}$), two vertical ($\overline{AG}$ and $\overline{CD}$), two parallel to the bisector of the plane ($\overline{BC}$ and $\overline{GF}$) and one orthogonal to the bisector of the plane ($\overline{ED}$); all tangent to the obstacle and non coincident with each other. In particular $\overline{ED}$ is tangent to the farthest point from the origin. Of course there are some instances of the forbidden region where the segments $\overline{BC}$, $\overline{GF}$ and $\overline{ED}$ degenerate into a point (e.g., when the collision region is square). This approximation is justified by the observation that frequently, in AGV application, the collision regions have a strip shape (see Sec. 3.2.3). Since the directions of each edge are fixed, the forbidden region can be defined using seven parameters, one for each edge: $Y_{\min}$ for $\overline{AB}$, $Y_{\max}$ for $\overline{FE}$, $X_{\min}$ for $\overline{AG}$, $X_{\max}$ for $\overline{CD}$, C_1 for $\overline{BC}$, C_2 for $\overline{GF}$ and C_3 for $\overline{ED}$. Note that the definition of forbidden region is a refinement of the definition of enclosing rectangle given in Sec. 5.3.2. This allows to reduce the number of action constraints defined for a given plane of coordination diagram.

The blocks in which a plane $\mathcal{S}_{ij}$ is partitioned can be grouped according to their positions with respect to the forbidden region $\mathcal{F}$. At each iteration the algorithm evaluates at which group the reference block b_{l_i,l_j} belongs. On each plane $\mathcal{S}_{ij}$, the position of b_{l_i,l_j} is evaluated by the function Find Constraints (see Alg. 9). Alg. 9 extracts, at line 2, the coordinates (x_s, y_s) and (x_f, y_f) and determines the position of

Algorithm 8 Action Computation

```
1: function ACTION COMPUTATION($\mathcal{F}, S, l_1, \ldots, l_M$)
2:     $V = S, \quad E = \emptyset, \quad u_i = 0, \quad \forall i$
3:     for all $i \in S$ do
4:         for all $j = 1, \ldots, M$ do
5:             $\kappa =$ FIND CONSTRAINTS($l_i, l_j, \mathcal{F}$)
6:             if $\kappa =$ "$u_i + u_j \leq 1$" then
7:                 $E = E \cup (i, j)$
8:             else if  then$\kappa =$ "$u_j = 0$"
9:                 $V = V \setminus \{j\}$
10:                $u_j = 0$
11:            else if $\kappa =$ "$u_i = 0$" then
12:                $V = V \setminus \{i\}$
13:                $u_i = 0$
14:            else if $\kappa =$ "$u_i, \ uj = 0$" then
15:                $V = V \setminus \{i, j\}$
16:                $u_i = 0, \quad u_j = 0$
17:            end if
18:        end for
19:    end for
20:    $Z =$ MAXIMUM INDEPENDENT SET($V, E$)
21:    return $u_i = 1, \quad \forall i \in Z$
22: end function
```

b_{l_i,l_j} with respect to the forbidden region. The points (x_s, y_s) and (x_f, y_f) are compared with the parameters of the forbidden region in order to determine in which region the block belongs. Once that the region has been determined, the function outputs the constraint on the actions u_i and u_j of the AGVs $\mathcal{A}_i$ and $\mathcal{A}_j$ through the variable κ. At each group of blocks (also called *region*), a constraint that restricts the set of actions that can be chosen by the algorithm is associated. These constraints ensure that the two vehicles will reach their goals without colliding. Given a forbidden region (see e.g., Fig. 6.3) the groups of blocks and the associated constraints are:

- *Antumbra blocks:* As long as the reference block b_{l_i,l_j} is in this group, all the actions remain valid (i.e. $u_i, \ u_j \in \{0, 1\}$).

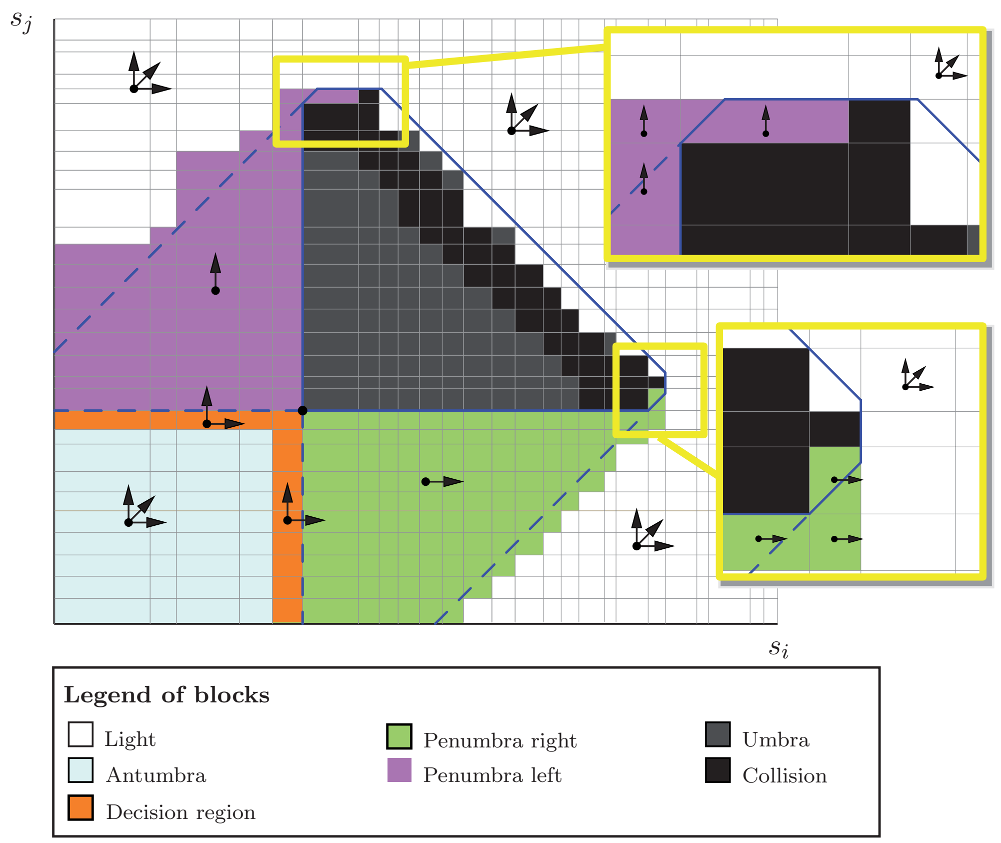


Figure 6.3: The colors denote the block grouping. For each group the actions that are allowed are represented with black arrows.

- *Penumbra blocks:* To escape from this region the following constraint between the actions has to be satisfied:

$$\begin{aligned} u_i &= 0 \ \text{ if } \ b_{l_i,l_j} \text{ in Penumbra left} \\ u_j &= 0 \ \text{ if } \ b_{l_i,l_j} \text{ in Penumbra right} \end{aligned} \tag{6.1}$$

- *Decision blocks:* When the reference block is in this region it has to be chosen which AGV has to stop in order to avoid the collision region. The following constraint is imposed:

$$u_i + u_j \leq 1 \tag{6.2}$$

- *Umbra blocks:* When b_{l_i,l_j} is in this region, no action are allowed to both the vehicles. Thanks to the constrains imposed the reference block b_{l_i,l_j} will never be in umbra group. However, a bad initial positioning of the fleet at start-up or a system failure could lead to this situation. In this case both the vehicles are stopped an error message is issued.

- *Light blocks:* All other blocks. There are no constrained actions.

For the sake of brevity, the part of the code that identifies the blocks that overlaps the forbidden region is synthesized by the function FindInside. When the reference block b_{l_i,l_j} has one corner inside the forbidden region (see the example in Fig. 6.5), this function checks the blocks of the CD which are above it (i.e. $\forall b_{k_i,l_j} k_i \in [l_i \dots n_i]$) and on right side (i.e. $\forall b_{l_i,k_j} k_j \in [l_j \dots n_j]$). If no collision blocks are found, b_{l_i,l_j} is Light and the function returns $\kappa =$ "$\emptyset$"; if a collision block is found in both sides, b_{l_i,l_j} is Inner and $\kappa =$ "$u_i,\ u_j = 0$"; if a collision block is found only in the above side, b_{l_i,l_j} is Penumbra right and $\kappa =$ "$u_j = 0$"; if a collision block is found only in the right side, b_{l_i,l_j} is Penumbra right and $\kappa =$ "$u_i = 0$".

The problem of finding a coordinated action $\mathbf{u}^*$ which maximize the advancement of the vehicles in S can be expressed as a Binary Integer Program (BIP) in which each constraint is either unary (eq. (6.1)) or binary (eq. (6.2)). The unary constraints are trivially satisfied since they simply require to put the related variable to 0 (see lines 10, 13 and 16 in Alg. 8). The remaining problem can be stated as follows.

$$\begin{array}{l} \text{Max } U = \sum u_i \\ \text{Subject to } \left\{ \begin{array}{l} u_i + u_j \leq 1 \ \ \forall (i,j) \in E \\ u_i \in \{0,1\} \ \ \forall i \in V \end{array} \right. \end{array} \tag{6.3}$$

Where V is the set of actions u_i that are not constrained by unary constraints and E are the pairs of actions that are constrained by binary constraints. This particular BIP can be represented as a undirected graph $G = (V, E)$ where V is the set of vertices and the E is the set of edges. This fact allows to transform this optimization problem into a *Maximum Independent Set* (MIS) problem. Given an undirected graph, the MIS problem consist of finding a maximum-cardinality subset Z of vertices such that no two vertices in Z have an edge between them. By solving the MIS for the graph $G = (V, E)$, the solution Z indicates which variables have to be set to 1 in order to obtain a solution for the problem in (6.3) (see [43]). In other words, the set Z corresponds to the maximum set of AGVs that can advance without violating any of the constraints.

In order to find the set Z, the heuristic procedure developed in [43] is applied (see line 20 in Alg. 8). This algorithm incrementally removes the vertices which are not within Z. At each iteration a vertex is selected and removed from the graph along with all its incident edges. The algorithm is based on two characteristics of the vertex: the *degree*, which is the number of neighbors of the vertex, and the *support*, which is the sum of the degree of the neighbors of the vertex. Among the vertices which maximize the support, the one with the minimum degree is selected. The process terminates when all edges of the graph have been removed. The set of remaining vertices is Z. Note that, for our application, it is likely that there could be tie situations where more than one vertex exists with the same maximum support and minimum degree. In this case, one of these vertices can be indifferently selected. This fact can be exploited in order to further reduce the time that each vehicle has to stop for the traffic. The delay accumulated by a vehicle in order to avoid a collision with another one can be evaluated by using the CD. When a pair of vehicles ($\mathcal{A}_i$ and $\mathcal{A}_j$) is approaching to collision, the corresponding block b_{l_i,l_j} is in Antumbra or in Decision region within the plane $\mathcal{S}_{ij}$ (see e.g., Fig. 6.4). We denote with s_{ij} the upper right corner of the block b_{l_i,l_j} (i.e., the point with coordinates defined by the right limit of the intervals $\Delta^i_{l_i}$ and $\Delta^j_{l_j}$) and with e_1 and e_2 respectively the lines coincident with the borders $\overline{BC}$ and $\overline{GF}$ of the forbidden region. For each block in Antumbra or in Decision region for which s_{ij} is between lines e_1, e_2 two points in the plane are individuated. These points, indicated as P_1 and P_2 in Fig. 6.4, are at the intersections between the two rays λ_1 and λ_2 (outgoing from s_{ij}, parallel to and directed in the positive direction of the coordinate axis) and e_1 and e_2. The distances between s_{ij} and the points P_1 and P_2, are denoted respectively with L_j and L_i. Consider two vehicles at the configuration s_{ij}, L_i (or L_j) corresponds to the amount of delay (recall that the parameterization of the paths is based on time) that $\mathcal{A}_i$ (or $\mathcal{A}_j$) accumulates, due to its stop, while the other one advances. The maximum delay which can be accumulated by each vehicle due to its stop can be determined by considering all planes of the CD. We have modified the algorithm in [43] so that, when a tie situation occurs, the vertex corresponding to the vehicle with the minimum value of delay is selected from among all candidates (recall that the vertices selected during the iterations of the algorithm are those corresponding to the actions that will be set to 0).

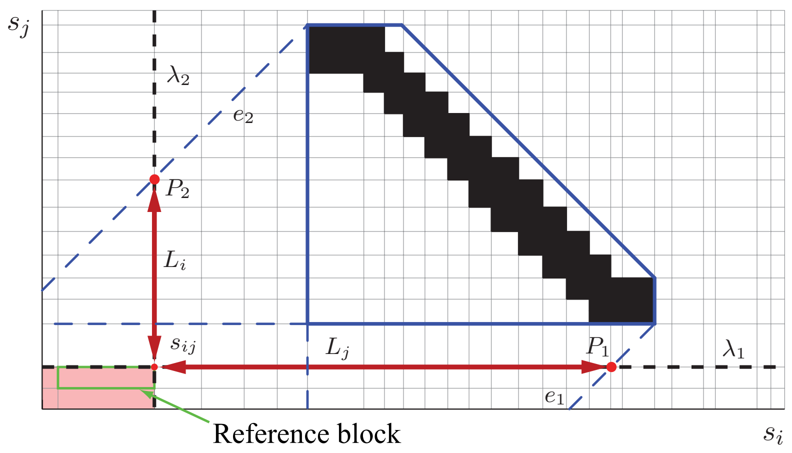


Figure 6.4: Evaluation of the distances L_i and L_j

6.1.3 Multiple collision regions on a single plane

The set of collision blocks on a plane $\mathcal{S}_{ij}$ can be composed of many disjoint components (see e.g., Fig. 6.2). We refer to these separated components as collision regions. In order to manage this case, some features must be added to the algorithm. First of all, a forbidden region is defined for each collision region. When two forbidden regions within the same plane are overlapping, a new forbidden region which encloses both the collision regions is defined (see Fig. 6.5).

Each forbidden region that is defined for a plane of the CD induces a different blocks partitioning and thus many different constraints can be associated to the same block b_{l_i,l_j}. The function Action Computation runs the Find Constraints routine for each forbidden region and then the most restrictive constraint is retained. If the constraints imposed by two forbidden regions are respectively $u_i = 0$ and $u_j = 0$ the algorithm checks the neighboring blocks of b_{l_i,l_j} in order to determine which constraint must be considered: if b_{l_i+1,l_j} is a collision block, the constraint is $u_i = 0$, if b_{l_i,l_j+1} is a collision block, the constraint is $u_j = 0$, otherwise the constraint of the region that is closer to the origin of the diagram is considered. The case in which both b_{l_i+1,l_j} and b_{l_i,l_j+1} are colliding is not considered since in this case the blocks are enclosed by the same forbidden region. The delay evaluated in order to brake the ties of the MIS solver can be computed almost in the same way described in Sec. 6.1.2. On each plane, the values L_1 and L_2 are computed for each forbidden region with respect to which the block b_{l_i,l_j} is in Antumbra or in Decision region and the the maximum value is retained.

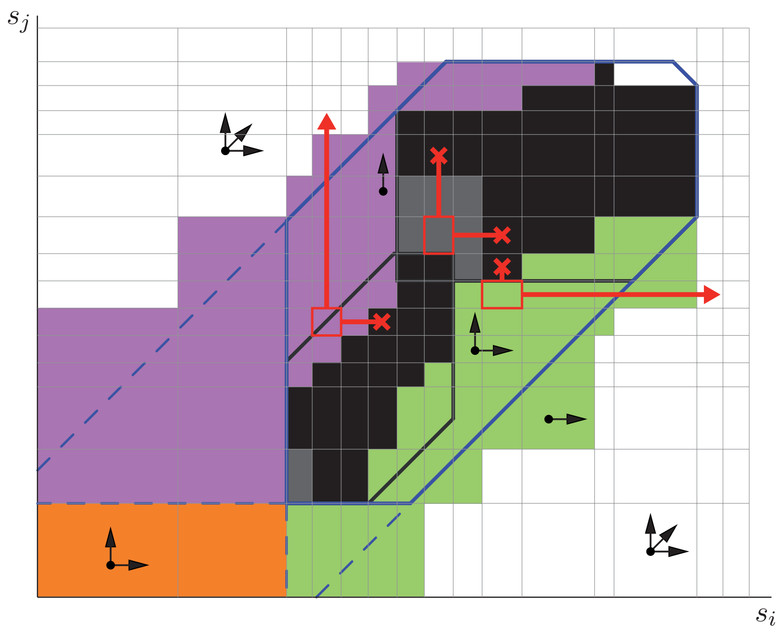


Figure 6.5: A pair of overlapping forbidden regions (in dark gray) are merged. Three examples of blocks inside the forbidden region are represented in red. In order to determine the constraint to apply when a block overlaps the forbidden region, the position of the collision blocks, above and on the right of the considered block is checked.

6.1.4 Complexity analysis

In summary, the algorithm Coordinator defined in this section (see Alg. 7) updates the lists of reserved segments that each vehicle $\mathcal{A}_i$ is allowed to cover. The coordinated action $\mathbf{u}$, with respect to which the lists are updated, is computed by the algorithm Action Computation (see Alg. 8). This algorithm finds the coordinated action which maximizes the advancement of the fleet subject to a set of constraints. The constraints are defined by the function Find Constraints (see Alg. 9) according to the configurations of every pair of AGVs (represented by the block b_{l_i,l_j}). For the analysis of the time complexity of the proposed algorithm a fleet of M starving AGVs is considered. The maximum number of segments composing the path of an AGV is denoted with n. Note that the number of starving vehicles can be reduced by adopting a small value of the time period δ: the less is the time period between any two consecutive iterations, the fewer are the vehicles that reach the starving condition in that period.

In a plane $\mathcal{S}_{ij}$, the maximum number of disjoint collision regions is $O(n^2)$. This happens, for example, if the paths of the vehicles $\mathcal{A}_i$ and $\mathcal{A}_j$ repetitively traverse the same group of colliding segments. However in real applications this is never the case and a reasonable upper bound is $O(n)$. This corresponds to the case in which the paths of $\mathcal{A}_i$ and $\mathcal{A}_j$ repetitively traverse a sequence of different groups of colliding segments.

At the first execution a set of new paths is assigned to all the vehicles, thus the forbidden regions must be computed for all $M(M-1)/2$ planes $\mathcal{S}_{ij}$. The number of blocks in which $\mathcal{S}_{ij}$ is partitioned is n^2. Checking for all collision blocks within a plane and identifying the connected components takes $O(n^2)$ time. Considering $O(n)$ disjoint collision regions the time complexity of computing all the forbidden regions on a single plane is $O(n^3)$. The complexity of computing the forbidden regions for the entire CD is $O(M^2n^3)$.

The maximum number of iterations of the while loop in COORDINATOR (line 3 in Alg. 7) is n. At each iteration, the function ACTION COMPUTATION (see Alg. 8) evaluates the actions constraints by executing the function FIND CONSTRAINTS (see Alg. 9) for each plane. Since Alg. 9 requires constant time, the overall time required to evaluate the constraints on all planes is $O(M^2)$. If multiple collision regions are considered on each plane the complexity becomes $O(M^2n)$. Once that all the constraint have been collected the MIS problem can be solved executing the function MAXIMUM INDEPENDENT SET (line 20 in Alg. 8). The time complexity of the algorithm used to solve the MIS problem ([43]) is related to the the number of vertices of the graph. If all AGVs are starving at the same time the number of vertices is M and the time required by algorithm is $O(M^3)$. This upper bound is derived considering that the maximum number of iterations of the algorithm is M and that, at each iteration, the computation of the minimum degree and the maximum support of all the vertices takes $O(M^2)$.

The overall complexity of the algorithm is

$$O(M^2n^3 + M^2n + M^3) \tag{6.4}$$

6.2 Experiments

In order to test the effectiveness of the presented TMS some simulation experiments have been executed. The two aspects that are evaluated are the computational time of the algorithm and the time performances with which the transportation tasks are

Table 6.1: Computation times of our TMS [ms].

	Number of vehicles							
	10		15		20		25	
	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.
Update $\mathcal{F}$	93	33	201	58	336	131	629	95
Seg. Res.	213	16	266	27	547	56	908	67

executed. The simulation framework emulates the real control framework described in Chap. 2. Three processes are executed on the same PC (an Intel Pentium D 3.20 GHz):

- the *mission manager emulator* (MME) which generates the transportation requests according to a statistical model of the material flow within the considered plant.
- the *vehicle manager emulator* (VME) which interfaces the TMS with the MME and the AGVs. The state of the AGVs is internally emulated by this process.
- the TMS, which communicates with the VME through a UDP protocol. Our TMS is implemented in MATLAB.

For testing the computational requirements of the coordinator algorithm (see Sec. 6.1), the vehicles are supposed to move on a roadmap used in a real industrial plant which is composed both of curvilinear and straight line segments (see Fig. 6.6). The size of the physical layout is 220m x 150m. The roadmap has 988 pick/drop stations and 4515 trajectory segments. The maximum driving speed of the AGVs is 1.5 m/s. The average number of segments constituting a path is 25. For each execution of the COORDINATIOR, the average number of iterations of the while loop (see Alg. 7) is 2. Four simulations running 10, 15, 20 and 25 AGVs have been executed. The average and maximum time required to define the forbidden regions $\mathcal{F}$ (line 3 in Alg. 7) and to update the lists of reserved segments (time to execute the while loop in Alg. 7) are reported in Tab. 6.1. The computational burden required by the other functionalities of the TMS (see Chap. 2) is negligible. The AGVs send their positions and status to the TMS every $\delta = 0.5s$. In some cases the coordinator takes a longer time run. The messages that are received when the TMS is busy executing the coordinator algorithm, are discarded.

However, when a vehicle is allowed to advance, the list of reserved segments is long enough to let it to advance for $5s$. Thus the loss of messages from the AGVs does not affect the motion of the vehicles.

In order to evaluate the performances with which the transportation tasks are executed we consider two standard measures denoted as *time to pick* (TTP) and *time to drop* (TTD). The TTP is the time a load has to wait at the source station before it is picked by an AGV. The TTD is the time required by an AGV to transport the load to the destination. These tests are executed running a simulation on a smaller plant (size: 260m x 45m, 179 pick/drop stations and 1191 trajectory segments) with 7 AGVs. The engineers of Elettric80 have decided to to test the TMS on this layout since it is more affected by traffic congestions. The ratio of loads generation is 140 per hour. The duration of the simulations is about 70 minutes. The proposed algorithm is compared to the one proposed by Elettric80 considering the results from three simulation runs. The average and maximum values of the TTP and TTD are reported in Tab. 6.2. Our TMS allows to obtain TTP values that are lower than the ones obtained by Elettric80. This is mainly due to the better strategy implemented by our TMS for the mission assignment (see Sec. 2.2.1). Considering the TTD values there is not a clear advantage of one algorithm over the other. However, we should consider that the original requires up to 15 days of engineer work to be fine tuned. Differently, the proposed algorithm is able to coordinate vehicles on different roadmaps without the need of any additional engineering work.

6.3 Conclusions and future work

In this chapter we have proposed an algorithm for coordinating multiple AGVs moving on a predefined roadmap for an industrial application. The CD has been applied within an existent industrial framework which implements the zone control approach for specifying motion commands to the vehicles. The overall time complexity of the proposed algorithm is polynomial with respect to the number of AGVs. This is a considerable improvement of the algorithm presented in Chap. 5, the complexity of which is exponential. The computational requirements have also been evaluated through simulations involving up to 25 vehicles on a real roadmap. Simulation tests have been executed

Table 6.2: Performances comparison.

Proposed TMS				
Sim. No.	TTP [s]		TTD [s]	
	Max.	Avg.	Max.	Avg.
1	251	77	191	86
2	242	81	245	89
3	152	69	200	85

Elettrc80 TMS				
Sim. No.	TTP [s]		TTD [s]	
	Max.	Avg.	Max.	Avg.
4	241	90	253	87
5	223	78	159	83
6	256	93	176	85

also in order to compare the performance of our algorithm with the one currently implemented by the company. The results show that currently, the implementation of our algorithm into the existent framework, can improve the performance of the system. Another advantage of our approach is that it saves several days of engineering work each time a new plant must be deployed since it is applicable to any kind of roadmap without the need of specific traffic rules.

In this last algorithm we have removed the possibility for the vehicles to move backward. However it is currently under development a strategy for allowing the AGVs to backtrack along their paths without significantly increasing the computational complexity. When a vehicle is required to go backward, its path can be modified so that the the forward motion along the new path will correspond to a backward motion along the previous one. This method can be seen as a special case of dynamic routing. The amount of segments to be covered in backward motion is determined by using the CD. This could also be the first work in which the CD is exploited for dynamic routing purposes.

Algorithm 9 Find Constraints

```
function FIND CONSTRAINTS($\mathcal{F}, S, l_1, \ldots, l_M$)
    $(xs, xf, ys, yf)$ = BLOCK CORNERS($b_{l_i,l_j}$)
    $(C1, C2, C3, X_{min}, Y_{min}, X_{max}, Y_{max}) = \mathcal{F}$
    if $yf - xs \geq C1 \wedge ys - xf \leq C2$ then
        if $xf \geq X_{min} \wedge yf \geq Y_{min}$ then
            if $xs < X_{min}$ then
                if $ys < Y_{min}$ then
                    $\kappa$ = "$u_i + u_j \leq 1$"                ▷ Decision region
                else
                    $\kappa$ = "$u_i = 0$"                         ▷ Penumbra left
                end if
            else if $ys < Y_{min}$ then
                $\kappa$ = "$u_j = 0$"                             ▷ Penumbra right
            else if $xf \leq X_{max} \wedge yf \leq Y_{max} \wedge$
                    $\wedge\, xf + yf \leq C3$ then
                $\kappa$ = FINDINSIDE($CD$)                        ▷ Forbidden region
            else
                $\kappa$ = "$\emptyset$"                           ▷ Light
            end if
        else
            if $xf \geq X_{min}$ then
                if $xs < X_{min}$ then
                    $\kappa$ = "$u_i + u_j \leq 1$"                ▷ Decision region
                else
                    $\kappa$ = "$u_j = 0$"                         ▷ Penumbra right
                end if
            else if $yf \geq Y_{min}$ then
                if $ys < Y_{min}$ then
                    $\kappa$ = "$u_i + u_j \leq 1$"                ▷ Decision region
                else
                    $\kappa$ = "$u_i = 0$"                         ▷ Penumbra left
                end if
            else
                $\kappa$ = "$\emptyset$"                           ▷ Antumbra
            end if
        end if
    else
        $\kappa$ = "$\emptyset$"                                   ▷ Light
    end if
    return $\kappa$
end function
```

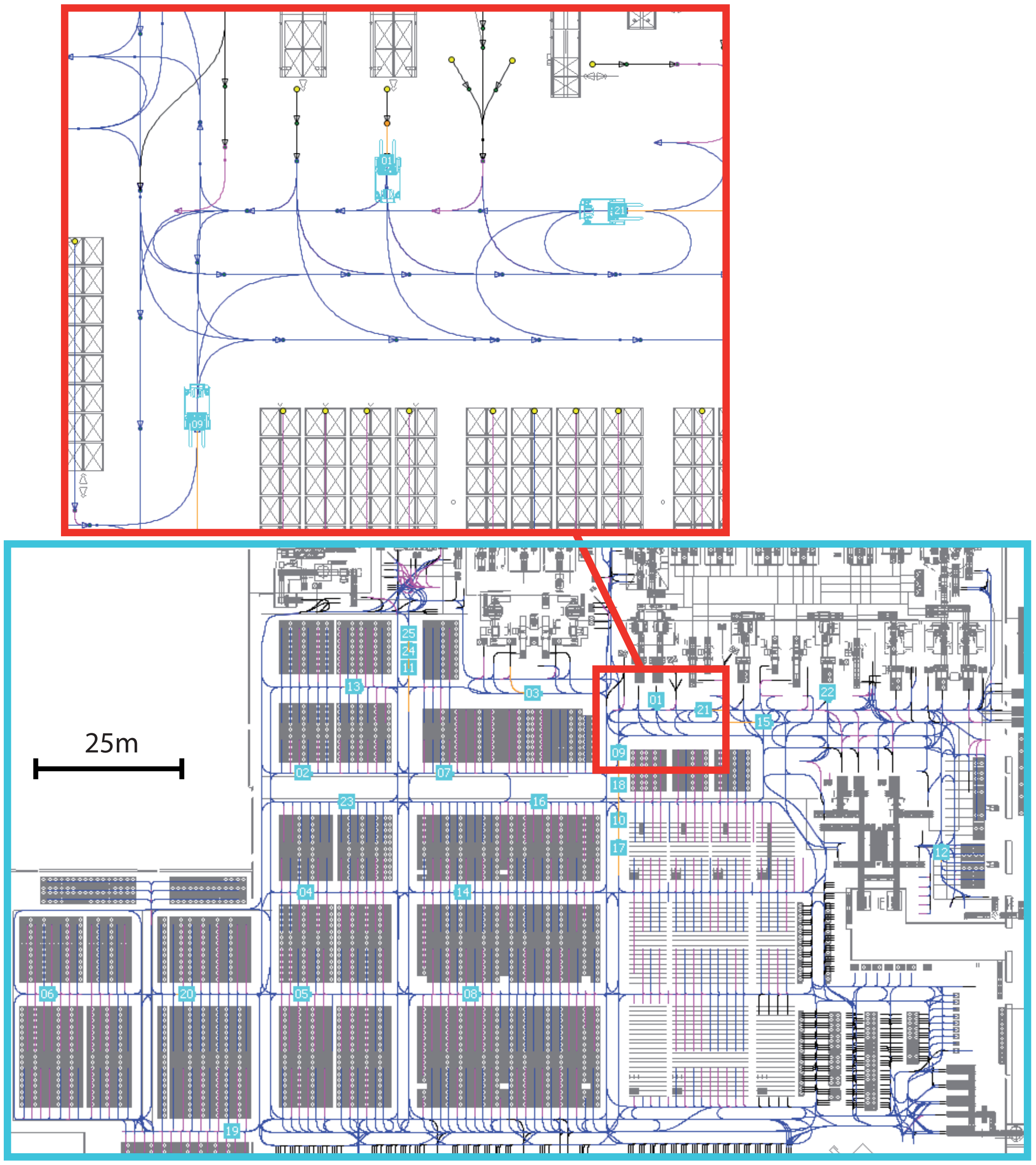


Figure 6.6: Layout of the roadmap used for the test with 25 AGVs. This roadmap is used on a real plant.

Chapter 7

Conclusions

The work realized during the last three years and presented in this thesis has been devoted to the research of a new strategy for coordinating the motion of multiple AGVs operating in a dynamic industrial environment (see Chap. 1). This coordination strategy has to be implemented into the existing framework developed by Elettric 80 S.p.A., a company producing end-of-line automation solutions and AGVSs for warehouses and production plants.

All the four solutions explored in this dissertation exploit the CD ([38]). This tool allows to represent the possible collisions that can take place between any pair of AGVs as regions within a set of planes. The CD can thus be used in order to transform the coordination problem of N vehicles as a path planning problem within an N-dimensional space.

The first issue that has to be considered is the construction of the CD. Exploiting the structure of the application we are dealing with, an efficient procedure for composing the CD has been developed in Sec. 3.3. The possible collisions that can take place between AGVs and the induced geometry of the resulting CD are classified in Sec. 3.2.3. This classification motivates the approximated representations (enclosure rectangles and forbidden regions) of the CD which allows to develop the coordination algorithms proposed in this thesis.

The first approach that has been presented (see Chap. 3) is an algorithm that computes a complete coordination plan for the overall path that each vehicle has to execute. The algorithm generates a set of possible coordination plans and then gives the optimal one. Even if this approach has been successfully applied in a simulation experiment involving 10 vehicles, the computational complexity is too high for an application in a real industrial environment. In fact, the path of the AGVs are frequently modified and unexpected events may prevent the AGVs from executing the planned coordination. In these cases the coordination plans have to be discarded and AGVs needs to stop and wait for a new plan.

In order to coordinate the motion of AGVs within a dynamic industrial environment, where a lot of unexpected events may happen, an incremental coordination is more suitable. In Chap. 4 an algorithm which determines the motion of the AGVs step by step is designed. The approach is based on a mapping between the configuration space of the fleet and a set of motion constraints that the AGVs must satisfy in order to avoid mutual collisions. At each iteration the algorithm defines the motion of the AGVs considering the actual configuration reached by the fleet. Since the motion of the AGVs is not planned in advance, unexpected events can be considered online without the need of replanning. This algorithm has been compared with the complete algorithm both from a computational complexity and a performance point of view. The time factor ([42]) has been used as a performance measure. The results shows that, even if the performance is lower than that obtained with the complete approach, the computational complexity is orders of magnitude lower than that of the complete approach.

In order to simplify the problem, the above mentioned approaches does not take into account the vehicles dynamics. Moreover they can not be easily implemented into the considered industrial framework (see Chap. 2). These issues are considered by the approach described in Chap. 5 and Chap. 6. The algorithm presented Chap. 5 is the implementation of the incremental algorithm into the zone control framework used in the considered application. A partitioned CD has been defined and the constraints introduced in Sec. 4.1 have been adapted in order to consider the new structure of the CD. The main advantage of this solution is that it allows to save several days of engineering work each time a new plant must be deployed since it is applicable to any kind of roadmap without the need of specific traffic rules. Simulation tests on a real roadmap have been executed in order to compare the performance of the algorithm with the one currently implemented by Elettric 80. The results shows the implementation of our algorithm into the existent framework, can slightly improve the performance of the system. The main limitation on the application of this algorithm in a real implementation is the complexity which in the worst case is still exponential in the number of AGVs.

This problem has been solved with the last algorithm, proposed in Chap. 6. This improvement has been achieved mainly by applying a polynomial time heuristic algorithm for the action selection problem. Differently from before, each vehicle is only allowed to wait or to move forward along its path. The coordinator computes the maximum set of AGVs that are allowed to advance while respecting the constraints. A formal analysis of the computational time complexity shows that the overall complexity of this coordinator algorithm is cubic in the number of AGVs. The performances obtained with the proposed TMS are evaluated through some tests involving up to 25 vehicles on a paths layout used in a real industrial plant. Even if the vehicles are not allowed to move backward, as in the previous algorithm, our solution still gives better performances with respect to the one used by Elettric 80.

Besides the study for the development of the coordinator algorithm, the presented work also proposes a solutions for other important issues that have to be considered by a TMS (see Chap. 2). The main functionalities that have been developed are: task dispatching, vehicle routing, management of idle AGVs. In particular, the transport tasks are assigned to available AGVs by solving a linear assignment problem through the Jonker Volgenant algorithm. The objective is to minimize the sum of the times required by each vehicle to reach the pick station assigned to it. The same criteria is applied for dispatching the idle vehicles to a set of predefined home positions.

List of Author's Publications

[1] R. Olmi, C. Secchi, and C. Fantuzzi. **Coordination of multiple AGVs in an Industrial application**. In *IEEE Int. Conf. on Robotics and Automation*, pages 1916–1921, May 2008.

[2] R. Olmi, C. Secchi, and C. Fantuzzi. **Coordination of Industrial AGVs**. *Int. J. Vehicle Autonomous Systems*, **9**(1/2):5–25, 2011.

[3] R. Olmi, C. Secchi, and C. Fantuzzi. **A Traffic Management System for Automated Guided Vehicles in Automatic Warehouses**. *Int. J. Autonomous Robots, Special issue: Motion Safety for Robots*, **Under review**, 2011.

[4] R. Olmi, C. Secchi, and C. Fantuzzi. **Coordination of Multiple Robots with Assigned Paths**. In *7th IFAC Symp. on Intelligent Autonomous Vehicles*, 2010.

[5] R. Olmi, C. Secchi, and C. Fantuzzi. **Coordinating the motion of multiple AGVs in automatic warehouses**. In *IEEE - ICRA10 - Int. Workshop on Robotics and Intelligent, Transportation System*, 2010.

[6] D. Ronzoni, R. Olmi, C. Secchi, and C. Fantuzzi. **AGV Global Localization Using Indistinguishable Artificial Landmarks**. In *IEEE Int. Conf. on Robotics and Automation*, 2011.

[7] R. Olmi, C. Secchi, and C. Fantuzzi. **A Coordination Technique for Automatic Guided Vehicles in an Industrial Environment**. In *IFAC Symp. on Robot Control*, 2009.

Bibliography

[8] R. Jonker and A. Volgenant. **A shortest augmenting path algorithm for dense and sparse linear assignment problems**. *Computing*, **38**(4):325–340, 1987.

[9] J.A. Tompkins, J.A. White, Y.A. Bozer, and J.M.A. Tanchoco. *Facilities Planning, 4th ed.* John Wiley & Sons, Inc., 2010.

[10] Elettric 80 S.p.A. **The autonomous guided vehicles market**. In *Internal report*, 2009.

[11] L. Schulze, S. Behling, and S. Buhrs. **Automated Guided Vehicle Systems: a Driver for Increased Business Performance**. In *Int. MultiConference of Engineers and Computer Scientists*, 2008.

[12] I.F.A. Vis. **Survey of research in the design and control of automated guided vehicle systems**. *Europ. J. of Operational Res.*, **170**:677–709, 2006.

[13] D. Weyns and T. Holvoet. **Architectural Design of a Situated Multiagent System for Controlling Automatic Guided Vehicles**. *Int. J. on Agent-Oriented Soft. Eng.*, **2**(1):90–128, 2008.

[14] T. Le-Anh and M.B.M. DeKoster. **A review of design and control of automated guided vehicle systems**. *Europ. J. of Operational Res.*, **171**:1–23, 2006.

[15] S.-Y. Huang L. Qiu, W.-J. Hsu and H. Wang. **Scheduling and routing algorithms for AGVs: a survey**. *Int. J. Production Res.*, **40**(3):745–760, 2002.

[16] D. Naso and B. Turchiano. **Multicriteria Meta-Heuristics for AGV Dispatching Control Based on Computational Intelligence**. *IEEE Trans. on Systems, Man, and Cybernetics - part B: Cybernetics*, **35**(2):208–226, 2005.

[17] J.H. Lee, B.H. Lee, and M.H. Choi. **A real-time traffic control scheme of multiple AGV systems for collision free minimum time motion: a routing table approach**. *IEEE Trans. on Systems, Man, and Cybernetics - part A: Systems and Humans*, **28**(3):347–358, 1998.

[18] R.L. Moorthy, W. Hock-Guan, N. Wing-Cheong, and T. Chung-Piaw. **Cyclic deadlock prediction and avoidance for zone-controlled AGV system**. *Int. J. Production Economics*, **83**:309–324, 2003.

[19] B. Fleischmann, S. Gnutzmann, and E. Sandvoss . **Dynamic Vehicle Routing Based on Online Traffic Information**. *Transportation Science*, **38**(4):420–433, 2004.

[20] T. Le-Anh, R.B.M. de Koster, and Y. Yu. **Performance evaluation of dynamic scheduling approaches in vehicle-based internal transport systems**. *Int. J. of Production Res.*, **48**(24):7219–7242, 2010.

[21] C. Hu and P.J. Egbelu. **A framework for the selection of idle vehicle home locations in an automated guided vehicle system**. *Int. J. of Production Res.*, **38**(3):543 – 562, 2000.

[22] A. Farinelli, L. Iocchi, and D. Nardi. **Multirobot systems: a classification focused on coordination**. *IEEE Trans. Systems, Man and Cybernetics, Part B*, **34**(5):2015–2028, Oct 2004.

[23] R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert. **Multirobot cooperation in the MARTHA project**. *IEEE Robot. Automat. Mag.*, **5**(1):36–47, Mar 1998.

[24] E. Xidias and N. Aspragathos. **Motion planning for multiple nonholonomic robots: A geometric approach**. *Robotica*, **26**(4):525–536, 2008.

[25] J. Peng and S. Akella. **Coordinating Multiple Double Integrator Robots on a Roadmap: Convexity and Global Optimality**. In *IEEE Int. Conf. on Robotics and Automation*, pages 2751–2758, Apr 2005.

[26] R. Ghrist, J.M. O'Kane, and S. LaValle. **Computing Pareto optimal coordination on roadmaps**. *Int. J. Robot. Res.*, **24**(11):997–1010, 2005.

[27] N. Smolic-Rocak, S. Bogdan, Z. Kovacic, and T. Petrovic. **Time Windows Based Dynamic Routing in Multi-AGV Systems**. *IEEE Trans. on Automation Science and Engineering*, **7**(1):151–155, 2010.

[28] S.M. LaValle and S.A. Hutchinson. **Optimal motion planning for multiple robots having independent goals**. *IEEE Trans. on Robotics and Automation*, **14**(6):912–925, Dec 1998.

[29] O. Purwin, R. D'Andrea, and J.W. Lee. **Theory and implementation of path planning by negotiation for decentralized agents**. *Robot. Auton. Syst.*, **56**(5):422–436, 2008.

[30] D. Herrero-Pérez and H. Martínez-Barberá. **Modeling Distributed Transportation Systems Composed of Flexible Automated Guided Vehicles in Flexible Manufacturing Systems**. *IEEE Trans. on Industrial Informatics*, **6**(2):166–180, 2010.

[31] A. Giridhar and P.R. Kumar. **Scheduling automated traffic on a network of roads**. *IEEE Trans. on Vehicular Technology*, **55**(5):1467–1474, 2006.

[32] N. Wu and M. Zhou. **Modeling and deadlock control of automated guided vehicle systems**. *IEEE/ASME Trans. on Mechatronics*, **9**(1):50–57, Mar 2004.

[33] M.P. Fanti. **Event-based controller to avoid deadlock and collisions in zone control AGVs**. *Int. J. Production Res.*, **40**(6):1453–1478, 2002.

[34] S.A. Reveliotis. **Conflict resolution in AGV systems**. *IIE Transactions*, **32**(7):647–659, 2000.

[35] T. Simeon, S. Leroy, and J.P. Laumond. **Path coordination for multiple mobile robots: a resolution-complete algorithm**. *IEEE Trans. Robot. Automat.*, **18**(1):42–49, Feb 2002.

[36] T.W. Min, L. Zhe, H.K. Yin, G.C. Hiang, and L.K. Yong. **A rules and communication based multiple robots transportation system**. In *IEEE Int. Symp. on Computational Intelligence in Robotics and Automation*, pages 180–186, 1999.

[37] T. Borowiecki and Z. Banaszak. **A constraint programming approach for AGVs flow control**. In *Workshop on Robot Motion and Control*, pages 153–158, 1999.

[38] P.A. O'Donnell and T. Lozano-Perez. **Deadlock-Free and Collision-Free Coordination of Two Robot Manipulators**. In *IEEE Int. Conf. on Robotics and Automation*, **1**, pages 484–489, May 1989.

[39] Y. Guo and L.E. Parker. **A distributed and optimal motion planning approach for multiple mobile robots**. In *IEEE Int. Conf. on Robotics and Automation*, **3**, pages 2612–2619, May 2002.

[40] M. Jager. **Using software agents to avoid collisions among multiple robots**. In *IEEE Int. Cof. on Tools with Artif. Intell.*, pages 50–57, 2001.

[41] S. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. Also available at http://planning.cs.uiuc.edu/.

[42] S. Berman, E. Schechtman, and Y. Edan. **Evaluation of Automatic Guided Vehicle Systems**. *Robot. Comput. Integr. Manuf.*, **25**(3):522–528, 2009.

[43] S. Balaji, V. Swaminathan, and K. Kannan. **A Simple Algorithm to Optimize Maximum Independent Set**. *Adv. Modeling and Optimization*, **12**(1):107–118, 2010.