CHALMERS
UNIVERSITY OF TECHNOLOGY

Improving deadlock avoidance in multi-AGV systems

Master's thesis in Production Engineering
Master's thesis in Systems, Control and Mechatronics

Weihan Qin
Zixiao Ren

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

MASTER'S THESIS 2020

Improving deadlock avoidance in multi-AGV systems

Weihan Qin
Zixiao Ren

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Improving deadlock avoidance in multi-AGV systems
Weihan Qin
Zixiao Ren

Supervisor: Sarmad Riazi, AGVE
Examiner: Martin Fabian, Electrical Engineering

Master's Thesis 2020
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: AGVs in the Volvo Cars plant. The left vehicle carries a rack loaded with sheet metal parts.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Improving deadlock avoidance in multi-AGV systems
Weihan Qin, Zixiao Ren
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Automated Guided Vehicles (AGVs) are widely used for materials handling operations in varieties of industries, including the automotive industry, energy industry, and food industry. AGVs play an increasingly important role because of the high efficiency, low management cost, and high flexibility. This thesis is carried out in cooperation with AGVE AB, a provider of AGV systems located in Gothenburg, aiming to improve the deadlock avoidance performance in the control system. Currently, the AGVs move on predefined paths. The system is collision-free but requires a significant amount of manual work to avoid collisions and deadlocks. Based on the existing centralized control system, a structured procedure to avoid deadlocks was proposed. Traffic control rules were automatically generated to integrate with the system. Tests were done on two different layouts and showed good results. Due to the constraints of the current system, further ideas were discussed at the end as a foundation of future work.

Keywords: AGV, deadlock avoidance, traffic control.

Acknowledgements

I would like to express my gratitude to AGVE AB for offering this interesting topic to work with. Moreover, I would like thank our supervisor Sarmad for the patient guidance and the support, especially the mental support during this pandemic. It was nice working with you! Further, I appreciate the support from Thomas who spent time answering our endless questions about the software. I would also acknowledge the help from our examiners Martin and Kristofer, who listened to our ideas and suggested thoughtful comments. Thank you Martin for coming up with the verification ideas even though we did not manage to apply it and reading the report, and thank you Kristofer for suggesting helpful ideas. Big thanks! Meanwhile, thank you my partner for working together and getting this great results. Finally, many thanks to my friends and family for supporting me throughout entire process, both by listening to my complaints and encouraging me to go on.

Weihan Qin
Gothenburg, August 2020

I would like to extend my deep gratitude to all those who have offered me a lot of help and support in the process of my thesis writing, especially during the Corona-19 epidemic period.
Firstly, I am greatly indebted to my supervisor Sarmad Riazi for his enlightening guidance and valuable advice through the entire process of the thesis. Without your help and encouragement, I could not have finished this thesis.
Besides my supervisor, I have benefited from my examiner, Martin Fabian and co-examiner, Kristofer Bengtsson. Thanks for the instructions of research topic and directions and guidance in academic writing. And also thank you for helping us contact with the company AGVE AB to find this interesting thesis work.
Also, I would like to thank Thomas Diding for your patience in answering our questions and Christian Skeppstedt for providing us this thesis in AGVE AB.
Last my thanks would go to my family and my friends for supporting me a comfortable atmosphere to finish this thesis.

Zixiao Ren
Gothenburg, August 2020

Contents

Abbreviation

AGV	Automated Guided Vehicle
AGVs	Automated Guided Vehicles
AGVS	Automated Guided Vehicle System
IDOT	Information, Digital, and Operation Technologies
BAM	Best AGV Manager
ACE	AGV Command Executive
LDK	Layout Definition Kit(LDK)

List of Figures

1
Introduction

1.1 Background

Materials handling is an important process to improve the productivity and a key to a successful business. When working in the warehouse or plant, it is possible that the shop-floor workers made mistakes hurting themselves and the products, which could suspend the production. For example, palletized goods can be damaged due to mishandling, which results in unsaleable products and unnecessary work.

Automated Guided Vehicles (AGVs) are a solution to this problem, being unmanned pallet trucks with higher precision than humans. In addition to the safety and damage reduction, AGVs are able to reduce the non-value-adding transportation time. For example, with the help of AGVs transporting kits, the workers can focus on the assembly work rather than spend time searching for components. Furthermore, AGVs can be more flexible than conveyors. These advantages and the current experience level led to AGVs being introduced to a variety of industries, involving production facilities and warehouses.

The first AGV was invented in 1953 in America. The first Automated Guided Vehicle System (AGVS) was installed as a tractor-trailer in 1954 at Mercurcy Motor Freight Company in South California [2]. As the fourth industrial revolution and the underlying digital transformation going on, AGV has been playing an increasingly important role as a part of advanced Information, Digital, and Operation Technologies (IDOT) [3]. Most big manufacturing companies have already implemented AGVS in the plant, such as Volvo Cars.

There are some research activities in this field, such as vision, path planning, fast reconfiguration, and optimal resource allocation, etc. Additionally, multiple AGVs are coordinated to work together to handle the materials efficiently in an AGVS, which may lead to collision and deadlock problems. If the AGVS is not collision-free, the vehicles and the materials in it may be in danger. If the deadlock problem is not handled properly, the AGVs may be stuck waiting for each other and contribute no values or even have a negative impact such as wasting energy.

Collisions with static and dynamic objects should be avoided in the AGVS. One straightforward way to solve this is applying sensors to keep enough safety zones and the AGVs slowed down and stopped when anything is detected in the safety zone [4]. To avoid collision between the AGVs, there are centralized collision avoidance and decentralized collision avoidance algorithms such as Forward sensing, Re-planning using A^* or D^*, and Predictive models [4].

As for the deadlock avoidance, there are already several ways, including centralized and decentralized methods [4]. One centralized way is solving it as a large optimization problem together with task allocation and collision avoidance since all the locations and itineraries of the AGVs are updated in the central controller. Petri nets, Automata, and digraphs are three major tools to do that [5]. Correspondingly, the decentralized deadlock avoidance takes use of the local information of the AGVs. In addition, zone control is another effective strategy, which restricts the maximum number of AGVs in several control zones. Moreover, some routing strategies can be designed to prevent deadlocks [4].

In this thesis, the main AGV management system has been developed and used for years. There is already an efficient approach to solve the conflict free scheduling and routing of AGVs, which was proposed by Riazi, et al [6]. However, the current method to solve the deadlock is totally manual. The engineer manually writes the traffic management rules according to observations from simulation results and feedback from the customer. Furthermore, this work have to be repeated every time before implementing the AGVS on a new layout, which is inefficient and error-prone. This thesis will improve the deadlock avoidance in the AGVS, based on the previous research in this area and the existing AGVS. This thesis is done together with AGVE AB, a provider of AGVs and control devices, located in Gothenburg. The simulation and tests will be run on systems that are already implemented.

1.2 Objective

This thesis aims to improve the deadlock avoidance in the existing AGVS. This thesis will focus on developing a tool to solve the deadlock problem without repetitive heavy manual work. This tool should be automatic or minimal manual work required and effective regardless of different layouts and AGVs of different sizes.

1.3 Scope

This thesis will be carried out on the existing AGV management system and software developed by AGVE AB. This project will not modify the order allocation algorithm or the path planning algorithm. Moreover, the layouts marked with nodes and arcs were provided by AGVE AB, which means the AGV will follow predefined paths and not move freely. No modification to the layouts will be made to improve the deadlock avoidance.

Given the constraints mentioned above, this project will only consider centralized deadlock avoidance. Similarly, neither the methods related to the routing strategy nor freely-moving AGVs will be taken into consideration.

One of the main focuses of this project is to solve the problem for the company. The methods that not be integrated with the existing system currently will also be discussed and functioned as the foundation of future development.

1.4 Work Division

Most of the work were completed with the cooperation of the authors, such as the literature review, report writing and programming. Zixiao took the main responsibility for testing and simulation, while Weihan had the main responsibility for researching and coming up with other methods. The report writing was divided as: Weihan mainly wrote Introduction and Preliminaries. Zixiao mainly wrote Methods. Results and Future work are completed by the authors together.

1.5 Outline

The first chapters, Chapter 1 and Chapter 2, give the background and preliminaries required to comprehend the contributions of the thesis. In Chapter 2, we introduce the software and its functions relative with this project, and a literature review of the research on deadlock avoidance is also included. Chapter 3 introduces the methods to be implemented in the thesis based on the literature review. Two methods are discussed in this chapter. Chapter 4 summarizes the results from the testing and simulation as well as the discussion of the results. In Chapter 5.1, we propose some suggestions on how to extend the work in this thesis.

2
Preliminaries

In order to control and simulate the AGVs, AGVE AB has developed an AGV control software package. Considering the compatibility and time limitation, the main tests and simulations in this project were carried out in this software. It is necessary to introduce some features in this software package for understanding the work in this thesis.
In the following sections, the AGV control system and the features related to the thesis work are introduced. In addition, there are already research done to solve the deadlock problem. A literature review of this field is presented at the end of this chapter.

2.1 Best AGV Manager

Best AGV Manager is the real-time transportation and traffic control system using the AGVE AB's AGV Command Executive (ACE) concept for navigation and communication with automatic vehicles. It is a Microsoft Windows based stationary control software package for AGV based material handling system. The operations of the AGV system performing materials transportation is controlled by the Best AGV Manager. When the external sources make requests for moving materials, the system can queue and allocate the requests to AGVs for execution. To assign the request to an AGV, the request is broken down into discrete AGV commands such as moving to somewhere, picking the material, and dropping the material. Then the communication between the control system and the AGVs can send these commands as well as monitor the status of the AGVs, for example, the position of the AGV, and the execution process of the order.
AGV traffic management and collision avoidance are two main features of the software. In addition to these two features, the system can also schedule and control the battery charging, communicate with external devices such as doors and conveyors, and detect system faults. In the following sections, the features related to this project are introduced in order to clarify how the system is implemented.

2.1.1 Layout

The layout is one of the most basic features in the user interface of the system, which provides the overall information about the space utilization. The basic elements used to define a layout are *nodes* and *links*. *Nodes* define physical areas or logical zones that an AGV can occupy. *Links* connect nodes to adjacent nodes and define the

ACE tables used to command the AGV to move from the starting node to the ending node. Nodes with special functions are called *locations*. A location can be a station where the materials are picked and dropped, door, or battery charge. All the nodes are numbered. It is worth noting that all the paths are one-way paths with an arrow marking the direction. A layout with four AGVs presented in the user interface is shown in Figure 2.1.

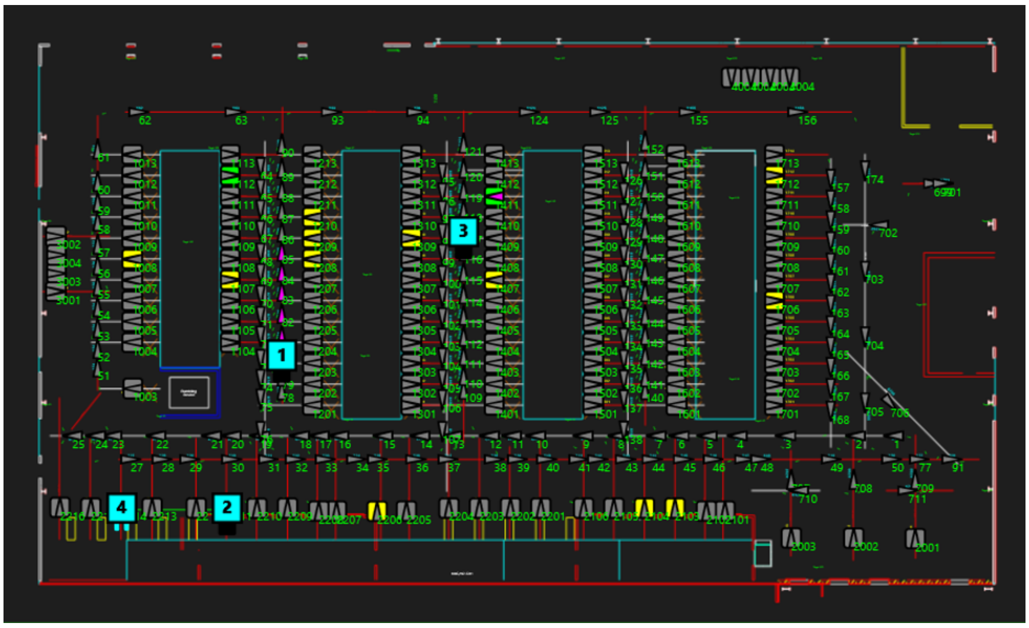


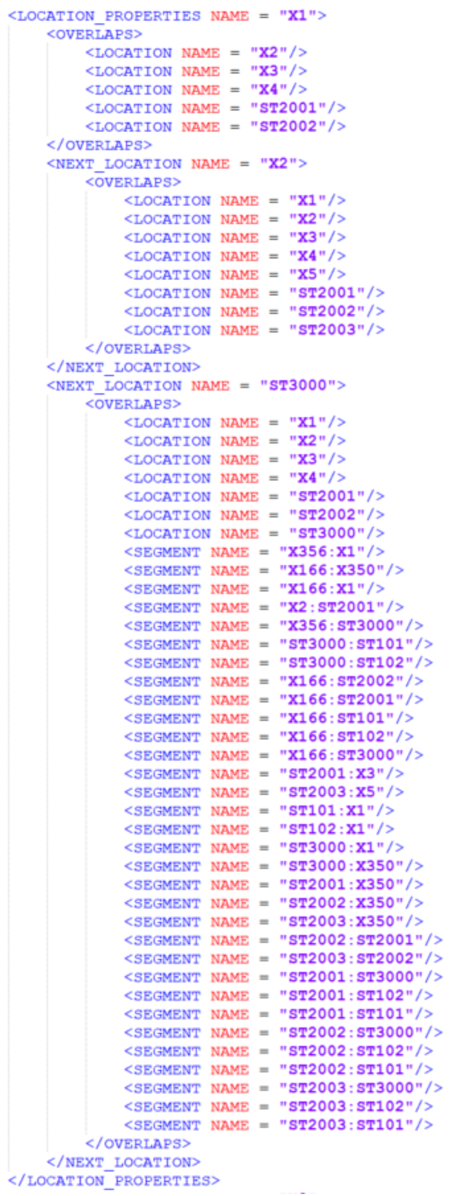
Figure 2.1: A layout containing 4 AGVs presented in the user interface. The green numbers stand for the identifier of the nodes and the boxes numbered 1 to 4 represent the respective AGVs.

The layout was designed in AutoCAD with the help of a powerful plug-in tool named Layout Definition Kit (LDK). With LDK, all ACE tables, curve shapes, routing data, layout build files are automatically generated.

2.1.2 System Documents

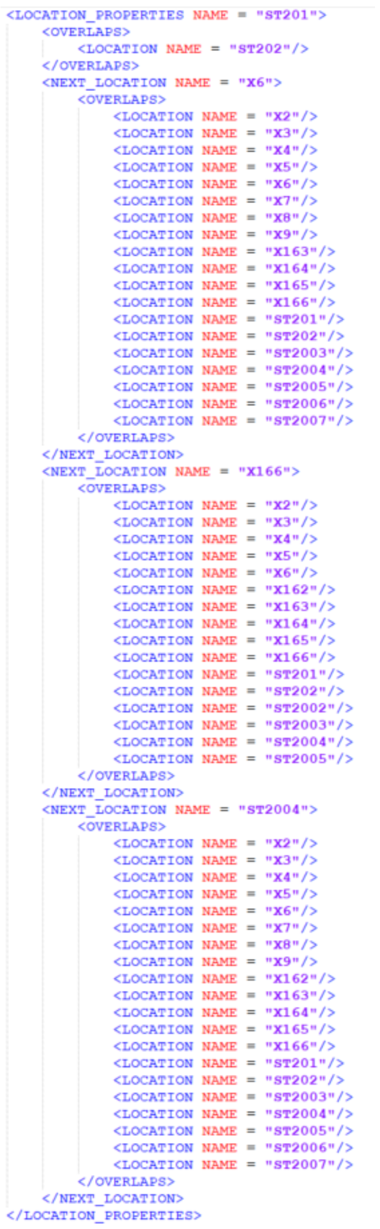
To support the operation of the entire AGVS, all the related data and codes are stored in several documents. To clarify some terminologies and definitions for better understanding, three documents related to this thesis are described below: SYSTEM_SETUP.XML, BLD.XML, OVERLAPINFO.XML.
The commands and rules controlling the behaviour of the AGVs and the properties of the locations are stored in the file named SYSTEM_SETUP.XML, where the codes implemented and mentioned in Chapter 3 to avoid deadlock are stored in this document as well. To combine the CAD layout information with the control system, the coordinates and the number of nodes are stored in the file named BLD.XML. Besides that, the link information is stored here by defining both the connected nodes and the direction.
The file named OVERLAPINFO.XML provides overlap information of the AGVs. That is to say, different nodes are occupied by AGVs of different directions or

itineraries. The first branch of OVERLAPINFO.XML file is the node name. The second branch is the overlap information of the node or the node's next location's node name. The third branch is the overlap information of the nodes. The forth branch is the overlap information when the node has direction to the next location. For instance, Figure 2.2.a shows the overlap information of Node X1, which is stored in a tree structure. When an AGV waits on Node X1, the four nodes X2, X3, X4, ST2001, ST2002, right under the second branch OVERLAPS, are occupied. Furthermore, for an AGV passing by Node X1 with direction to Node X2, the nodes right below the branch NEXT_LOCATION NAME = 'X2' are occupied. Similarly, more nodes and links are occupied with Node ST3000 in the itinerary.



```
<LOCATION_PROPERTIES NAME = "X1">
    <OVERLAPS>
        <LOCATION NAME = "X2"/>
        <LOCATION NAME = "X3"/>
        <LOCATION NAME = "X4"/>
        <LOCATION NAME = "ST2001"/>
        <LOCATION NAME = "ST2002"/>
    </OVERLAPS>
    <NEXT_LOCATION NAME = "X2">
        <OVERLAPS>
            <LOCATION NAME = "X1"/>
            <LOCATION NAME = "X2"/>
            <LOCATION NAME = "X3"/>
            <LOCATION NAME = "X4"/>
            <LOCATION NAME = "X5"/>
            <LOCATION NAME = "ST2001"/>
            <LOCATION NAME = "ST2002"/>
            <LOCATION NAME = "ST2003"/>
        </OVERLAPS>
    </NEXT_LOCATION>
    <NEXT_LOCATION NAME = "ST3000">
        <OVERLAPS>
            <LOCATION NAME = "X1"/>
            <LOCATION NAME = "X2"/>
            <LOCATION NAME = "X3"/>
            <LOCATION NAME = "X4"/>
            <LOCATION NAME = "ST2001"/>
            <LOCATION NAME = "ST2002"/>
            <LOCATION NAME = "ST3000"/>
            <SEGMENT NAME = "X356:X1"/>
            <SEGMENT NAME = "X166:X350"/>
            <SEGMENT NAME = "X166:X1"/>
            <SEGMENT NAME = "X2:ST2001"/>
            <SEGMENT NAME = "X356:ST3000"/>
            <SEGMENT NAME = "ST3000:ST101"/>
            <SEGMENT NAME = "ST3000:ST102"/>
            <SEGMENT NAME = "X166:ST2002"/>
            <SEGMENT NAME = "X166:ST2001"/>
            <SEGMENT NAME = "X166:ST101"/>
            <SEGMENT NAME = "X166:ST102"/>
            <SEGMENT NAME = "X166:ST3000"/>
            <SEGMENT NAME = "ST2001:X3"/>
            <SEGMENT NAME = "ST2003:X5"/>
            <SEGMENT NAME = "ST101:X1"/>
            <SEGMENT NAME = "ST102:X1"/>
            <SEGMENT NAME = "ST3000:X1"/>
            <SEGMENT NAME = "ST3000:X350"/>
            <SEGMENT NAME = "ST2001:X350"/>
            <SEGMENT NAME = "ST2002:X350"/>
            <SEGMENT NAME = "ST2003:X350"/>
            <SEGMENT NAME = "ST2002:ST2001"/>
            <SEGMENT NAME = "ST2003:ST2002"/>
            <SEGMENT NAME = "ST2001:ST3000"/>
            <SEGMENT NAME = "ST2001:ST102"/>
            <SEGMENT NAME = "ST2001:ST101"/>
            <SEGMENT NAME = "ST2002:ST3000"/>
            <SEGMENT NAME = "ST2002:ST102"/>
            <SEGMENT NAME = "ST2002:ST101"/>
            <SEGMENT NAME = "ST2003:ST3000"/>
            <SEGMENT NAME = "ST2003:ST102"/>
            <SEGMENT NAME = "ST2003:ST101"/>
        </OVERLAPS>
    </NEXT_LOCATION>
</LOCATION_PROPERTIES>
```

a. Overlap information of node X1



```
<LOCATION_PROPERTIES NAME = "ST201">
    <OVERLAPS>
        <LOCATION NAME = "ST202"/>
    </OVERLAPS>
    <NEXT_LOCATION NAME = "X6">
        <OVERLAPS>
            <LOCATION NAME = "X2"/>
            <LOCATION NAME = "X3"/>
            <LOCATION NAME = "X4"/>
            <LOCATION NAME = "X5"/>
            <LOCATION NAME = "X6"/>
            <LOCATION NAME = "X7"/>
            <LOCATION NAME = "X8"/>
            <LOCATION NAME = "X9"/>
            <LOCATION NAME = "X163"/>
            <LOCATION NAME = "X164"/>
            <LOCATION NAME = "X165"/>
            <LOCATION NAME = "X166"/>
            <LOCATION NAME = "ST201"/>
            <LOCATION NAME = "ST202"/>
            <LOCATION NAME = "ST2003"/>
            <LOCATION NAME = "ST2004"/>
            <LOCATION NAME = "ST2005"/>
            <LOCATION NAME = "ST2006"/>
            <LOCATION NAME = "ST2007"/>
        </OVERLAPS>
    </NEXT_LOCATION>
    <NEXT_LOCATION NAME = "X166">
        <OVERLAPS>
            <LOCATION NAME = "X2"/>
            <LOCATION NAME = "X3"/>
            <LOCATION NAME = "X4"/>
            <LOCATION NAME = "X5"/>
            <LOCATION NAME = "X6"/>
            <LOCATION NAME = "X162"/>
            <LOCATION NAME = "X163"/>
            <LOCATION NAME = "X164"/>
            <LOCATION NAME = "X165"/>
            <LOCATION NAME = "X166"/>
            <LOCATION NAME = "ST201"/>
            <LOCATION NAME = "ST202"/>
            <LOCATION NAME = "ST2002"/>
            <LOCATION NAME = "ST2003"/>
            <LOCATION NAME = "ST2004"/>
            <LOCATION NAME = "ST2005"/>
        </OVERLAPS>
    </NEXT_LOCATION>
    <NEXT_LOCATION NAME = "ST2004">
        <OVERLAPS>
            <LOCATION NAME = "X2"/>
            <LOCATION NAME = "X3"/>
            <LOCATION NAME = "X4"/>
            <LOCATION NAME = "X5"/>
            <LOCATION NAME = "X6"/>
            <LOCATION NAME = "X7"/>
            <LOCATION NAME = "X8"/>
            <LOCATION NAME = "X9"/>
            <LOCATION NAME = "X162"/>
            <LOCATION NAME = "X163"/>
            <LOCATION NAME = "X164"/>
            <LOCATION NAME = "X165"/>
            <LOCATION NAME = "X166"/>
            <LOCATION NAME = "ST201"/>
            <LOCATION NAME = "ST202"/>
            <LOCATION NAME = "ST2003"/>
            <LOCATION NAME = "ST2004"/>
            <LOCATION NAME = "ST2005"/>
            <LOCATION NAME = "ST2006"/>
            <LOCATION NAME = "ST2007"/>
        </OVERLAPS>
    </NEXT_LOCATION>
</LOCATION_PROPERTIES>
```

b. Overlap information of station ST201

Figure 2.2: The Overlap Information is stored in a tree structure. The SEGMENT represents the link between the two nodes after the equal sign.

The overlap information varies according to the size of AGV even in the same system since this is used for collision avoidance and deadlock avoidance as well. In the following sections, the working principle will be detailed.

2.1.3 Curved route

The routes connecting the locations are either straight or curved, depending on the type of the locations. The routes connecting the nodes are always straight, while the routes between station and node are always curved. As shown in Figure 2.3, when one AGV enters or leaves a station, it will follow a predefined curve. When an AGV wants to leave station ST2012, it will either follow the yellow arrow upwards to go to node X158 or move downwards to node X14 by following the yellow arrow downwards. In the process of leaving the station, if other AGVs are occupying overlapped locations, a collision could happen. It is worth noting that the paths for AGVs to enter and leave stations are different. The red arrow in Figure 2.3 shows the curve for an AGV to enter station ST2011 from node X158.

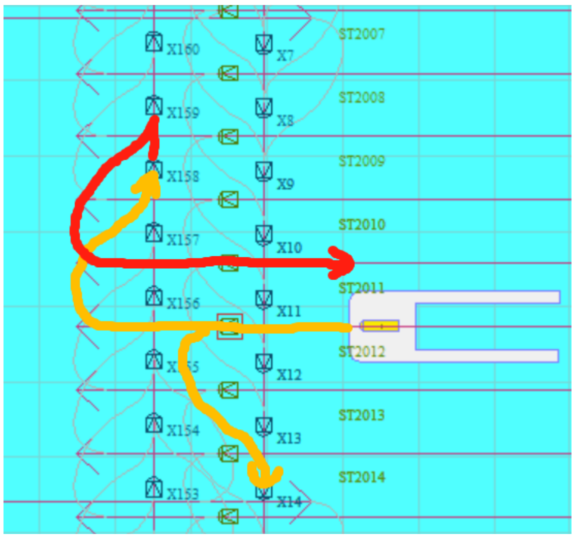


Figure 2.3: Route for AGV to go into or get out of stations.

2.1.4 Order allocation

To decide which order to process and which AGV to assign the order to, the order allocation system controls the assignment of an order from the list of orders to the AGVs according to different criteria. The user can select from these criteria via an option, to carry out the order allocation. For instance, one criterion is to minimize the delays.
The working principle is based on the communication between the AGV control system BAM and the AGVs. The algorithm uses the input from the location and availability report of AGVs, as well as the orders in the queue, to optimize the allocation process and minimize according to a certain criterion. Figure 2.4 shows the working principle and communication between the control system and the AGVs.

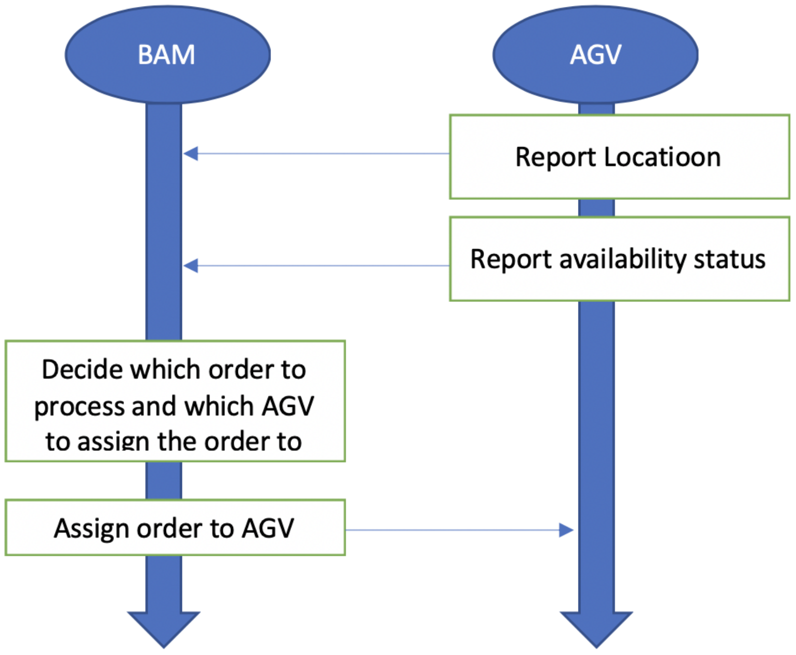

BAM
AGV
Report Locatioon
Report availability status
Decide which order to process and which AGV to assign the order to
Assign order to AGV

Figure 2.4: Working principle of the order allocation and the communication between the control system BAM and AGVs.

2.1.5 Path planning and Collision avoidance

As mentioned in Section 2.1.1, the routes are predefined in the layout with the LDK. To execute the material handling operation, AGVs move from one position to a destination position following certain routes. The route, or to say the itinerary, is a list of node numbers to be traversed from the current node to the destination node. It is calculated from the directed graph defined by the layout editor using Dijkstra's Algorithm [7], which is an algorithm for finding shortest paths between nodes in a graph. This routing system can determine a path from any node to any other node in the system in real time to guide the AGVs.
To avoid collisions in the AGVS, a node locking mechanism supports the traffic control. The AGV occupies or reserves several nodes usually along the direction of the destination. Different nodes are occupied or locked according to different statuses of AGVs, either waiting at one node or heading somewhere. Once a node is reserved or occupied, only the AGV locking the node can enter that node by receiving commands to move to that node, and any other AGVs wanting to occupy the node are commanded to wait until the AGV locking the node leaves and releases it.
However, this locking mechanism is not enough to make the system collision-free; when two nodes are defined so close together that if two AGVs were at each node, their outlines would overlap and thus a collision would occur, the normal locking mechanism still regards this as safe. To handle this, a *Lock* statement is created by LDK when creating the layout to define areas that must not currently be locked for the AGV to lock the problematic node. Thus, if one AGV wants to lock a node

associated with a *Lock* statement, the traffic control system checks whether the next node in question is locked. If it is already locked, the AGV waits on the current node. If it is not locked, then all nodes in the *Lock* area are checked. The AGV can lock and move on to the next node only when none of them are locked.

2.2 Deadlock introduction

In this section, the basic concept of deadlock is introduced, especially the characteristic of deadlock in this AGVS. The study of deadlocks can be traced back to computer scientists' work in the 1960's, where memory assignments in operating systems were studied as resource allocation systems [5]. Although there exist research done for the deadlock problem in operation systems and distributed databases, this cannot be applied in this case directly because of different technical backgrounds. Coffman in [8] defined four necessary conditions for a deadlock to arise:

- Mutual exclusion: The resources are exclusively used by the tasks.
- Wait for: Tasks hold resources already allocated to them while they may require additional resources.
- No preemption: Resources cannot be forcibly removed from the tasks holding them and they can only be released after the completion of the task.
- Circular wait: Two or more tasks require the next resource currently held by other tasks in the chain, as illustrated in Figure 2.5.

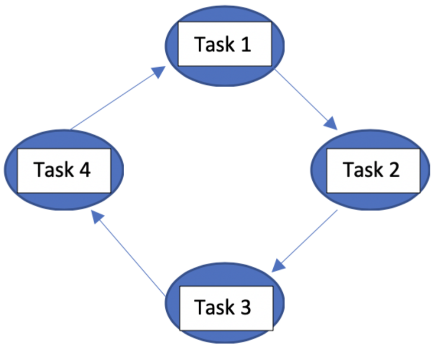


Figure 2.5: A graph illustrating circular wait. The circles represent the resources held by the tasks denoted in the boxes.

That is to say, all these four conditions must hold to form a deadlock situation.
To relate these conditions with the AGVS in this project, the vehicle can be regarded as the task seeking for the opportunity to acquire resources, while the nodes in the layout can play the role of resources. Then the process of vehicles moving around can be understood as the task acquiring resources. For the *mutual exclusion* condition, a bunch of nodes or to say, the zone can be only occupied by one vehicle, which otherwise leads to a collision. The move commands sent to the vehicle mean the requirement for additional nodes, which is the *wait for* condition. As for the *no*

preemption condition, no nodes can be preempted since the nodes are fixed and the vehicles can only move following the link between them. The first three conditions are somehow inherent features of the system and not changeable, while the fourth condition *circular wait* seems to be a breakthrough to solve the deadlock problem. As mentioned in Section 2.1, the locking mechanism supports the movement and collision avoidance of AGVs. The AGV first checks whether the next node on its itinerary is occupied or locked, and it moves only when it is free. According to the above circular wait condition, one of the most typical kinds of deadlock in this system is explained here.

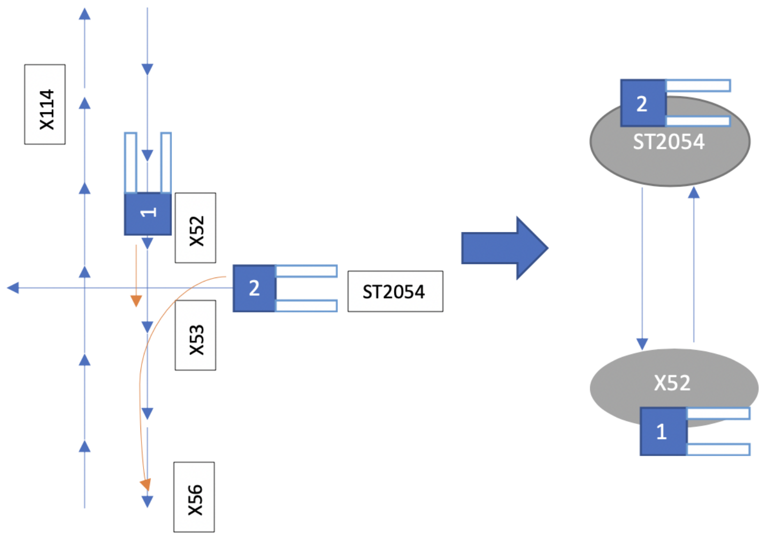


Figure 2.6: Illustration of a deadlock involving two AGVs.

As shown in Figure 2.6, AGV 1 standing on X52 heads for node X53, while AGV2 standing on ST2054 heads for X56. According to the OVERLAPINFO.XML, the control system requires ST2054 to be free to allow AGV1 to move from X52 to X53. However, ST2054 is currently occupied by AGV2. Meanwhile, node X52 occupied by AGV1, is required to be idle to allow AGV2 moving from ST2054 to X56. The OVERLAPINFO.XML of node X52 and ST2054 is attached in Figure A.1 in Appendix A. This means that a circular wait situation involving two states occurs, as shown in Figure 2.6, right.

2.3 Literature review

Deadlock avoidance in AGV systems is an active research field. Some techniques developed in the research are limited to specific layout designs and not generic, and these do not fit the layouts in this project. Although it is challenging to directly apply many of the developed theories and methods, they are still inspirational and helpful. Some literature is reviewed in this section.

Fanti, et al. in [1] uses digraph theoretical concepts to define the necessary and sufficient conditions of deadlock and propose real-time deadlock avoidance rules by inhibiting unsafe transitions. The control scheme consists of two levels, a path scheduler to assign paths to vehicles and a real-time controller to avoid deadlocks. The real-time controller has two functions [1]:

- Path Validation: Check whether the new path is admissible, that is, does not lead to deadlocks. If not admissible, the path scheduler level proposes other paths or postpones this order.
- One-step Control: Check whether the movement can result in a deadlock situation or not. The movement is prevented if leading to a deadlock situation.

In order to implement the digraph tools, the AGVS is considered as a job/resource system. A layout with paths is divided into different *zones*, as in Figure 2.7. The AGVs are regarded as the jobs while the zones are regarded as the *resources* [9].

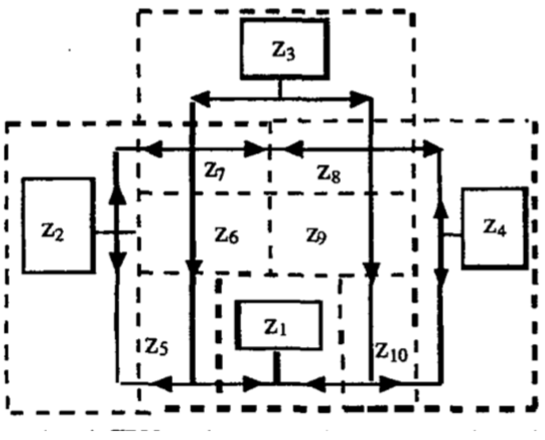
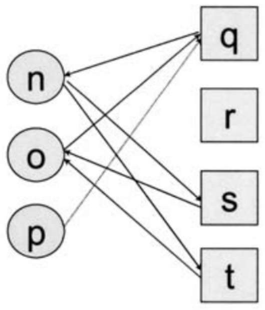


Figure 2.7: A layout for an AGVS, where the segment with arrows denote guide paths and the dotted line denotes a zone boundaries [1].

A working procedure is defined to represent a path which is a sequence of resources. Two digraphs named Working Procedure digraph (D_ω) and Transition digraph (D_T) are introduced to describe all the possible interactions between the jobs and resources [1].

Three algorithms with different restrictive levels on the paths are proposed in the path validation, based on the graphs derived from D_ω and D_T. These can ensure the safety of all the new paths, that is to say, all the enabling new paths assigned to vehicles will lead to no deadlocks. While the vehicles are moving on the paths, the one-step controller takes care of the deadlock avoidance. The method is to check the state that the system will reach after one AGV moving to the next zone. One of the check conditions is whether the D_T of that state contains a cycle that does not include any nodes with infinite capacity. The other condition is whether the next zone is occupied. The vehicle is not allowed to move if either condition holds. The vehicle can only move when none of the conditions hold. With the path validation and one-step control, no path and movement can lead the system to deadlock in the given examples.

Yoo et al. [10] proposed a simple deadlock avoidance algorithm for AGVS based on another graph-theoretic approach. A bipartite digraph was proposed in this article, consisting of a set of vertices and a set of edges. The AGVs and the zones in the layout are both considered as vertices but located in two different columns, as shown in Figure 2.8. Two kinds of edges are defined: one is *requested* edge, from vertices of AGVs to vertices of zones, which represents the AGV has made a request for that zone. In contrast, the other one is *assigned* edge, which represents the zone has already been allocated to that AGV. A path matrix is defined as in Figure 2.8, right, to represent the graph as a $V \times V$ matrix P, where V refers to the vertices in the graph. The value of position $[i, j]$ equals the number of paths from i to j, where i and j are vertices in the graph. When a new edge is inserted or removed, the matrix is recalculated.



$$\begin{array}{c|ccccccc|}
 & n & o & p & q & r & s & t \\
n & 0 & 2 & 0 & 0 & 2 & 1 & 1 \\
o & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
p & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
q & 1 & 2 & 0 & 0 & 2 & 1 & 1 \\
r & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
s & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
t & 0 & 1 & 0 & 0 & 1 & 0 & 0
\end{array}$$

Figure 2.8: An example of the resource allocation graph and corresponding path matrix, where n, o, p represent AGVs and q, r, s, t represent zones.

The existence of cycles in the resource-allocation graph means that the system is not deadlock-free. From the matrix perspective, the sum of diagonal elements of the path matrix being zero is a sufficient condition to say that there are no deadlocks.
The idea of Petri net based control is discussed in some literature, for example in [5], [1], and [10]. In [10], it is argued that the Petri net control is quite complex and time-consuming, since the Petri net model is not a universal solution to all the layouts. In other words, the model has to be modified according to different layouts. In [1], it is also discussed that some of the Petri net based methods are only suitable for a small system like the work in [11]. In [12], it is argued that the main drawback of discrete event system based approaches is that the performance is low in some situations. Wu in [13] proposes a real-time control policy based on coloured-resource-oriented Petri nets, but also argues that this would be computationally heavy for a bigger layout with more routes.
Overall, there are many publications addressing the deadlock avoidance in Flexible Manufacturing System, and particularly in AGV system. However, there is a large gap between the preconditions for those methods and what is actually the case in a realistic AGV system. This makes it very challenging to adapt those methods to the real system. For instance:

- The layout in this thesis is station-intensive and the space for the AGVs are limited.
- The layout is path-compact which means there are very limited alternative routes to follow.

- The division and definition of zones require reconsideration.
- It is challenging to integrate the method with the existing control system.

For these reasons, a new method suitable for the given AGVS is developed in this thesis.

2.4 Summary

In this chapter, some features of the control software package are introduced. Some terms were defined and documents were introduced for better understanding of further chapters. The deadlock was also explained to provide a general idea. A literature review on the deadlock field was summarized.

3 Methods

As mentioned in previous chapters, this thesis work has been carried out in collaboration with AGVE AB. The tests and simulations were conducted in the software package developed by the company, which limited the work somehow. Due to the time limitation, modifying AGVE AB's AGV control system or integrating python scripts with the company's AGV control system are not included in the thesis work. The methods developed in this chapter aim to avoid the deadlock automatically or with as little human involvement as possible.
In the following sections, the deadlock mechanism will be introduced from the software perspective first. Some existing tools that are provided by the company are also introduced in this section. We will discuss two categories of methods; *static* methods, which are check method and zone-control method, and *dynamic* methods, which is an idea inspired from the literature review but not implemented in this thesis.

3.1 Deadlock Mechanism

This section aims to help understand the mechanism of detecting deadlock in the software, which is the foundation of the method. The most common type of deadlock case is shown in Figure 3.1. This type of deadlock always appears first when running simulation on BAM with random orders. The characteristic of the type of deadlock is that two AGVs form a circular wait and one AGV is at station and another AGV is at the lane.
As shown in Figure 3.1, AGV1 stands on the node X134 that AGV2 wants to get, while AGV1 wants to get access to the station ST801 currently occupied by AGV2. This is recognized as a deadlock since AGV1 and AGV2 cannot move further without external help. Related to the conditions mentioned in Section 2.2, the essence is that these two AGVs could be considered as jobs in a circular wait.
To illustrate the situation, Figure 3.2 shows the overlap information of the nodes occupied by AGV1 and AGV2.
AGV2 picked its order at Station ST801 and was going to drop the order at station ST2045, while AGV1 was heading to pick its order at station ST802. AGV2 aimed to move out of the station first following a segment connecting station ST801 and node X37. However, according to the overlap information, AGV2 is asking for access to a number of nodes and stations when moving to node X37, as shown in Figure 3.2.a, including node X134. At the same time, AGV1 is asking for access to a set of nodes and stations when moving into station ST802, as shown in Figure 3.2.b,

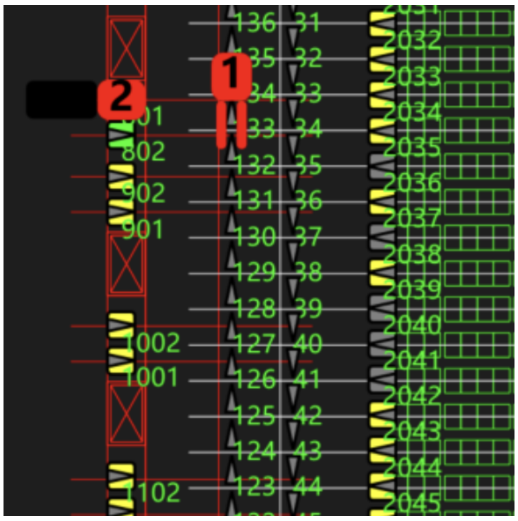
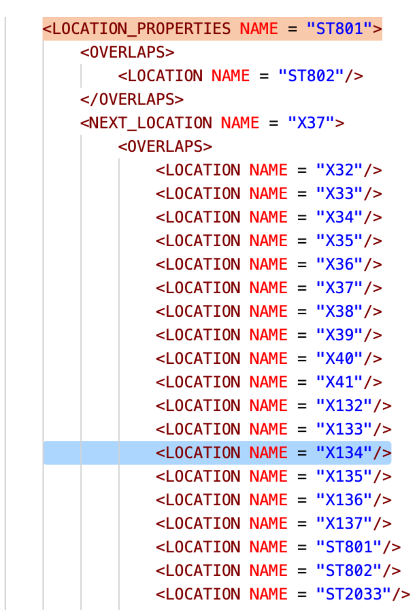


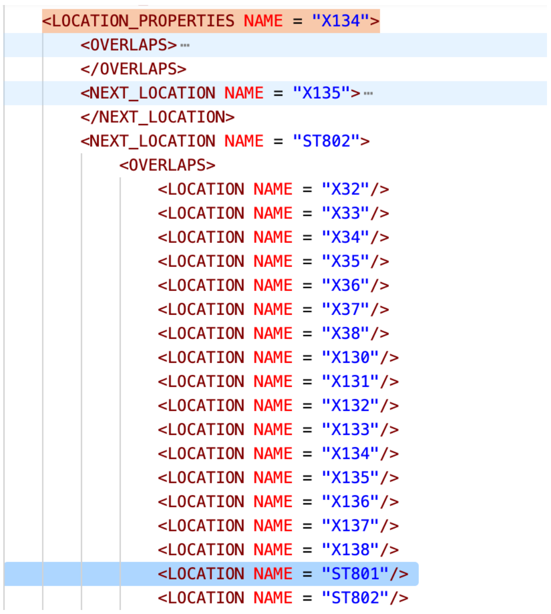
Figure 3.1: The most common type of deadlock situation involving AGV1 and AGV2, as presented in the GUI of the software.

including station ST801. Hence, the two AGVs are involved in a circular wait.



```
<LOCATION_PROPERTIES NAME = "ST801">
    <OVERLAPS>
        <LOCATION NAME = "ST802"/>
    </OVERLAPS>
    <NEXT_LOCATION NAME = "X37">
        <OVERLAPS>
            <LOCATION NAME = "X32"/>
            <LOCATION NAME = "X33"/>
            <LOCATION NAME = "X34"/>
            <LOCATION NAME = "X35"/>
            <LOCATION NAME = "X36"/>
            <LOCATION NAME = "X37"/>
            <LOCATION NAME = "X38"/>
            <LOCATION NAME = "X39"/>
            <LOCATION NAME = "X40"/>
            <LOCATION NAME = "X41"/>
            <LOCATION NAME = "X132"/>
            <LOCATION NAME = "X133"/>
            <LOCATION NAME = "X134"/>
            <LOCATION NAME = "X135"/>
            <LOCATION NAME = "X136"/>
            <LOCATION NAME = "X137"/>
            <LOCATION NAME = "ST801"/>
            <LOCATION NAME = "ST802"/>
            <LOCATION NAME = "ST2033"/>
```

a. Overlap information of Station 801



```
<LOCATION_PROPERTIES NAME = "X134">
    <OVERLAPS>⋯
    </OVERLAPS>
    <NEXT_LOCATION NAME = "X135">⋯
    </NEXT_LOCATION>
    <NEXT_LOCATION NAME = "ST802">
        <OVERLAPS>
            <LOCATION NAME = "X32"/>
            <LOCATION NAME = "X33"/>
            <LOCATION NAME = "X34"/>
            <LOCATION NAME = "X35"/>
            <LOCATION NAME = "X36"/>
            <LOCATION NAME = "X37"/>
            <LOCATION NAME = "X38"/>
            <LOCATION NAME = "X130"/>
            <LOCATION NAME = "X131"/>
            <LOCATION NAME = "X132"/>
            <LOCATION NAME = "X133"/>
            <LOCATION NAME = "X134"/>
            <LOCATION NAME = "X135"/>
            <LOCATION NAME = "X136"/>
            <LOCATION NAME = "X137"/>
            <LOCATION NAME = "X138"/>
            <LOCATION NAME = "ST801"/>
            <LOCATION NAME = "ST802"/>
```

b. Overlap information of Node 134

Figure 3.2: Overlap Information of two nodes (captured from the document OVERLAPINFO.XML).

3.2 Static method

It is observed from the layouts used in this thesis that they exhibit patterns with some common characteristics. First, they are station-intensive and well-arranged. All the stations are close to each other but they are arranged in parallel. Second, the paths are one-way and concentrated between the stations. These observations

lead the authors to come up with the first method.
The first method proposed is named Static Method, and it uses the system documents OVERLAPINFO.XML and SYSTEM_SETUP. To be specific, the static method aims to avoid the deadlock by generating rules automatically in the SYSTEM_SETUP file to manage traffic according to the layout features, which are given in the OVERLAPINFO.XML. In this section, existing tools are described first. The procedure and two methods to implement them automatically is introduced later.

3.2.1 Existing tools

There are already some existing tools developed in AGVE's software package, and two of them are of special interest in this project. One named the *resource check tool* checks whether a certain condition is met before moving forward. The other one is called the *detour tool*, and it modifies the routes of the AGVs under certain conditions.

3.2.1.1 Resource check tool

The resource check tool stipulates that an AGV is allowed to move to a node if another node is free, otherwise it is commanded to wait at the current node. The syntax is:

- <CHECK NODES = "<NODES>" IF_NODES_FREE = "<NODES_FREE>" />
- NODES = A regular expression naming the NodeNames or NodeIds to trigger upon.
- NODES_FREE = The nodes to check if free. This is a regular expression naming the NodeNames or NodeIds to check.

There is also a variant of the resource check tool which contains the AGV's itinerary condition. An AGV is allowed to move to a node if another node is free and the AGV's itinerary includes the required node, otherwise it is commanded to wait at the current node. The syntax is:

- <CHECK NODES = "<NODES>" IF_NODES_FREE = "<NODES_FREE>" WHEN_ITIN_HAS_NODES = "<ITIN_NODES>" />
- NODES = A regular expression naming the NodeNames or NodeIds to trigger upon.
- NODES_FREE = The nodes to check if free. This is a regular expression naming the NodeNames or NodeIds to check. This check is conditioned by the ITIN_NODES parameter.
- ITIN_NODES = The nodes to check if included in itinerary. This is a regular expression naming the NodeNames or NodeIds in itinerary.

3.2.1.2 Detour tool

The detour tool can make an AGV detour to another itinerary if it gets stuck right before its destination. The syntax is:

- <REPLACE_ATTR OBJECT = "Link" NAMES = "<NODES_NODES>" DetourMethod = "DetourToNodeIfBlocked(<NODES_DETOUR>, ,NODES: <NODES_BLOCK>)" />
- NODES_NODES = A regular expression naming the NodeNames or NodeIds to trigger upon. The two NODES represent the AGV's present location and next location respectively.
- NODES_DETOUR = A regular expression naming the NodeNames or NodeIds to detour.
- NODES_BLOCK = A regular expression naming the NodeNames or NodeIds to trigger upon. These nodes block the AGV to move to the next location.

3.2.1.3 Advanced tools: function check

There are also some programmed functions that could be used to add the AGV's or other AGV's itinerary information in the resource check tool. The syntax is:

- <CHECK NODES = "<NODES>" IF_METHOD_TRUE = "CheckIfNot-Blocked(<Keywords>:<NODES_ITIN>, NODE:<NODES_BLOCK>)" />
- NODES = A regular expression naming the NodeNames or NodeIds to trigger upon.
- NODES_ITIN = The nodes to check if included in itinerary. This is a regular expression naming the NodeNames or NodeIds in itinerary.
- NODES_BLOCK = A regular expression naming the NodeNames or NodeIds to trigger upon. This node block the AGV move to the next location.
- Keywords = Different functions which will be listed as below
- MY_ITIN/MY_ITINS: if AGV has node in itinerary or standing on node
- UNLESS_MY_ITIN/UNLESS_MY_ITINS: do not detour if the AGV has a node in its itinerary or is standing on node
- ITIN/ITINS/OTHER_ITIN/OTHER_ITINS: check blockings for all other AGVs that have node in itinerary or standing on node
- UNLESS_ITIN/UNLESS_ITINS: don't detour for all other AGVs that have node in itinerary or standing on node
- MAX_AREA/MAX_AREAS/MAX_IN_AREA/MAX_IN_AREAS: check if area occupied by specifying max AGVs in area. Max number of AGVs decide if occupied or not. Own AGV is not accounted for

These two tools' rules are used in the static method. AGVE's software package has lots of tools for engineers to write deadlock avoidance rules manually based on experience. Engineers need to understand lots of different tools and do repetitive heavy manual work. The static method aims to generate deadlock avoidance rules automatically and only use these two tools.

3.2.2 Procedure

The steps to apply the static method is shown in the flowchart of Figure 3.3. The layout can be divided into different patterns with different features. The first step is to analyze the layout and summarize the patterns based on the observations and experience. A layout analysis tool is developed to detect the problematic stations where deadlock involved two AGVs may occur. Then, existing tools are used to avoid the most common type of deadlock in the layout. It should be noted that the rules need to be generated automatically based on the overlap information. A test that repeat the scenario of the most common type of deadlock is run to verify that the rules can avoid the deadlock if there is no deadlock happens. After verifying that the rules can avoid the most common type of deadlock, a new type of deadlock is generated and it is avoided by automatically generated rules. At this time, it is possible that the new rules have contradictions with the old rules, which could result in deadlock. A long test with randomly orders is run at the end of the procedure.

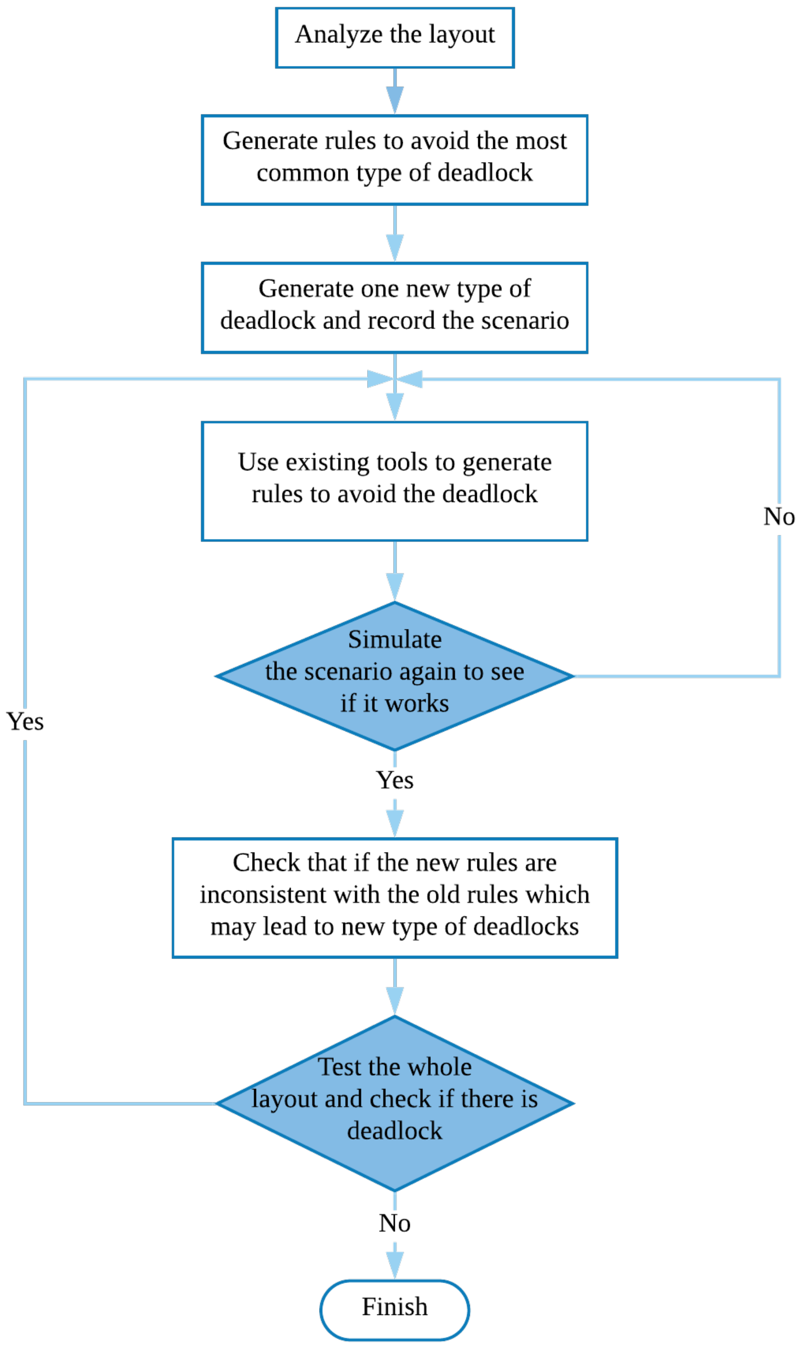

Analyze the layout
Generate rules to avoid the most common type of deadlock
Generate one new type of deadlock and record the scenario
Use existing tools to generate rules to avoid the deadlock
Simulate the scenario again to see if it works
No
Yes
Check that if the new rules are inconsistent with the old rules which may lead to new type of deadlocks
Test the whole layout and check if there is deadlock
Yes
No
Finish

Figure 3.3: Flowchart of static method's procedure.

3.2.3 Layout analysis tool

Inspired from the literature, a zone-control method is developed to prevent the AGVs get into the circular wait state so that no deadlock will happen.
The first step is analyzing the layout to find the problematic locations where there is a possibility for AGVs to get in deadlock situations. An algorithm to do that is

proposed. The algorithm consists of two pieces of python code. Algorithm 1 parses the relation among the nodes in the file "OVERLAPINFO.XML" and stores the data in a dictionary for further use. Then, Algorithm 1 can tell which stations are problematic by comparing the next-location and the overlap in the file "OVERLAPINFO.XML". With these two conditions listed below, the station ST B is defined as a problematic station:

1. A node named Node A exists in the layout where there is a directional link starting from it and ending on a station ST B, which means that an AGV can move into ST B from Node A.
2. There are one or more nodes (Node C, Node D, . . .) in the overlap zone of Node A connected to ST B by a directional link starting from ST B and ending at Node C, Node D, etc., which means that an AGV can move to Node C, or Node D, etc., from ST B.

```
Parse the OVERLAPINFO.XML ;
Save all the overlapped locations of the node as all_des_overlap
Initialize loc_list to all the nodes in the OVERLAPINFO.XML
Initialize pro_station to empty;
for loc in loc_list do
    if loc is node on the lanes(not stations) then
        Save all the next locations of the loc as next_loc.
        for the next_loc which is a station do
            Store the all the nodes in the next location of next_loc as
             other_next_loc.
            if other_next_loc is a node on the lanes then
                if other_next_node in the all_des_overlap of loc then
                    add next_loc to pro_station
                end
            end
        end
        Save the pro_station;
    end
end
Compare the problematic stations(pro_station) with all the stations in the
 layout and show the difference;
```

Algorithm 1: Given the OVERLAPLAP.XML, the algorithm can identify the problematic stations in the layout.

3.2.4 Check method

The basic idea is checking whether all the overlapping locations of this location are not occupied before moving into this location to prevent the most common type of deadlock. If the overlapped locations are not occupied by other AGVs, deadlock will not occur and the AGV can go to this location. For instance, Figure 3.4 shows the resource check rules of Node X1. The syntax <CHECK NODES = "X1"

IF_NODES_FREE = "X3" WHEN_ITIN_HAS_NODES = "X2" refers to the command: Check if X3 is free before the AGV moving to node X1, when X2 is in the itinerary of the AGV. The AGV can only move in if X3 is free, otherwise it waits at its current node.

The resource check rules are generated according to the information in Figure 2.2.b and the algorithm to automatically generate the rules is shown in Algorithm 2. It should be noted that when the resource check tool is used, the information of the next location should be included in the syntax. The reason is that the overlapped locations vary according to different next locations, as shown in Figure 2.2.b.

```
for nodes or stations in first_branch do
    Save the LocationName of the nodes or stations;
    for overlaps or next_location in the second_branch do
        if second_branch.tag is OVERLAPS then
            for LocationName in third_branch do
                Save the OverlapsName of the overlaps;
                Output <CHECK NODES = "LocationName"
                  IF_NODES_FREE = "OverlapsName" />;
            end
        end
        if second_branch.tag is NEXT_LOCATION then
            Save the NEXT_LOCATION_NAME;
            for LocationName in third_branch do
                for overlaps in forth_branch do
                    Save the OverlapsName of the overlaps;
                    Output <CHECK NODES = "LocationName"
                      IF_NODES_FREE = "OverlapsName"
                      WHEN_ITIN_HAS_NODES =
                      "NEXT_LOCATION_NAME"/>;
                end
            end
        end
    end
end
```

Algorithm 2: Check method's algorithm to automatically generate the rules to avoid the most common type of deadlock.

```
<CHECK NODES = "X1" IF_NODES_FREE = "X2" />
<CHECK NODES = "X1" IF_NODES_FREE = "X3" />
<CHECK NODES = "X1" IF_NODES_FREE = "X4" />
<CHECK NODES = "X1" IF_NODES_FREE = "ST2001" />
<CHECK NODES = "X1" IF_NODES_FREE = "ST2002" />
<CHECK NODES = "X1" IF_NODES_FREE = "X1" WHEN_ITIN_HAS_NODES = "X2"/>
<CHECK NODES = "X1" IF_NODES_FREE = "X2" WHEN_ITIN_HAS_NODES = "X2"/>
<CHECK NODES = "X1" IF_NODES_FREE = "X3" WHEN_ITIN_HAS_NODES = "X2"/>
<CHECK NODES = "X1" IF_NODES_FREE = "X4" WHEN_ITIN_HAS_NODES = "X2"/>
<CHECK NODES = "X1" IF_NODES_FREE = "X5" WHEN_ITIN_HAS_NODES = "X2"/>
<CHECK NODES = "X1" IF_NODES_FREE = "ST2001" WHEN_ITIN_HAS_NODES = "X2"/>
<CHECK NODES = "X1" IF_NODES_FREE = "ST2002" WHEN_ITIN_HAS_NODES = "X2"/>
<CHECK NODES = "X1" IF_NODES_FREE = "ST2003" WHEN_ITIN_HAS_NODES = "X2"/>
<CHECK NODES = "X1" IF_NODES_FREE = "X1" WHEN_ITIN_HAS_NODES = "ST3000"/>
<CHECK NODES = "X1" IF_NODES_FREE = "X2" WHEN_ITIN_HAS_NODES = "ST3000"/>
<CHECK NODES = "X1" IF_NODES_FREE = "X3" WHEN_ITIN_HAS_NODES = "ST3000"/>
<CHECK NODES = "X1" IF_NODES_FREE = "X4" WHEN_ITIN_HAS_NODES = "ST3000"/>
<CHECK NODES = "X1" IF_NODES_FREE = "ST2001" WHEN_ITIN_HAS_NODES = "ST3000"/>
<CHECK NODES = "X1" IF_NODES_FREE = "ST2002" WHEN_ITIN_HAS_NODES = "ST3000"/>
<CHECK NODES = "X1" IF_NODES_FREE = "ST3000" WHEN_ITIN_HAS_NODES = "ST3000"/>
```

Figure 3.4: Check method's rules of Node X1.

3.2.5 Zone-control method

The basic idea of the Zone-control method is to prevent the AGVs to get into a circular wait. That is to say, ensuring the AGV has enough space to go into and get out of the station. The rules stop other AGVs on the main lane getting too close to the target AGV. A straightforward method is proposed as generating the resource check rules for the problematic stations. These rules ensure that there is enough space for an AGV in the station to exit, no matter how many AGVs are moving around. With these rules, deadlocks involving three or more AGVs can also be avoided as they cannot form a circular wait situation.
Different from the check method, in the overlap information, only the overlaps information of stations is checked and at this time the information of the next location is not important. For instance, Figure 3.5 shows the resource check rules of station ST201. The resource check rules are generated according to the information in Figure 2.2.b and the algorithm to automatically generate the rules is shown in Algorithm 3. If one station is occupied by an AGV, the zone that contains all locations overlapped with the station is locked. No AGV can move to these nodes until the station checked is free. With this method, other AGVs will wait outside the locked zone until the AGV in the station gets out. If the zone-control strategy is directly applied on the main lane's nodes, it could affect the AGV's normal moving on the main lane.

```
<CHECK NODES = "X2" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "X3" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "X4" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "X5" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "X6" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "X7" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "X8" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "X9" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "X163" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "X164" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "X165" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "X166" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "ST201" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "ST202" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "ST2003" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "ST2004" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "ST2005" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "ST2006" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "ST2007" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "X162" IF_NODES_FREE = "ST201"/>
<CHECK NODES = "ST2002" IF_NODES_FREE = "ST201"/>
```

Figure 3.5: A part of the rules of the zone-control method. Station 201 was checked in this case and the former nodes are locked.

```
for stations in first_branch do
    Save the StationName of the stations;
    for overlaps or next_location in the second_branch do
        if second_branch.tag is NEXT_LOCATION then
            for LocationName in third_branch do
                for overlaps in forth_branch do
                    Save the OverlapsName of the overlaps;
                    Output <CHECK NODES = "OverlapsName"
                     IF_NODES_FREE = "StationName"/>;
                end
            end
        end
    end
end
```

Algorithm 3: The zone-control method's algorithm to automatically generate the rules to avoid the most common type of deadlock.

3.2.6 Further solutions

After avoiding the most common type of deadlock, there are still some different types of deadlock. The simple zone locking strategy is not enough to solve all the deadlocks in such a complex system. It can cause some problems since a zone that is bigger than actually needed is locked. The reason is that the strategy is only effective for the situation when one AGV has already occupied one station and another AGV passed by. The nodes on the main lane are not taken into consideration. New rules to manage the traffic in this area are necessary to be developed.

Since the paths or to say the lanes are one-way, which means no AGVs can move towards each other on the same lane, the complexity of generating the rules is reduced. However, problems still arise when two parallel lanes with different direction get close together. This can be discussed in two situations, depending on whether these two lanes overlap with each other or not.

The first situation is that two lanes do not overlap with each other, which means two AGVs moving on the parallel lanes can pass each other. Two possible deadlocks could happen.

- Firstly, the situation where two close AGVs move on parallel lanes and head to two nearby stations at the same time may cause a problem. With the current rules, no syntax can check the destinations or itineraries of both AGVs. Thus, sending one of the AGVs to other nodes is the only solution to this situation. The algorithm proposed aims to generate detour rules for this situation based on the existing syntax. The idea is to send away the AGVs heading to the nearby stations when certain nodes on the other lane are occupied. Algorithm 4 shows a way to automatically generate detour rules.
- Another situation is when two AGVs get out of nearby stations at the same time and head to different lanes. One solution is to stop one of the AGVs

```
for stations in first_branch do
    Save the NodeName of the nodes;
    for next_location in the second_branch do
        if second_branch.tag is NEXT_LOCATION then
            Save Next_Location_Name of stations;
            Calculate the detour destination:
              $DetourNode \rightarrow NodeName + variable\_number$;
            for LocationName in third_branch do
                for overlaps in forth_branch do
                    Save the OverlapsName of the overlaps;
                    Output <REPLACE_ATTR OBJECT = "Link" NAMES =
                      "NodeName_Next_Location_Name" DetourMethod =
                      "DetourToNodeIfBlocked(DetourNode,
                      ,NODES:OverlapsName)" />;
                end
            end
        end
    end
end
```

Algorithm 4: Algorithm to automatically generate rules to avoid the situation that two AGVs go into nearby stations at the same time.

and let the other one leave, but there are no such resource check rules in the system. One feasible solution is to avoid the situation where two AGVs go into nearby stations simultaneously. If two AGVs' routes are staggered, there is no deadlock when two AGVs go into nearby stations at the same time. Based on Algorithm 4, adding more overlapped nodes could avoid two AGVs going into nearby stations simultaneously.

Another feasible solution is to send one of the AGVs in the opposite direction or to say a different lane, which commands these two AGVs heading to the same directions so that the deadlock is avoided. The algorithm is shown as Algorithm 5.

The second situation is that two lanes are overlapped with each other, which means two AGVs moving on parallel lanes cannot pass each other. In this case, the zone locking strategy is not enough and can only solve a small part of the problem, as further discussed in Chapter 4.

Therefore, more CHECK rules are required to ensure that the AGVs cannot move towards each other on the lanes. An initial idea is making use of the tool CHECK, OTHER_ITIN, NODES mentioned in Section 3.2.1 to control the traffic in and out of this zone, which means other AGVs' current positions and itineraries are checked before one AGV asking for moving to a node. However, this function could confuse the system when there are more than two vehicles since all the information of other AGVs are considered in this function and the OTHER_ITIN and NODES may belong to different AGVs so that it is impossible to use this piece of information to know the routines of other AGVs. Even though the checked information exactly

```
Parse the $OVERLAPINFO.XML$;
Divide *stations* into *right_stns* and *left_stns* according to the position
  related to the lanes;
for *station in this area* do
    Divide *nodes* into *up_exit_nodes* and *down_exit_nodes* according to the
      exit node's relative position to the station
    if *stn in left_stns* then
        if *exit_node in up_exit_nodes* then
            Save the overlapped nodes of the *exit_node* as *exit_up_overlap*;
            Initialize *problem_stn* to empty list;
            Set the detour node as *detour_node*
            for *another_stn in right_stns and right_stns* do
                if *another_exit_node in down_exit_nodes* then
                    Save the overlapped nodes of the *another_exit_node* as
                      *exit_down_overlap*
                    if *exit_up_overlap overlapped with exit_down_overlap*
                      then
                        add *another_stn* to *problem_stns*
                    end
                end
            end
            *write_stns = problem_stns- overlapped_stns*
            Write the rules using the *sta*, *exit_node*, *write_stas*, *detour_node*
        end
    end
    if *stn in right_stns* then
        do the same as the case *sta* in *left_stns*
    end
end
```

Algorithm 5: Another algorithm to automatically generate the rules to avoid the situation that two AGVs go out of nearby station at the same time.

belongs to one AGV in some certain part of the layout, the itinerary of the AGV asking for entering is still required to make sure they will not meet on the lanes. Based on this analysis, the method implemented was dividing this area into two zones due to the limited space between the stations on both sides, which are upper zone and lower zone. Then the rules consist of two parts. The first part rules are created to prevent the entering to the upper zone/lower zone when lower/upper zone is occupied. The second part rules are responsible for the deadlocks situation caused by exiting the stations. For instance, one AGV occupying a station in the upper zone wants to exit to the lower zone, which is occupied by other AGVs. The rules will command it to detour to the upper zone and exit from above. With these two kinds of methods, the vehicles can not move towards each other on the lanes. Furthermore, it is possible to set a maximum number of AGVs existing in the zone in the system. By setting the maximum number as 1, no other vehicles will enter

the upper zone or lower zone when there is already one AGV in it.
Overall, the rules can avoid some frequent types of deadlocks in these two situations. There are still other deadlock types possible, these are analyzed following the procedure shown in Chapter 4.

3.3 Dynamic method

The Dynamic method aims to develop an online tool that can detect possible a deadlock a few steps ahead and then avoid the deadlock either by stopping or changing the speed of one of the AGVs. The detection of deadlock is accomplished through verifying if there is a circular wait for the target AGVs. If so, a deadlock will occur a few steps later and the command to stop or change speed should be sent to one of the AGVs. Furthermore, the methods to avoid deadlocks in advance may involve some optimization design such as which one to stop, how many steps ahead or which speed is more sustainable.
The basic condition for dynamic methods is that the online software can interact with the AGV control system and the AGV's physical data such as location, itinerary and speed. A socket connection among online software, AGV control system and the states of AGVs is needed. The speed of the AGV is considered to be constant. The stop and start process's time losses are not taken into account since it is negligible in the whole process and using constant speed value simplifies the calculations.

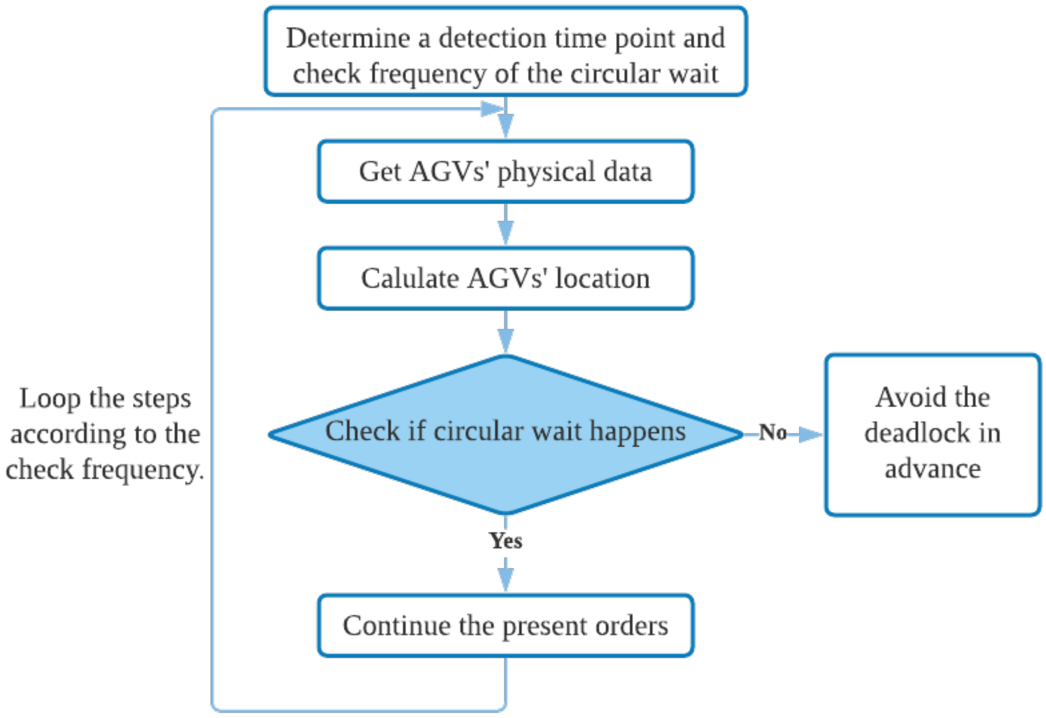

Determine a detection time point and check frequency of the circular wait
Get AGVs' physical data
Calulate AGVs' location
Check if circular wait happens
No
Avoid the deadlock in advance
Yes
Continue the present orders
Loop the steps according to the check frequency.

Figure 3.6: Flowchart of the dynamic method's procedure.

The procedure for the dynamic method is shown in the following flowchart. The first step is to determine a detection time point in the future and the circular wait check frequency. If the detection time point is 10 seconds and the check frequency is one check per 3 seconds, the online software would calculate the locations of AGVs and check if there is a circular wait after 10 seconds every 3 seconds. These values could be changed according to the features of layouts. With the physical data of the AGVs, the location of AGVs at the time point could be calculated since the itinerary, the distance between every location, and the speed of the AGVs are known. If circular wait does not happen at the time point, the AGVs could follow the order. Otherwise, avoidance measures would be taken.
In the case of two AGVs, the simplest avoidance measure is stop one AGV. This solution is not the best solution for layouts with more AGVs since the stopped AVG occupies the lane, which will affect the efficiency. There are more complicated ways to avoid the deadlocks. Speeding up or slowing down the AGVs could increase the efficiency of the system. In addition, path changing is also an option.

3.4 Summary

This chapter introduced the deadlock mechanism of the company's system. Static methods include the procedure of avoiding deadlocks in different layouts. Check method, zone-control method and different types of deadlock's solution were came up to avoid the deadlocks offline. The dynamic method provides a solution that could detect and avoid the deadlocks online. With the basic conditions, this method could provide AGVE AB a new design idea of AGV control system in the future.

4
Results

In Chapter 3, a work procedure is proposed to study deadlocks in the AGV control system. Two methods were developed including some algorithms to generate rules. In this chapter, two layouts are studied and tested by following the procedure. Their OVERLAPINFO.XML file was used to generate the rules automatically. The results of the layout analysis and the methods to avoid the most common type of deadlock are described. Besides, the first layout's further different types of deadlock's situations are summarized. Some deadlocks that are caused by the first layout's special area's characteristics and need to be manually avoided are also introduced. Moreover, the difference between Layout 1 and Layout 2 and what influence it brings are discussed at the end.

4.1 Layout 1

Layout 1 is a project handling meat containers in cooperation with Vanderlande, shown in Figure 4.1. There is only one pair of lanes to follow. A pair of lanes refers to two nearby lanes with different directions, one upward and the other one downwards in this case. The AGVs can pass by each other when moving forward. Two types of stations are lined up on both sides of the road. The AGVs pick up the materials from the right-side stations and drop it on the left-side stations. The right-side grid is storage shelves. It is worth noting that it is possible for the AGVs to move inside the right-side storage shelves to pick up goods as the handling progresses. At the top of the layout, there is a horizontal lane (along X356, X355, X354, X353) connecting the battery stations (ST1801, ST1802), the pair of lanes and washing machine (the white object at right of X3000). This horizontal lane is bidirectional and only one AGV can go through this lane at a time.

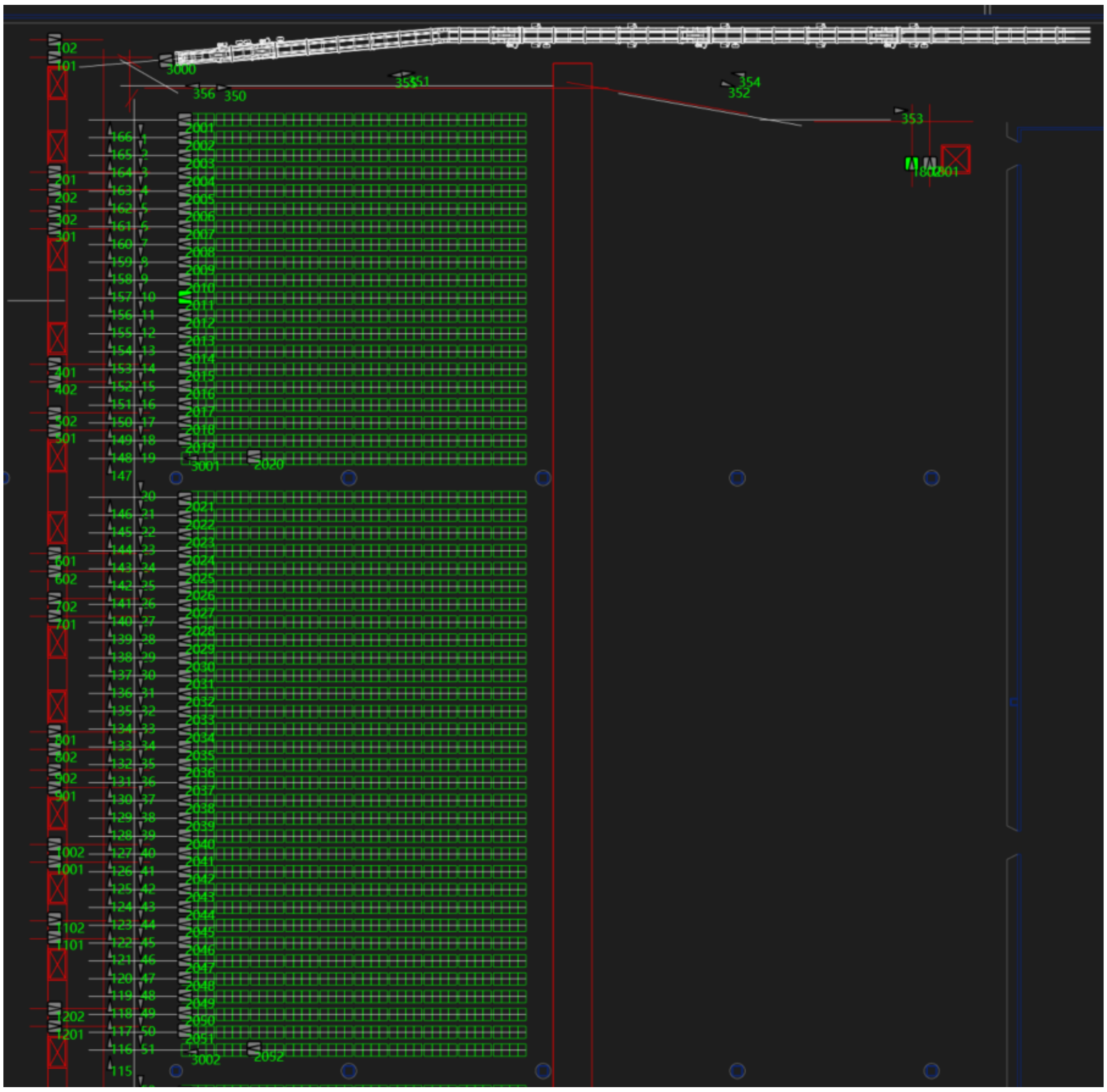


Figure 4.1: The layout for the Vanderlande project.

4.1.1 Layout analysis

As the first step of the procedure, the layout was analyzed. The main pairs of lanes are located between the stations for picking and dropping respectively. The layout analysis tool mentioned in Section 3.2.3 was implemented on the first layout. The result showed that there were 125 stations in total and only two of them are *not* problematic, ST1801, ST1802. Thus, the check rules are generated for rest of the stations. Layout 1 was divided into three different patterns for analysis.

1. The pair of lanes with stations on both sides: This pattern's nodes and stations have similar physical location. The overlap information has similar design laws, which are designed by AGVE AB and the customer. Automatically generated rules are suitable for this area.
2. The area connected with the horizontal lane: The characteristic of this pattern is that the horizontal lane can hold one AGV at a time and the overlaps

information for this pattern is different with the first pattern. The deadlock avoidance rules need to be written manually based on experience.

3. The nodes and stations near the gap at the middle of the layout: This pattern's overlap information is a little different from the first pattern, the automatically generated rules cannot avoid all the deadlocks.

4.1.2 Simulation results

The simulation was carried out in the BAM. The results shown in the following sections are also based on it. It is worth noting that the orders are randomly generated for testing the whole layout. When testing a certain pattern, a piece of script will be used to generate the orders in this certain area. Furthermore, to check whether a certain type of deadlock is avoided, the AGVs can be manipulated.

4.1.2.1 Check method

The Check method avoids the most common type of deadlock and reduces the incidence of the deadlock. But there is a specific type of deadlock that is hard to solve under existing conditions. The common characteristic of the deadlock is that AGV1 is going to a station, AGV2 is going to a station that is near the destination of AGV1. AGV2 waits at a node or station due to the check rules to let AGV1 go into the destination. Once AGV1 arrives at the destination, AGV2 is free to go. There is a deadlock happening because the destination of AGV2 is near the station occupied by AGV1 and AGV1 is assigned a new task and wants to go out of the station simultaneously and at this time AGV2 blocks AGV1.
Figure 4.2 shows the deadlock situation: AGV1 is at ST2015 and wants to go to X17 (overlapped with X151), and AGV2 is at X151 and wants to go to X152 (overlapped with ST2015) and ST502. The forming process is that at first, AGV1 and AGV2 are both on the left main lane, AGV1's itinerary are X154, X155, ST2015 and during this process, AGV2 waits at X150 due to the collision avoidance system. When AGV1 arrives in ST2015, AGV2 is free to go from X150 to X151 since ST2015 is not checked due to the check method's rules of Node X151. The rules are attached in Figure A.2 in Appendix A. At this time, the deadlock happens.

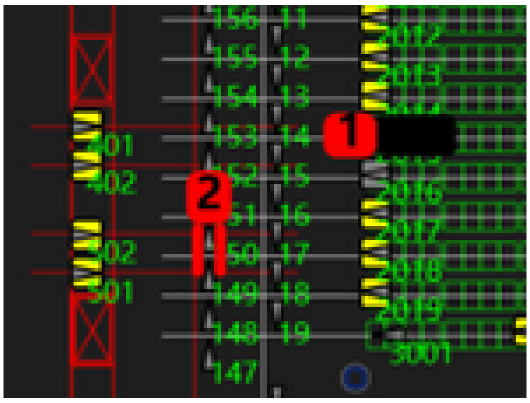


Figure 4.2: Check method's deadlock situation.

If AGV1 wants to go from ST2015 to X155, X151 should be free, but because AGV2 occupies X151 there is no chance for AGV1 to detour to X155. Another possible solution is to detour AGV2 to ST2018, but the solution does not have universality for the whole layout. To solve this problem, the overlap information could be expanded. For example, add some nodes and some stations such as X151 in order to let AGV2 stay at X150 and AGV1 get out of station. Another way is to avoid assigning tasks to AGVs of which the destination stations are close to each other. But it is hard to implement this method automatically.

4.1.2.2 Zone-control method

The Zone-control method solves the most common type of deadlock and reduces the deadlock's incidence. And this method does not cause unsolvable deadlock. There are still some types of deadlock that need to be solved, though.

The first type is that two AGVs want to get into two nearby stations at the same time. For example, Figure 4.3 shows that two AGVs go into stations that are close to each other. AGV1 is at X19 and wants to go to ST502 (overlap with X152), AGV2 is at X152 and wants to go to ST2017 (overlap with X19). The forming process is as: At first, AGV1 is at X16 and AGV2 is at X150, AGV1's itinerary is X16, X17, X18, X19, X502 and AGV2's itinerary is X150, X151, X152, ST2017. At this time, the deadlock happens.

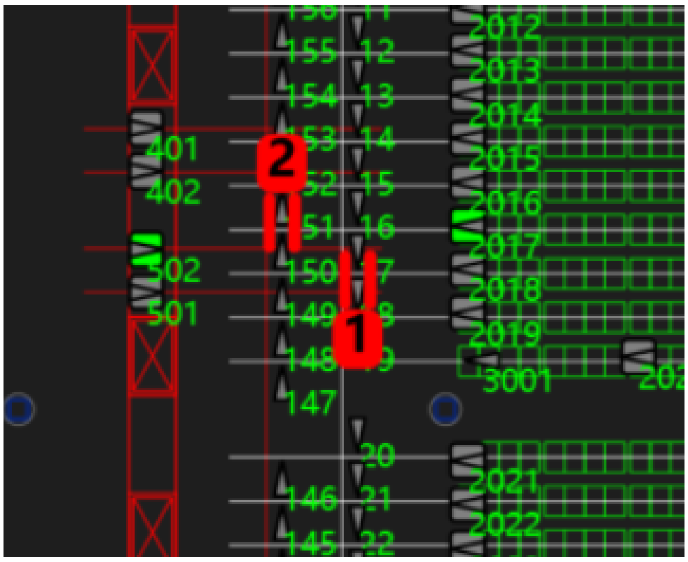


Figure 4.3: Zone-control method's deadlock situation: two AGVs go into stations that are close to each other.

A detour method is used to solve this type of deadlock:
<REPLACE_ATTR OBJECT = "Link" NAMES = "X19_ST2019" DetourMethod = "DetourToNodeIfBlocked(27, ,NODES:152)" />.
This means that if AGV1 wants to go from X19 to ST2019 and node X152 is occupied, AGV1 detours to X27. When AGV1 continues to move forward, AGV2 could go into ST2017 and AGV1 will turn around and try to go into ST502 again.

The second type occurs when two AGVs want to get out of two stations at the same time and their next locations are nearby. For example, Figure 4.4 shows that two AGVs go out of stations at the same time and the next nodes are close to each other. AGV1 is at ST401 and wants to go to X17 (overlap with ST2019 due to lock-zone method), AGV2 is at ST2019 and wants to go to X151 (overlap with ST401 due to lock-zone method). The forming process is: At first, AGV1 is at X152 and AGV2 is at X17. AGV1's itinerary is X152, X153, X154, X401, X17 and AGV2's itinerary is X17, X18, X19, ST2019, X151. At this time, the deadlock happens.

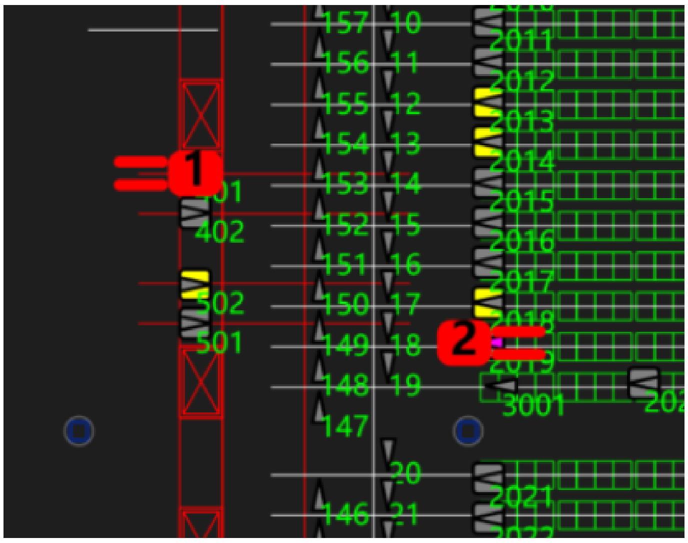


Figure 4.4: Lock-zone method's deadlock situation: two AGVs go out of stations at the same time and the next nodes are close to each other.

The forming process of this type of deadlock is: Two AGVs go into two stations which are close to each other at the same time. Unlike the previous type of deadlock, there is no deadlock when two AGVs go into two nearby stations at the same time, which is because the routes are staggered. To solve this type of deadlock, a detour method is used to prevent two AGVs to go into two nearby stations at the same time.
<REPLACE_ATTR OBJECT = "Link" NAMES = "X155_ST401" DetourMethod = "DetourToNodeIfBlocked(162, ,NODES:19)" />.
This means that if AGV1 wants to go from X154 to ST401 and node X19 is occupied, AGV1 detours to X162. When AGV1 continues to move forward, AGV2 could go into ST2019 and get out of ST2019 successfully and AGV1 will turn around and try to go into ST401 again.
This solution cannot avoid the case that an AGV arrives at the station from the right-side storage shelves. To avoid this situation, when AGVs go to the station from storage shelves, a check rule needs to be added:
<CHECK NODES = "ST2019" IF_NODES_FREE = "ST401" />.

After avoiding these two types of deadlocks and testing with 3 AGVs operating in the layout for 24 hours, the main part of layout is regarded as deadlock-free. The top part of the layout has different features compared to the main part, so the lock-zone method and the detour rules cannot solve all the deadlock in this part. There are some special deadlocks that need to be manually avoided by adding check or detour rules, or deleting some useless rules that are generated by the automatic generation method but can cause deadlock at a certain part. Because the main purpose of this thesis is to automatically generate rules to solve the deadlock, only three special cases and solutions are introduced below.

Figure 4.5 shows the first case. AGV2 is at X356 and wants to go to X1 (overlap with X166), AGV3 is at X166 and wants to go to X350 (overlap with X356). The forming process is: At first, AGV2 comes from the horizontal lane which is bidirectional and only one AGV can go through this lane at a time, AGV2's itinerary is X356, X1. AGV3 is waiting at X166 until the horizontal lane is available. AGV1's itinerary is X166, X350. At this time, the deadlock happens. The reason for this deadlock is node X356 is not a station, and the other AGVs do not leave enough space for AGV2 to go from X356 to X1. So, a check rule (<CHECK NODES = "X166" IF_NODES_FREE = "X356"/>) is added to avoid this situation to let other AGVs wait at X165.

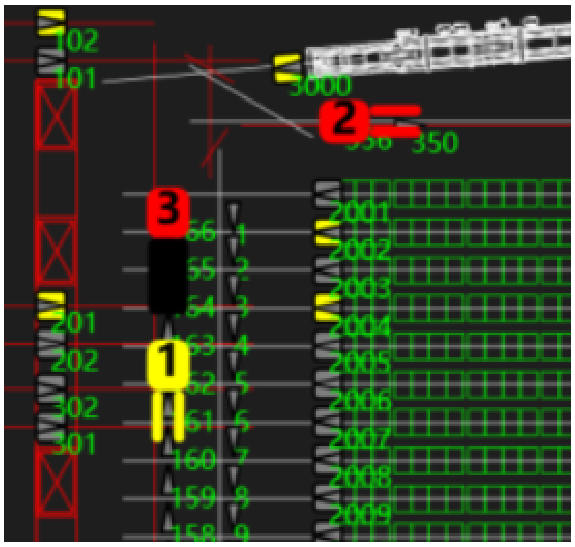


Figure 4.5: Special deadlock which can be solved by adding check rules.

Figure 4.6 shows the second case. AGV2 is at X355 and wants to go to X356 (overlap with ST2001, due to lock-zone method), AGV1 is at ST2001 and wants to go to X350 (overlap with X355). The forming process is that at first, AGV1 is at X2 and goes into ST2001 and at this time, AGV2 comes from the horizontal lane which is bidirectional. AGV1's itinerary is X2, ST2001, X350. AGV2's itinerary is X355, X356. At this time, the deadlock happens. The reason for this deadlock is AGV1's next location conflicts with AGV2's location. Therefore, a detour rule (<REPLACE_ATTR OBJECT = "Link" NAMES = "ST2001_X350" DetourMethod = "DetourToNodeIfBlocked(3, ,NODES:355)" />) is added to avoid this situation to let the other AGV detour to node X3.

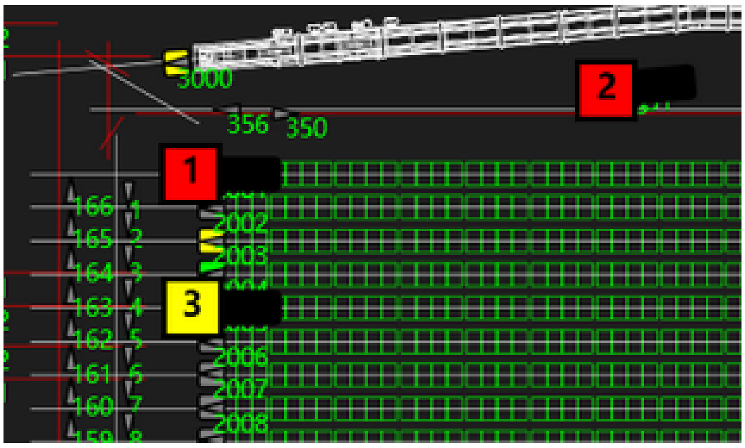


Figure 4.6: Special deadlock that can be solved by adding detour rules.

Figure 4.7 shows the third case. AGV1 is at ST101 and wants to go to X1 (overlap with AGV2), AGV2 is at X1 and wants to go to X2 (overlap with ST101, due to lock-zone method). The forming process is: At first, AGV1 is at 2004, AGV2 is at 356. AGV1 is waiting for AGV2 to get out of horizontal lane. When AGV2 arrives X1, it stops to wait for AGV1 getting out of 2004 because of lock-zone method. Then AGV1 heads to ST101 and AGV2 waits at X1 due to lock-zone method. When AGV1 wants to get out of ST101, X1 is already occupied. AGV1's itinerary is ST2004, X166, ST101, X1. AGV2's itinerary is X356, X1, X2. At this time, the deadlock happens. The reason for this deadlock is that the automatically generated rules for ST101 did not take the situation where AGV at X356 could go to X1 into consideration. Therefore, the lock-zone method for ST101 is deleted to avoid this situation.

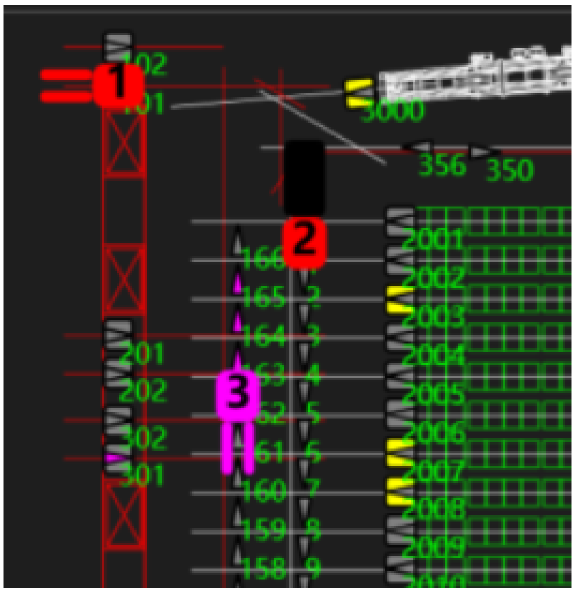


Figure 4.7: Special deadlock that can be solved by deleting check rules.

4.1.3 Discussion

With the rules automatically generated by the zone-control method and manual work, the Layout 1 has no more deadlocks after a long time experiment with three AGVs. The orders were randomly generated all over the layout.

The check method failed on this layout. The reason is summarized as the limitation of layout design; to be specific, the stations are close to each other and there are many same nodes existing in the overlap information of the nearby stations. This leads to an unsolvable deadlock. This method is more suitable for a layout with expanded overlap information or larger station spacing.

The zone-control method successfully avoided the most common type of deadlock in this layout. The imperfection is the low efficiency problem arises when the layout contains more AGVs. If an AGV is occupying a station, the pair of lanes outside the station are locked and other AGVs need to wait outside the zone. Although as long as the AGV in the station takes no actions until other AGVs passed by the station, there is no deadlock or collision. The method does not allow the other AGVs to pass because there is no resource check tool to judge whether the AGV plans to get out of the station or not.

Besides the most common type of deadlock, there were also two more types of deadlock. One type is where two AGVs want to go into nearby stations from different lanes, shown in Figure 4.3. The other one is where two AGVs want to leave the nearby stations at the same time, shown in Figure4.4. The strategy to avoid these two types of deadlocks is to avoid two AGVs going into nearby stations simultaneously. Only the Algorithm 4 is tested in this layout. And further tests on different layouts may be required to verify the feasibility of the strategy.

4.2 Layout 2

layout 2 is a project in cooperation with Linde Gas, shown in Figure 4.8. This one is more complex compared to the first layout, since there are 12 lanes in total forming a transportation network. The picking stations are distributed in the middle of the vertical lanes, while the dropping stations are located at the bottom. A battery charging station is located on node BATT1003. Three horizontal lanes connect each vertical lanes. It is worth noting that AGVs get stuck when they try to pass by each other in opposite directions on the vertical pairs of lanes in the middle, which is a quite different case from Layout 1. AGVs can still pass by each other if they move in opposite directions on the bottom pair of lanes. Three single lanes are located on both sides of the layout. The left one is a single lane with stations on both sides. The other two are on the right side and they have the same direction.

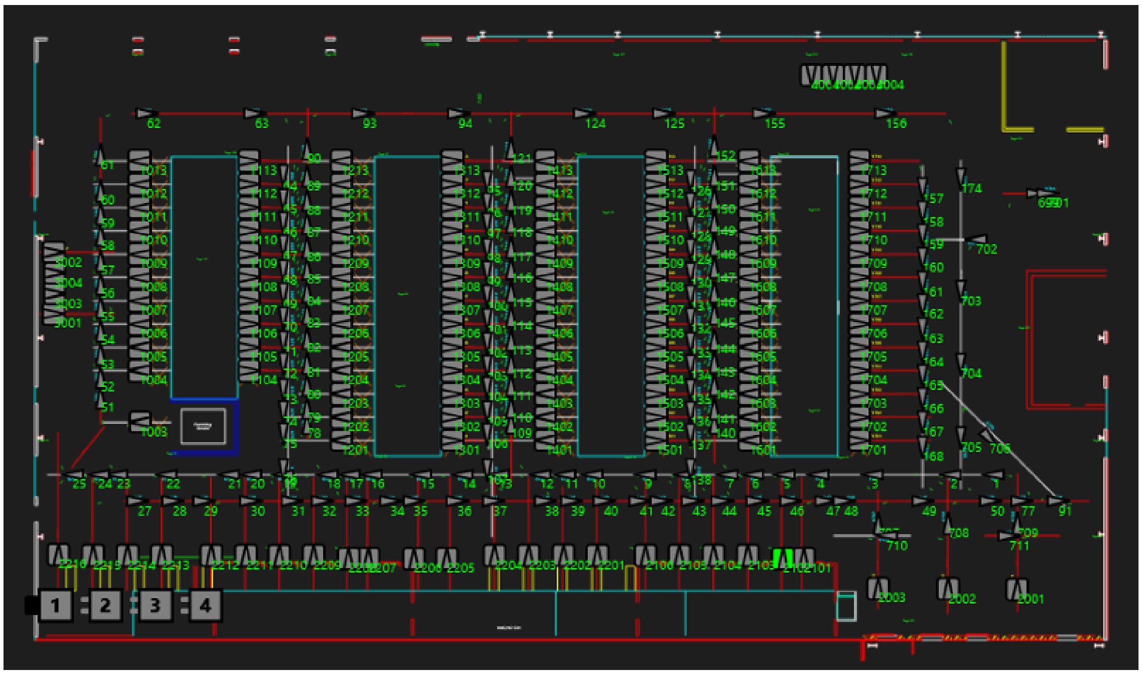


Figure 4.8: The layout for the Linde Gas project.

4.2.1 Layout analysis

In Layout 1, there is only one pair of lanes thus all the previous results are based on that pattern, a pair of lanes are located between stations and the space is enough for two AGVs to pass by. According to this experience, Layout 2 was divided into four different patterns for further analysis.

1. The left single lane: The characteristic of this pattern is that stations are located on both sides of a single lane. Furthermore, when an AGV exits from the stations, it does so by moving to a node upwards. For instance, an AGV exits following the link from station ST1008 to node X58.
2. The pair of lanes in the middle: Stations are located on both sides of a pair of nodes, and the AGV exits from the stations either by following a link upwards or downwards. For instance, AGVs can follow the link between station ST1107 and node X71 to move downwards or node X86 to move upwards. Another important feature is the space between the stations are not enough for two AGVs to pass by.
3. Two single lanes on the right: Stations are located on both sides of these two lanes downwards. The exiting direction is also downwards. For example, to exit station ST1710, an AGV follows the link between ST1710 and node X161.
4. A pair of lanes on the bottom: This pattern is similar to the pattern in Layout 1, where the space is enough for two AGVs to pass by. The connection of vertical lanes and horizontal lanes is considered as a station for the rule generation.

The layout analysis tool was also implemented on this layout. The results showed that only three stations, out of the 132, are safe, ST1013, ST1017, ST2216.

4.2.2 Simulation results

Based on the results from Layout 1, the check method is discarded. Only the zone-control method is discussed below.

With the results from the layout analysis tool and the experience from Layout 1, some “CHECK” rules and “DETOUR” rules were automatically generated and then the test was done. AGVs move quite smoothly on pattern 1, 3, and 4 mentioned in the previous section. However, these rules cannot prevent the case where two AGVs move towards each other in pattern 2, shown in Figure 4.9.

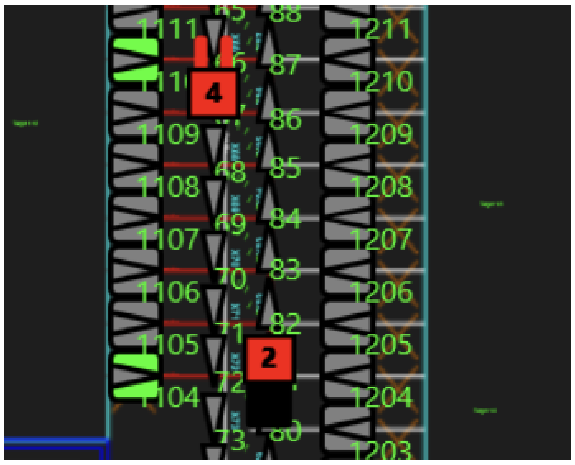


Figure 4.9: AGV2 and AGV4 got stuck in a deadlock situation.

Further investigation was carried out on applying the current rules to solve this type of deadlock. However, it ended up with the conclusion that it is not possible to solve it with the existing rules when the AGVs are already in the lanes. The details are discussed in Chapter 5.

Based on the experience of the engineers in the company, another zone-control method was proposed to suit in this pattern. The idea is expanding the definition of zones and then applying ”CHECK” rules to control the behaviour of AGVs at the entrance and exit of the lanes. For example, all the nodes (from X95 to X121) and stations (from ST1301 to ST1413) in the area in Figure 4.10, denoted as *A34*, are divided to two groups, upper zone and lower zone. The maximum number of AGVs in the zone is one. The reasons why divide into two such zones can be summarized as:

- The stations are overlapped with the lanes next to it. For example, the station ST1307 is overlapped with three nodes, X100, X101, X102. Thus, no AGVs from the upper zone can pass by ST1307 from the upper zone.
- The movement from the stations in A34 to the nodes on the lanes require a relatively big zone. To be specific, when an AGV moves from station ST1307 to node X102, nodes from X99 to X104 and from X114 to X116 are locked.
- The entrances and exits are only located on the upper and lower sides of this area respectively.

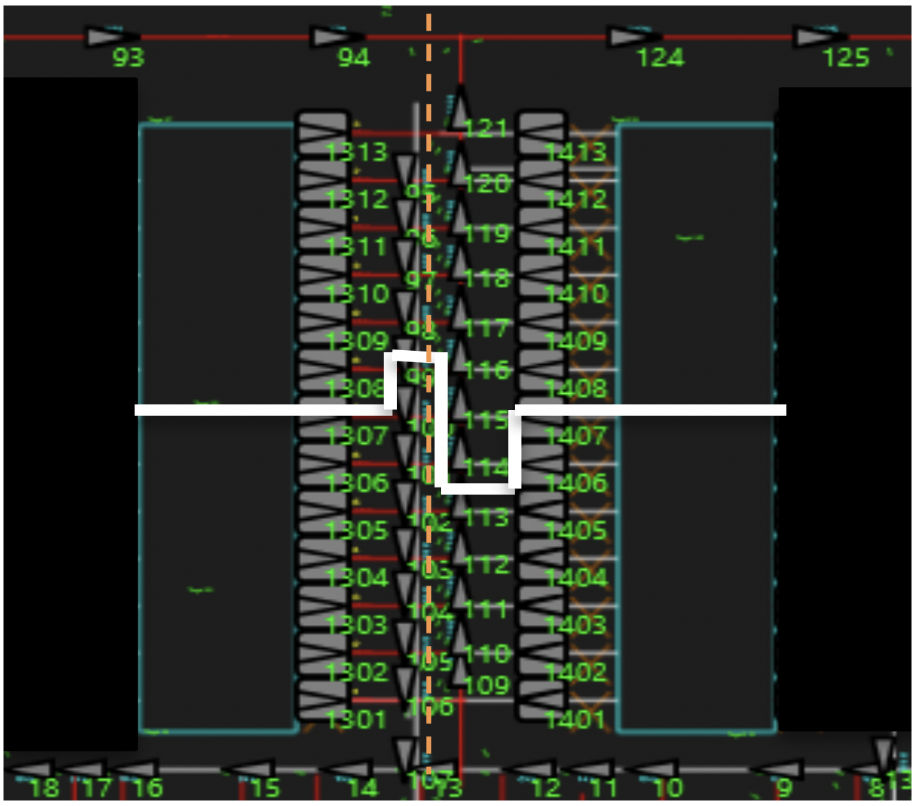


Figure 4.10: Dividing a pattern 2 case(A34) to two zones by the white line: Upper zone(A34Upper) is above the white segment and lower zone(A34Lower) is under the white segment. (The orange segment is explained in Section 4.2.3.)

With this division, the rules consists of “CHECK” and “DETOUR” are generated:

1. Using “CHECK” to control the entering to A34. For example, if the AGV is heading to the lower zone, marked as *A34Lower*, a command will be made to stop the AGV at node X94 if A34Lower is occupied. Correspondingly, if the AGV is heading to the upper zone *A34Upper*, it will be commanded to stop at node X12 to make a check on the A34Upper. The syntax looks like:
   - <CHECK NODES = "X95" IF_AREAS_FREE = "A34Lower" WHEN_ ITIN_HAS_NODES = "X98" />
   - <CHECK NODES = "X109" IF_AREAS_FREE = "A34Upper" WHEN_ ITIN_HAS_NODES = "X113" />"
2. Using “DETOUR” to control the exiting of the stations in A34. For instance, an AGV occupying station ST1308 will check whether A34Lower is occupied before moving to A34Lower. If A34Lower is occupied, a detour command to move to node X124 will be sent. The syntax looks like:
   - <REPLACE_ATTR OBJECT = "Link" NAMES = "ST1308_X101" DetourMethod = "DetourToNodeIfBlocked(124, MY_ITIN:102, AREA:A34Lower)" /> The syntax "MY_ITIN: 102" makes sure the AGV standing at node X1308 wants to move further to A34Lower rather than the stations ST1305 or ST1405.
   - <REPLACE_ATTR OBJECT = "Link" NAMES = "ST1301_X111" DetourMethod = "DetourToNodeIfBlocked(107, MY_ITIN:117,

AREA:A34Upper)" />

The Algorithm 6 to automatically generate the detour rules are presented as below:

```
1  Parse the OVERLAPINFO.XML to dic();
2  Divide the nodes on the lanes to upwards and downwards;
3  Divide the stations to upper and lower;
4  Store the exit node as exit_top and exit_down;
5  Store the node in the middle to identify the movement as up_mid and
     down_mid;
6  for sta in the sta_list do
7      if sta is in upper list then
8          store the next_loc in the downwards
9      end
10     Write the rules with sta, next_loc, exit_top, down_mid, and upperzone.
        if sta is in lower list then
11         store the next_loc in the upwards
12     end
13     Write the rules with sta, next_loc, exit_down, up_mid, and lowerzone.
14 end
```

Algorithm 6: Given the *OVERLAPLAP.XML*, the algorithm can generate the detour rules for pattern 2.

With these checks and detour rules, the deadlock situation in Figure 4.9 was successfully avoided. It should be noted that the test on Layout 2 was only carried out on different patterns due to the time limitation. No tests on the whole layout were done. There is still the possibility for AGVs to get into deadlock situations when they move from one pattern to another. However, this deadlock can be avoided by adding check rules to the SYSTEM_SETUP. This part of work is not included in this project.

4.2.3 Discussion

Layout 2 was broken into four different patterns. The zone-control method can avoid the deadlock situation on patterns 1, 3, and 4 with limited testing time. An improved version of zone-control method was proposed to fix deadlocks on pattern 2 and no deadlock happened there.

The test on the whole layout was not done on this layout due to time limitation. Thus, there are possibilities for the AGVs to get in trouble when moving from one pattern to another. This can be solved by manually adding check rules on the intersections of the patterns. This type of rules varies according to some factors, such as the position of the pattern, the type of the pattern and the design of the path. It is challenging to automate this process for all the layouts and the work load for manually adding is acceptable.

The strategy of dividing pattern 2 is based on the experience of the engineers. Another strategy, which is dividing it into four zones by further breaking these two zones, was also considered and discussed. This ends up with a conclusion that it may cause some unsolvable deadlocks with the existing tools. For example, in Figure 4.10, further dividing the upper zone or lower zone into two zones. The maximum number of AGVs allowed in each of these four zones is 1. However, the deadlock will occur when two AGVs reserve the lanes to exit from the stations simultaneously and it is not possible handle this case with existing syntax. A further discussion is presented in the Section 5.1.

4.3 Summary

In this chapter, the procedure proposed in Chapter 3 was implemented on two layouts. It is concluded that the check method is not suitable for the layouts tested in this project, due to the complexity of the layouts. The zone-control method successfully made Layout 1 deadlock-free but it is hard to directly apply it on Layout 2 because of the complexity of the layout. A modified version was then proposed and tested in Layout 2. The results showed that, in certain areas of the layout, deadlock never occurred in the limited simulation.

5
Future work

Previous chapters presented a structured procedure to resolve deadlocks in the AGVS. Due to time limitation, only two layouts were tested and results were discussed. As mentioned in Section 1.3, there are many limitations on this project since the focus is on solving the problem for the company. With the results above, it is convinced that the problem was solved on some layouts even though it is hard to directly use for other layouts.
In this chapter, some suggestions are made on the extension of this thesis work from the authors' perspective. It can be mainly divided into two parts. Firstly, more syntax or existing tools are suggested to get a better performance. Secondly, some ideas that came up during the process of the project are also discussed, even though there is currently not a suitable way to integrate them with the system. The authors believe these are inspirational and possible to be used in the company.

5.1 Syntax suggestions

This syntax suggestion is based on pattern 2 in Layout 2. In Section 4.2.3, it is mentioned that more advanced syntax could be developed for better performance. With the existing syntax, it is possible to make use of the current node occupied and the itinerary node of one AGV, for example:
<"CHECK_NODES = X64 , WHEN_ITIN_HAS_NODE = ST1201">
The syntax above means: check whether node X64 is free before an AGV moves to X64 if ST1201 is in the itinerary of the AGV. Moreover, other AGVs' current node and itinerary can be used by the syntax:
<"CheckIfNotBlocked(OTHER_ITIN: X5 NODES: X3">
The function means that return value 0 if there is an AGV standing on node X3 and if there is one AGV has X5 in its itinerary. This function is usually used together with CHECK rules. For example:
<"CHECK_NODES = X64
IF_METHOD_TRUE = CheckIfNotBlocked(OTHER_ITIN: X5 NODES: X3) ">
This means that if the function returns 0, then the check fails and the AGV will not move to node X64. An idea inspired from Layout 2 is that the combination of them can make sure the route of two AGVs if there are only two AGVs in the system. For instance, the extended syntax will look like this:
<CHECK NODES = X64
IF_METHOD_TRUE= CheckIfNotBlocked(OTHER_ITIN: X5 NODES:X3)
WHEN_ITIN_HAS_NODE= X69 >

With this syntax, an AGV will make a check before moving to node X64 when X69 is in the itinerary of this AGV. The check refers to whether there is another AGV occupying node X3 and with node X5 in its itinerary. Then the route of these two AGVs can be partly determined. For instance, to get a better performance mentioned in Figure 4.9, the syntax below could be implemented:
<CHECK NODES = X95
IF_METHOD_TRUE= CheckIfNotBlocked(OTHER_ITIN:X124 NODES:A34Lower)
WHEN_ITIN_HAS_NODE= X100 >
This syntax means that if an AGV stands on node X94, a check will be made if this AGV is to be sent to node X100. The content of check is making sure no AGVs occupying the nodes in A34Lower are moving to node X124 which is the exit node of the A34Upper. By this way, the AGV will wait on node X123 only when other AGVs move from A34Lower to upper exit node X124. Otherwise, if other AGVs exit from the lower exit, the AGV will move on. Previously, no matter where other AGVs head to, the AGV always wait. This can improve the performance on this certain pattern, but it is believed it can be also helpful for other patterns as further investigation progresses.

5.2 Future layout design procedure

One of aims of this project was to reduce manual work. The method proposed in this thesis could automatically generate deadlock avoidance rules for different patterns of layout. In the future, the company could use the procedure shown below to design new layouts. The first step is to conclude different patterns from old layouts. Then, according to the features of the patterns, methods that could automatically generate rules are prepared. When the company needs to design new layouts, the existing patterns could be preferentially selected and the deadlock avoidance rules can be easily generated.

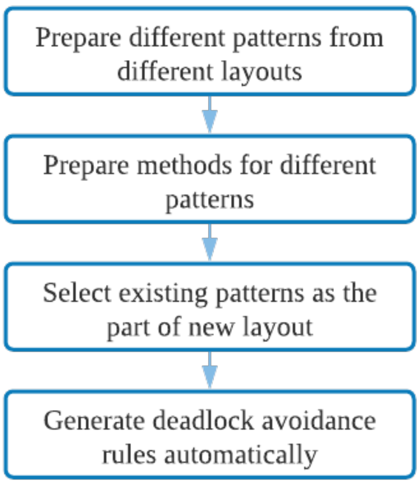

Prepare different patterns from different layouts
Prepare methods for different patterns
Select existing patterns as the part of new layout
Generate deadlock avoidance rules automatically

Figure 5.1: Flowchart of future layout design procedure.

Bibliography

[1] M. P. Fanti and B. Turchiano, "Deadlock avoidance in automated guided vehicle systems," *2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, vol. 2, pp. 1017–1022, 2001.

[2] U. Günter, *The History of Automated Guided Vehicle Systems.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, ch. 1, p. 2.

[3] M. Ghobakhloo, "Industry 4.0, digitization, and opportunities for sustainability," *Journal of Cleaner Production*, vol. 252, no. 119869, pp. 1017–1022, 2020.

[4] F. D. M. De Ryck*, M. Versteyhe, "Automated guided vehicle systems, state-of-the-art control algorithms and techniques," *Journal of Manufacturing Systems*, vol. 54, pp. 152–173, 01 2020.

[5] Z. Li, N. Wu, and M. Zhou, "Deadlock control of automated manufacturing systems based on petri nets—a literature review," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, p. 437–4626, 2012.

[6] S. Riazi and B. Lennartson, "Using cp/smt solvers for scheduling and routing of agvs," *IEEE Transactions on Automation Science and Engineering*, pp. 1–12, 2020.

[7] S. P. E. Mehlhorn, K., *Algorithms and Data Structures: The Basic Toolbox.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, ch. Shortest Paths, p. 191–215.

[8] E. G. Coffman, M. Elphick, and A. Shoshani, "System deadlocks," *ACM Comput. Surv.*, vol. 3, no. 2, p. 67–78, Jun. 1971. [Online]. Available: https://doi-org.proxy.lib.chalmers.se/10.1145/356586.356588

[9] M. P. Fanti, B. Maione, S. Mascolo, and B. Turchiano, "Event-based feedback control for deadlock avoidance in flexible production systems," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 3, pp. 347–363, June 1997.

[10] J. Yoo, E. Sim, C. Cao, and J. Park, "An algorithm for deadlock avoidance in an agv system," *The International Journal of Advanced Manufacturing Technology*, vol. 26, pp. 659–668, 2005.

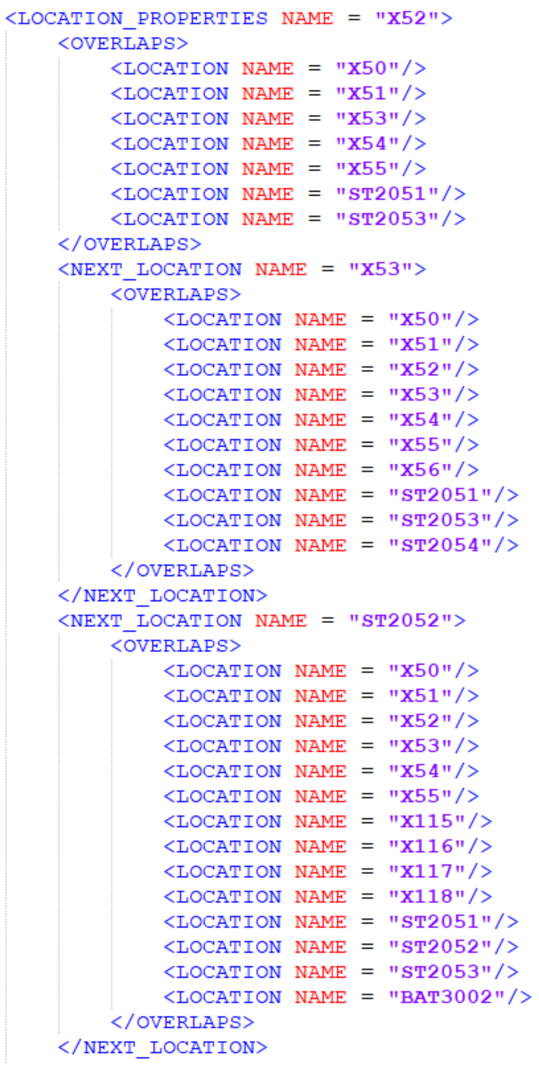
[11] N. Viswanadham, Y. Narahari, and T. L. Johnson, "Deadlock prevention and deadlock avoidance in flexible manufacturing systems using petri net models," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 6, pp. 713–723, 1990.

[12] C. F. C. Secchi, R. Olmi and M. Casarini, *Gearing Up and Accelerating Cross-fertilization between Academic and Industrial Robotics Research in Europe.* Springer Cham, 2014, vol. 94, ch. TRAFCON – Traffic Control of AGVs in Automatic Warehouses, p. 85–105.

[13] N. Wu and M. Zhou, “Modeling and deadlock control of automated guided vehicle systems,” *IEEE/ASME Transactions on Mechatronics*, vol. 9, no. 1, pp. 50–57, 2004.

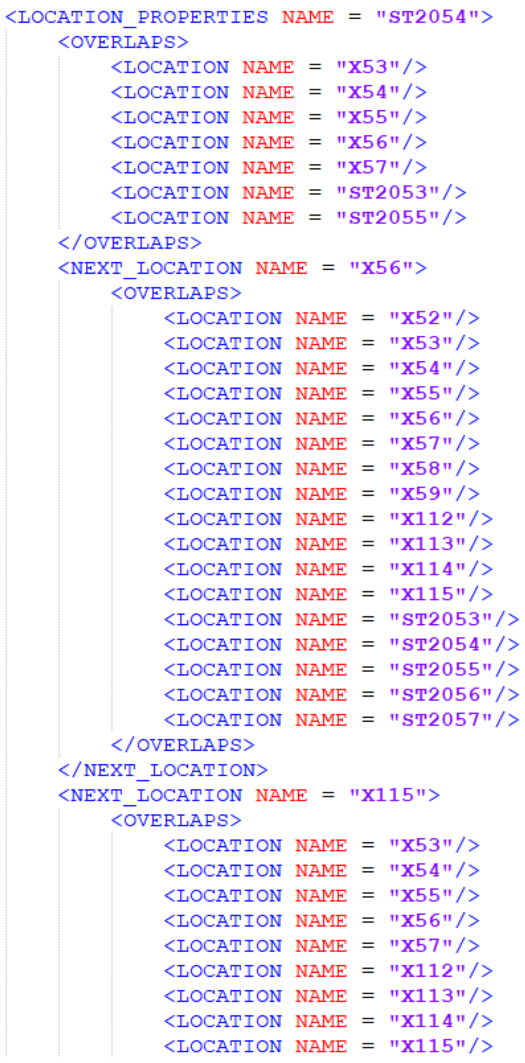
A

Appendix



```
<LOCATION_PROPERTIES NAME = "X52">
    <OVERLAPS>
        <LOCATION NAME = "X50"/>
        <LOCATION NAME = "X51"/>
        <LOCATION NAME = "X53"/>
        <LOCATION NAME = "X54"/>
        <LOCATION NAME = "X55"/>
        <LOCATION NAME = "ST2051"/>
        <LOCATION NAME = "ST2053"/>
    </OVERLAPS>
    <NEXT_LOCATION NAME = "X53">
        <OVERLAPS>
            <LOCATION NAME = "X50"/>
            <LOCATION NAME = "X51"/>
            <LOCATION NAME = "X52"/>
            <LOCATION NAME = "X53"/>
            <LOCATION NAME = "X54"/>
            <LOCATION NAME = "X55"/>
            <LOCATION NAME = "X56"/>
            <LOCATION NAME = "ST2051"/>
            <LOCATION NAME = "ST2053"/>
            <LOCATION NAME = "ST2054"/>
        </OVERLAPS>
    </NEXT_LOCATION>
    <NEXT_LOCATION NAME = "ST2052">
        <OVERLAPS>
            <LOCATION NAME = "X50"/>
            <LOCATION NAME = "X51"/>
            <LOCATION NAME = "X52"/>
            <LOCATION NAME = "X53"/>
            <LOCATION NAME = "X54"/>
            <LOCATION NAME = "X55"/>
            <LOCATION NAME = "X115"/>
            <LOCATION NAME = "X116"/>
            <LOCATION NAME = "X117"/>
            <LOCATION NAME = "X118"/>
            <LOCATION NAME = "ST2051"/>
            <LOCATION NAME = "ST2052"/>
            <LOCATION NAME = "ST2053"/>
            <LOCATION NAME = "BAT3002"/>
        </OVERLAPS>
    </NEXT_LOCATION>
```

a. node X52



```
<LOCATION_PROPERTIES NAME = "ST2054">
    <OVERLAPS>
        <LOCATION NAME = "X53"/>
        <LOCATION NAME = "X54"/>
        <LOCATION NAME = "X55"/>
        <LOCATION NAME = "X56"/>
        <LOCATION NAME = "X57"/>
        <LOCATION NAME = "ST2053"/>
        <LOCATION NAME = "ST2055"/>
    </OVERLAPS>
    <NEXT_LOCATION NAME = "X56">
        <OVERLAPS>
            <LOCATION NAME = "X52"/>
            <LOCATION NAME = "X53"/>
            <LOCATION NAME = "X54"/>
            <LOCATION NAME = "X55"/>
            <LOCATION NAME = "X56"/>
            <LOCATION NAME = "X57"/>
            <LOCATION NAME = "X58"/>
            <LOCATION NAME = "X59"/>
            <LOCATION NAME = "X112"/>
            <LOCATION NAME = "X113"/>
            <LOCATION NAME = "X114"/>
            <LOCATION NAME = "X115"/>
            <LOCATION NAME = "ST2053"/>
            <LOCATION NAME = "ST2054"/>
            <LOCATION NAME = "ST2055"/>
            <LOCATION NAME = "ST2056"/>
            <LOCATION NAME = "ST2057"/>
        </OVERLAPS>
    </NEXT_LOCATION>
    <NEXT_LOCATION NAME = "X115">
        <OVERLAPS>
            <LOCATION NAME = "X53"/>
            <LOCATION NAME = "X54"/>
            <LOCATION NAME = "X55"/>
            <LOCATION NAME = "X56"/>
            <LOCATION NAME = "X57"/>
            <LOCATION NAME = "X112"/>
            <LOCATION NAME = "X113"/>
            <LOCATION NAME = "X114"/>
            <LOCATION NAME = "X115"/>
```

b. station ST2054

Figure A.1: Part of overlap information of node X52 and station ST2054.

```
<CHECK NODES = "X151" IF_NODES_FREE = "X148" WHEN_ITIN_HAS_NODES = "X152"/>
<CHECK NODES = "X151" IF_NODES_FREE = "X149" WHEN_ITIN_HAS_NODES = "X152"/>
<CHECK NODES = "X151" IF_NODES_FREE = "X150" WHEN_ITIN_HAS_NODES = "X152"/>
<CHECK NODES = "X151" IF_NODES_FREE = "X151" WHEN_ITIN_HAS_NODES = "X152"/>
<CHECK NODES = "X151" IF_NODES_FREE = "X152" WHEN_ITIN_HAS_NODES = "X152"/>
<CHECK NODES = "X151" IF_NODES_FREE = "X153" WHEN_ITIN_HAS_NODES = "X152"/>
<CHECK NODES = "X151" IF_NODES_FREE = "X154" WHEN_ITIN_HAS_NODES = "X152"/>
<CHECK NODES = "X151" IF_NODES_FREE = "X155" WHEN_ITIN_HAS_NODES = "X152"/>
```

Figure A.2: Check method's rules of Node X151 with direction to X152.