Eindhoven University of Technology

Department of Mechanical Engineering
Systems and Control

Design, Implementation and Analysis of a Hierarchical Control Mechanism for a Grid-Based AGV System

Project supervisors:
Joost van Eekelen - TU/e
Karlijn Fransen - Vanderlande

Student:
Remco Vader - 1251597

Eindhoven, November 16, 2022

Declaration concerning the TU/e Code of Scientific Conduct for the Master's thesis

I have read the TU/e Code of Scientific Conduct[i].

I hereby declare that my Master's thesis has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct

Date

13-12-2022
…………………………………………………..…………..

Name

R.M. Vader…………………………………………..…………..

ID-number

1251597
…………………………………………………..…………..

Signature

[illegible]
…………………………………………………..…………..

Submit the signed declaration to the student administration of your department.

[i] See: https://www.tue.nl/en/our-university/about-the-university/organization/integrity/scientific-integrity/
The Netherlands Code of Conduct for Scientific Integrity, endorsed by 6 umbrella organizations, including the VSNU, can be found here also. More information about scientific integrity is published on the websites of TU/e and VSNU

Abstract

Although Vanderlande already has an existing control strategy in place for their real-life AGV system called FLEET, Vanderlande is also exploring alternative routing controllers using their grid-based control model. In this thesis a new hierarchical control strategy is developed, which aims at exploiting the scalability of decentralized approaches while limiting the model complexity and typically non-optimal character of decentralized approaches. The performance, potential and problems of the new hierarchical control strategy are compared with respect to the existing controller in Vanderlande's control model, called the supervisory controller.

The hierarchical control strategy that is designed in this thesis can be distinguished into a centralized and a decentralized layer. Global paths are determined by a centralized global controller and serve as an approximation of the shortest path that each vehicle must take to traverse from one location to another location. The layout is partitioned into multiple sectors in which traffic is controlled using a decentralized local controller, which only has access to local information. Because the desired properties of the controller in a particular sector depend on the characteristics of that sector, the properties of a controller are allowed to vary per sector.

Using the hierarchical control strategy, the mean throughput of the system typically slightly decreases and the average item lead time typically slightly increases. The cumulative controller cycle time of each individual local controller, i.e. the total time the controller spends performing its cycle during a simulation, is consistently significantly lower than the cumulative controller cycle time of the supervisory controller. However, in most cases, the total controller cycle time of all hierarchical controllers, i.e. all local controllers plus the global controller, exceeds the cumulative controller cycle time of the supervisory controller. Because the grid-based control model is simulated using a discrete-event simulation, during the simulation, all controllers perform their control cycle in series. In practice, however, the controllers can perform their control cycle in parallel with other controllers. If the control cycles are mainly performed in parallel, then the distribution of the computational burden seems to be relatively effective, though typically at the cost of a slight decrease in throughput and a slight increase in lead time. On the other hand, if the control cycles are mainly performed in series, the benefit of using a hierarchical control strategy is questionable, because in most cases the total cumulative controller cycle time of all hierarchical controllers is higher than that of the supervisory controller. Even if the control cycles are mainly performed in series, the hierarchical control mechanism that is developed in this thesis can still be used as a means to successfully introduce different types of local controllers with varying properties, depending on the desired properties per region, i.e. per sector, of the layout.

A significant problem that can arise using the designed hierarchical control strategy is the formation of a multi-sector deadlock. Such a deadlock is a situation where a group of vehicles is waiting for each other forever because their next tile is already occupied or reserved by another vehicle, and the involved vehicles are located in different sectors. Because each local controller only has access to local information, such a deadlock cannot easily be identified. How a layout is partitioned into sectors can make a major difference in the probability that multi-sector deadlock occurs.

Glossary

Term	Definition
Admissible	The heuristic cost of the A* algorithm is said to admissible if it never overestimates the cost of reaching the destination node from any particular node.
Centralized control	Control approach in which a central controller is present which manages all vehicles.
Circular wait	A situation where vehicles form a circular chain, such that each vehicle waits for a tile that is reserved by the next vehicle in the chain.
Communication object	Object in the simulation that can communicate with other objects via the mediator.
Complete algorithm	A complete algorithm is guaranteed to find a solution, or correctly reports that no solution exists, in a finite amount of time.
Cumulative controller cycle time	Total time that a controller spends performing its cycle during a simulation.
Deadlock	Situation where a group of vehicles are all waiting for each other forever because their next tile is already occupied or reserved by another vehicle.
Decentralized control	Control approach in which no central controller is present which manages all vehicles.
Dispatching	Assignment of a job to a vehicle that needs to perform said job.
Drop-off station	Location in the layout where a vehicle can drop off a load.
Entrance tile	Tile through which a sector can be entered from another sector.
Exit tile	Tile through which a sector can be left towards another sector.
Global controller	Centralized controller that plans global paths for all vehicles and forwards vehicle updates to the correct local controller.
Global layout	An approximation of the full layout in which most details are disregarded, which is used to determine global paths.
Global path	Approximation of the shortest path that a vehicle must take to traverse from one point of interest to another point of interest.
Global target	The final target of a global path, which is the final target that a vehicle must travel to.
Grid-based model	Model in which the controller keeps a virtual representation of the environment in the form of a grid of tiles.
Graph search algorithm	Algorithm used to determine the shortest path in a network of nodes and edges.
Hierarchical control strategy	Control strategy in which the control structure is arranged as a hierarchical tree.
Job provider	Entity responsible for generating and assigning jobs to vehicles.
Livelock	Situation where the system state constantly changes, typically in a circular manner, but fails to achieve some desired state in finite time.
Local controller	Decentralized controller that manages the local path planning and local traffic control in one particular sector, for the vehicles that are currently located in said sector.
Local layout	Digital representation of the environment in one particular sector.
Local path	Exact path a vehicle must take within the a sector in order to reach the local target of this sector.
Local target	Final target of a local path.
Mediator	Communication network between communication objects in the grid-based model.

Term	Definition
Mono-cycle deadlock	Deadlock situation in which a circular wait is present.
Multi-cycle deadlock	Deadlock situation in which multiple impending deadlock cycles are present.
Multi-sector deadlock	Deadlock situation in which vehicles from multiple sectors are involved.
Participant	An object in the simulation that has at least one type of event that it can execute.
Path planning	Generating a path from location A to B while minimizing some objective function.
Pick-up station	Location in the layout where a vehicle can pick up a load.
Routing	Combination of path planning and traffic control.
Scalability	Applicability to large and dense systems.
Sector	Portion of the layout that has its own local controller.
Supervisory controller	Controller used in Vanderlande's grid-based control model.
Time to standstill	Estimated time until the vehicle must stand still because the vehicle would otherwise surpass its waypoints.
Vehicle agent	Digital representation of a vehicle within the controller.
Vehicle density	Ratio of the number of vehicles in a system to the number of drivable tiles in the layout.
Well-formed infrastructure	An infrastructure where any robot standing on its endpoint can never entirely prevent other robots from moving between two other endpoints.

Contents

Abstract **i**

Glossary **iii**

1 Introduction **1**

1.1 Introduction to AGV systems . . . 1
1.2 Research objective . . . 1
1.3 Outline . . . 2

2 Literature background **3**

2.1 Path planning . . . 3
2.1.1 Dijkstra's and A* algorithm . . . 3
2.1.2 D* algorithm and variants of D* . . . 4
2.1.3 Optimization-based algorithms . . . 5
2.1.4 Metaheuristic algorithms . . . 5
2.1.5 Summary on path planning . . . 5
2.2 Traffic control . . . 6
2.2.1 Functions of traffic control . . . 6
2.2.2 Centralized and decentralized control . . . 10
2.2.3 Summary on traffic control . . . 13
2.3 Proposed control approach . . . 14

3 Grid-based control model **15**

3.1 Grid-based model setup . . . 15
3.1.1 Discrete-event simulation . . . 15
3.1.2 Participants in the simulation . . . 16
3.1.3 Other objects in the simulation . . . 18
3.1.4 Message types in grid-based control . . . 19
3.2 The supervisory control cycle . . . 20
3.3 Communication between objects . . . 21
3.3.1 Vehicle updates from vehicles to controller . . . 21
3.3.2 Vehicle commands from controller to vehicles . . . 22

4 Hierarchical control design **23**

4.1 Hierarchical control definitions . . . 23
4.2 Hierarchical component design . . . 24
4.2.1 Local controller . . . 24
4.2.2 Local layout . . . 25
4.2.3 Global layout . . . 27
4.2.4 Global controller . . . 33
4.3 Hierarchical control procedures . . . 34
4.3.1 Message types in hierarchical control . . . 34
4.3.2 Planning paths using global targets and local targets . . . 34
4.3.3 Vehicle updates from vehicles to local controllers . . . 36
4.3.4 Vehicle commands from local controllers to vehicles . . . 36
4.3.5 Assigning global paths to vehicle agents . . . 37
4.3.6 Vehicle transitions between local controllers . . . 38
4.4 The global control cycle . . . 39
4.5 The local control cycle . . . 40
4.6 Multi-sector deadlock . . . 42

5 Comparison of results **44**
5.1 Simple layout . . . 44
5.2 Intermediate layout . . . 47
5.3 Advanced layout . . . 49

6 Conclusion and recommendation **56**
6.1 Conclusion . . . 56
6.2 Recommendation . . . 57

References **63**

A Appendix A **66**
A.1 Vehicle updates from vehicles to controller . . . 66
A.2 Vehicle commands from controller to vehicles . . . 67

B Appendix B **68**
B.1 Vehicle updates from vehicles to local controllers . . . 68
B.2 Vehicle commands from local controllers to vehicles . . . 70
B.3 Assigning global paths to vehicle agents . . . 72
B.4 Vehicle transitions between local controllers . . . 74

1 Introduction

1.1 Introduction to AGV systems

In recent years, intralogistic solutions are shifting from fixed infrastructure like conveyor belts and sorters to mobile infrastructure like automated guided vehicles (AGVs) and autonomous mobile robots (AMRs) [Fragapane et al., 2021]. However, with the rise of such mobile infrastructure also come a lot of new research questions that require solutions. In [Fazlollahtabar and Saidi-Mehrabad, 2013] a literature study was performed in which existing methods for optimizing AGV systems were examined with regard to scheduling and routing of manufacturing, distribution, transshipment and transportation systems. They concluded that new analytical and simulation models need to be developed that can cope with large AGV systems and specifically more attention should be paid to integrating scheduling and routing simultaneously. The main motivation behind this is to overcome large computation times, NP-completeness, congestion, deadlocks and delays in the system.

The control strategy used to manage AGVs can roughly be distinguished into three components: dispatching (also called job assignment), path planning and traffic control. Moreover, path planning and traffic control can be combined into *routing*.

In *dispatching*, the controller determines which AGV should perform which job, and from which pick-up station to which drop-off station the job needs to be performed [Fransen et al., 2020]. In [Jacobs, 2020], such dispatching strategies have already been investigated and a new dispatching strategy has been designed and implemented that is applicable to a variety of AGV systems within Vanderlande. *Path planning* consists of generating a path from location A to B while minimizing some objective function [De Ryck et al., 2020]. Assuming that a network of nodes and edges, i.e. a graph, already exists, a graph search algorithm can be used to find such a path. When a path has been planned by the path planning algorithm, *traffic control* is concerned with coordinating vehicles so that they can successfully execute their planned path. One of the ways to accomplish this is to make use of a *grid-based model* where the controller keeps a virtual representation of the environment. This virtual representation consists of a grid of tiles, where each tile can only hold one vehicle at a time. In other literature, this is also referred to as zone-based control. The three major responsibilities of a traffic controller are collision avoidance, deadlock handling and livelock handling [Fransen et al., 2020].

1.2 Research objective

Although Vanderlande already has an existing control strategy in place for their real-life AGV system called FLEET [Vanderlande, 2022], Vanderlande is also exploring alternative routing controllers using their *grid-based control model*. Because Vanderlande aims to be able to cope with larger AGV systems than is possible with the current control strategy, in [Vader, 2022] a literature research was performed to obtain an overview of existing routing techniques. Based on this literature research, the choice was made to develop a new hierarchical control strategy in this thesis, where a trade-off between an increase in scalability and a decrease in throughput is one of the foreseen design decisions. The newly developed hierarchical control strategy is to be compared with the controller currently existing in the grid-based model. This controller is largely based on the findings in [Fransen et al., 2020] and [Fransen et al., 2022], and will be referred to as the *supervisory controller* in the remainder of this thesis. As dispatching was already treated in [Jacobs, 2020], in this thesis the focus lies on path planning and traffic control, i.e. the vehicle routing problem.

The main research question to be answered in this thesis is:

What does the design of a hierarchical control strategy look like and how can its performance be compared with the supervisory controller?

The new hierarchical control strategy that is developed in this thesis has a couple of requirements that *must* be satisfied to be able to make a fair comparison with the supervisory controller. The requirements for the new control strategy are as follows:

- The new controller must be suitable for a grid-based system. Vehicles cannot move freely across the layout but have to stick to a predefined grid of tiles.
- Vehicles have a finite acceleration. An infinite acceleration (i.e. vehicles can go from 0 to maximum speed and from maximum speed to 0 instantly) is too oversimplified and not a good representation of reality.
- The controller must be able to process jobs in an online manner. All jobs are unknown beforehand.
- The controller must perform all jobs in a correct fashion, i.e. all job instructions are performed at their corresponding task locations.

1.3 Outline

The outline of the remainder of this report looks as follows:

First, in Chapter 2 a literature study is performed to obtain an overview of existing path planning and traffic control techniques in literature. In this chapter, the notions of collision avoidance, deadlock handling and livelock handling are clarified and a distinction is made between centralized, decentralized and hierarchical control approaches.

Then, in Chapter 3 Vanderlande's existing grid-based control model is explained in full detail. All participants of the discrete-event simulation are introduced, as well as other relevant objects. Furthermore, the procedures in the control cycle of the supervisory controller and the communication processes between vehicles and the controller are introduced.

Next, in Chapter 4 the new hierarchical control strategy is developed. The design of all new components and message types in the model is explained and motivated, as well as all new procedures that are required so that vehicles can successfully perform their jobs. In addition, all procedures in both the global and local control cycles are introduced. The chapter is wrapped up with the introduction of multi-sector deadlock.

Thereafter, in Chapter 5, the performance of the new hierarchical control strategy is compared with the supervisory controller following experimental simulations. Simulations are performed on three different layouts for a variety of vehicle densities.

Finally, in Chapter 6 conclusions are drawn regarding the design and performance of the hierarchical control strategy in comparison with the supervisory controller. Moreover, several recommendations are given based on which further research can be performed.

2 Literature background

As was introduced in Section 1.1, the routing problem can be distinguished into two components, called path planning and traffic control. In this chapter, both of these concepts are introduced and a literature study is performed in order to obtain an overview of existing path planning and traffic control techniques in literature.

2.1 Path planning

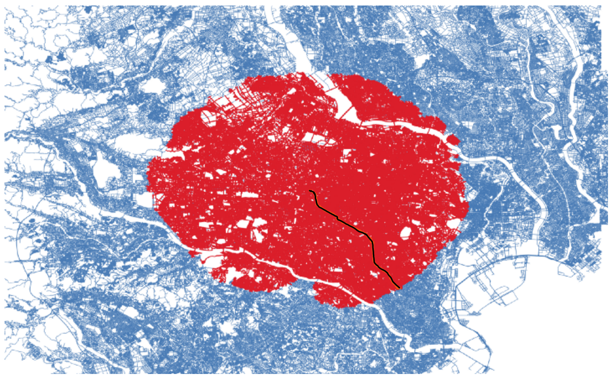
In a network of nodes and edges, there usually exist several ways in which a vehicle can travel from one location to another. Within path planning, the goal is to generate a path from location A to B while minimizing some objective function, which is often the travel distance or travel time [De Ryck et al., 2020]. Assuming that a network of nodes and edges already exists, a graph search algorithm can be used to find a solution to this objective. A search algorithm is said to be *complete* if it is guaranteed to find a solution for an arbitrary input, or correctly reports that no solution exists, in a finite amount of time. Incomplete algorithms, however, do not always find a solution, even if one exists [De Ryck et al., 2020].

In this section, several types of algorithms are introduced that can be used as such a graph search algorithm.

2.1.1 Dijkstra's and A* algorithm

One way to compute the shortest path between nodes is by using *Dijkstra's algorithm* [Dijkstra, 1959]. Starting from the source node, the distance to all neighboring nodes (i.e. the nodes that can be reached from this node via a directed edge) is explored. To each of these nodes, this distance is assigned as the cost of travelling to this node. Then, for the node with the lowest cost value, all its neighboring nodes are also explored. Again, the distance between these nodes is assigned as the cost of travelling to these nodes. For the node with the current lowest cost, again its neighboring nodes are evaluated. This process keeps repeating until the destination node has been reached.

The main drawback of Dijkstra's algorithm is that it does not include any sense of direction. Therefore, it could be that the algorithm first evaluates a path in the opposite direction of the destination, before evaluating paths in the correct direction. An example of this has been illustrated in Figure 2.1a, where the shortest path between two locations is determined in a real-life scenario. In this figure, unexplored nodes and edges are indicated in blue, whereas the portion of the graph that is explored by Dijkstra's algorithm is indicated in red, starting from the center of the image. The shortest path from the start to end location has been indicated in black. As can be seen, a lot of nodes and edges are evaluated in the opposite direction of the destination.


(a) Explored nodes using Dijkstra's algorithm.

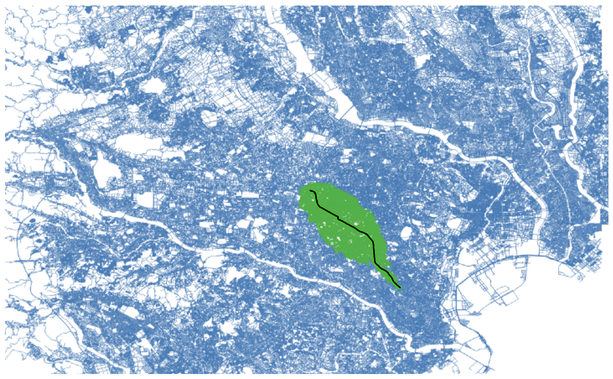

(b) Explored nodes using A* algorithm.

Figure 2.1: Comparison between the explored nodes using Dijkstra's algorithm and the A* algorithm in a real-life scenario [Malacad, 2022].

The *A* algorithm* is an extension of Dijkstra's algorithm which is widely used in literature. In contrast to Dijkstra's algorithm, the A* algorithm does include some sense of direction. In addition to the cost of travelling to a certain node, the A* algorithm namely introduces a heuristic cost that guides the algorithm in the direction of the destination.

The cost function of visiting a particular node has two components and is defined as

$$f(x) = g(x) + h(x) \tag{2.1}$$

where $g(x)$ is the actual cost of visiting this node, starting from the source node, and $h(x)$ is a heuristic cost that estimates the cost from this node to the destination. Usually, this heuristic cost is based on the Manhattan or Euclidean distance [Fragapane et al., 2021].

The further a node is located from the destination, the higher the $h(x)$-cost. Drifting in the wrong direction is therefore punished. The A* algorithm is proven to be complete and optimal under the condition that the heuristic cost $h(x)$ is *admissible*, i.e. it never overestimates the cost of reaching the destination node from any particular node [Malacad, 2022]. The effect of using this heuristic can be recognized in the example of Figure 2.1b. In this figure, unexplored nodes and edges are indicated in blue, whereas the portion of the graph that is explored by the A* algorithm is indicated in green, starting from the center of the image. The shortest path has been indicated in black. When comparing this figure with Figure 2.1a, it can be seen that the number of explored nodes and edges is significantly reduced when using the A* algorithm in comparison to Dijkstra's algorithm.

In literature there exist several adjusted versions of the A* algorithm that aim to avoid congested paths. In [Fransen et al., 2020], for example, an adjusted cost function is used which includes weights for using particular nodes. These node weights are updated over time, based on the time vehicles spend standing still on their respective nodes. Over time, these node weights are reduced back to zero using exponential smoothing. In addition, this adjusted cost function also incorporates turning costs. [Yuan et al., 2016] proposed an adjusted cost function in which a penalty is added for the paths that a vehicle shares with other vehicles. Vehicles that are closer to the target node are penalized more than vehicles that are further away.

2.1.2 D* algorithm and variants of D*

Although the A* algorithm is an efficient and complete algorithm, it implicitly assumes that it has full information on the layout and that the layout does not change. In reality, however, this might not be the case. One possible solution could be to recalculate the planned paths every once in a while, based on the most recent information. However, this is rather inefficient if the layout is large and the destination node is far away [Stentz, 1995]. Another possible solution is provided by the *D* algorithm.* In essence, the D* algorithm is a *dynamic* modification of the A* algorithm, which is also proven to be complete and optimal [Stentz, 1994].

Contrary to the A* algorithm, the D* algorithm begins at the destination node, and searches backwards to the source node. Once the D* algorithm discovers an obstacle, the current state is marked as invalid and all neighboring valid states are placed on a list of states that are to be evaluated. Then, new optimal paths are grown from the previous path through state expansion. Generally, this approach is advantageous over recomputing the entire cost map, since the algorithm only has to repair the optimal paths out of the vehicle's current location [Stentz, 1995]. In other words, the A* algorithm has to replan the path from the source node, whereas the D* algorithm keeps information about prior searches, which it can use in the next search [Kim et al., 2018]. However, this means that if no replanning has to be done, the D* algorithm unnecessarily stores information. As a result of this, the computation time of the first (few) iteration(s) of the D* algorithm is usually higher than the computation time of the A* algorithm. For later iterations, however, the computation time of the D* algorithm is usually lower, because it can use the information from prior searches. In [Kim et al., 2018] it was concluded that for large and complicated layouts, the D* algorithm normally finds the shortest path faster than the A* algorithm. However, in small and simple layouts, the A* algorithm outperforms the D* algorithm. So, there is a certain tipping point where using the D* algorithm becomes more efficient than using the A* algorithm, which depends on the system characteristics.

Over the years, several modified versions of the D* algorithm have been developed. Two of these are *Focussed D** [Stentz, 1995] and *D* Lite* [Koenig and Likhachev, 2002], who both use heuristics with the goal of decreasing the computation time of the D* algorithm.

2.1.3 Optimization-based algorithms

Although Dijkstra's algorithm, the A* algorithm and the D* algorithm are the most commonly used algorithms in path planning for AGVs, there are more general optimization algorithms that can be used.

In general, an *optimization-based algorithm* searches for solutions in a solution space that satisfy specific constraints and minimize a certain cost function, which is not only restricted to searching for the shortest path in a graph. If the solution space is small, it is possible to use an exact algorithm that can find an optimal solution to such a problem in a finite amount of time. However, if the size of the solution space becomes too large, it is no longer possible to find a solution in a finite time, and different types of algorithms must be used [De Ryck et al., 2020].

Examples of optimization-based algorithms are the *boolean satisfiability problem* (SAT), which is the problem of evaluating whether a feasible solution to a given set of Boolean clauses exists, *satisfiability modulo theories* (SMT), which is a generalization of SAT in which one is no longer restricted to Boolean variables, *Linear programming* (LP), which is an optimization framework where the objective function is linear and the variables involved are subject to linear constraints, and *quadratic programming* (QP), which is an optimization framework where the objective function is quadratic (instead of linear) and the variables involved are subject to linear constraints. Each of these optimization-based algorithms is evaluated in more detail in [Vader, 2022].

2.1.4 Metaheuristic algorithms

In [Osman and Laporte, 1996], a metaheuristic was defined as "an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, [in which] learning strategies are used to structure information in order to find efficiently near-optimal solutions." Thus, *metaheuristic algorithms* are high-level strategies for exploring search spaces, which can use a variety of methods [Blum and Roli, 2003]. Among these strategies are *Ant Colony Optimization* (ACO), which is based on the behavior of ants in real-life, *Evolutionary Computation* (EC), which is inspired by natural selection in nature where living beings also adapt to their environment, *Particle Swarm Optimization* (PSO), which is based on swarm intelligence to imitate social behaviors found in nature, *Tabu Search* (TS), which uses prior solutions in order to escape from local optima and explore outside its local neighborhood, and *Simulated Annealing* (SA), which also makes use of the idea that in order to escape from a local optimum, it might first be required to select a worsening solution. Each of these metaheuristic algorithms is also evaluated in more detail in [Vader, 2022].

In the general case, metaheuristic algorithms can be used when the solution space of an optimization problem, which is not restricted to searching a graph, becomes so large that exact optimization algorithms, such as SAT, SMT, LP or QP, cannot find a solution in finite time. In that case, one can choose to settle for a near-optimal solution in finite time using a metaheuristic algorithm [De Ryck et al., 2020].

2.1.5 Summary on path planning

Metaheuristic algorithms like ACO, EC, PSO, TS and SA can be very useful algorithms because they don't make any assumptions about the layout and can easily react to dynamically changing layouts. However, they have some drawbacks. The main drawback of these algorithms is that they generally do not scale that well. For all these algorithms a lot of potential solutions must be evaluated before converging to a near-optimal solution. Therefore, as the problem complexity increases, metaheuristic algorithms can become very computationally expensive. Furthermore, metaheuristic algorithms are approximate and usually non-deterministic [Blum and Roli, 2003], which is a drawback where optimal solutions are desired.

The main advantage of optimization-based algorithms is that they are exact and constraints can be added very easily [De Ryck et al., 2020]. For example, if one would want to restrict a specific node or edge from being used, one simple constraint can be added to the problem definition in order to enforce this restriction. However, also optimization-based algorithms have the major drawback that they can become very computationally expensive when more and more constraints are added.

Dijkstra's algorithm, the A* algorithm and the D* algorithm are all very computationally efficient. Moreover, they are all complete and optimal [Hart et al., 1968], given that the heuristic cost $h(x)$ is admissible. Because a new path has to be planned every time a vehicle reaches a pick-up or drop-off station, it is crucial that a new path can be computed as fast as possible to reduce the time vehicles spend waiting for a new path. Mainly because of this reason, graph search algorithms like the A* algorithm and the D* (Lite) algorithm are the most commonly used algorithms for path planning of mobile robotics [De Ryck et al., 2020]. As was concluded in [Kim et al., 2018], there is a tipping point where using the D* algorithm becomes more efficient than using the A* algorithm, depending on the system characteristics.

2.2 Traffic control

In this section, the functions of traffic control are introduced and examined in more detail, the concepts of centralized, decentralized, and hierarchical control are introduced, and several control approaches that exist literature are studied. All of these concepts can serve as inspiration for the hierarchical control strategy that is developed in Chapter 4.

2.2.1 Functions of traffic control

As was introduced in Section 2.1, path planning algorithms can be used to find a path from one location to another while minimizing some objective function. In a multi-vehicle system, where every vehicle performs one or more jobs, the shortest path from pick-up station to drop-off station often crosses the paths of other vehicles. In order to coordinate these vehicles and make sure that they can successfully execute their planned paths, a traffic controller has to be developed. A traffic controller has three major responsibilities [De Ryck et al., 2020]:

1. **Collision avoidance**
2. **Deadlock handling**
3. **Livelock handling**

Negligence of any of these three major responsibilities can deal serious damage to the (performance of the) multi-vehicle system, in the form of damaged vehicles, a system that is permanently stuck in a state where no progression is made, or a system that constantly switches between system states without achieving some desired state. Another optional yet less crucial function of the traffic controller is **congestion avoidance**. Each of these functions is briefly introduced in the remainder of this section.

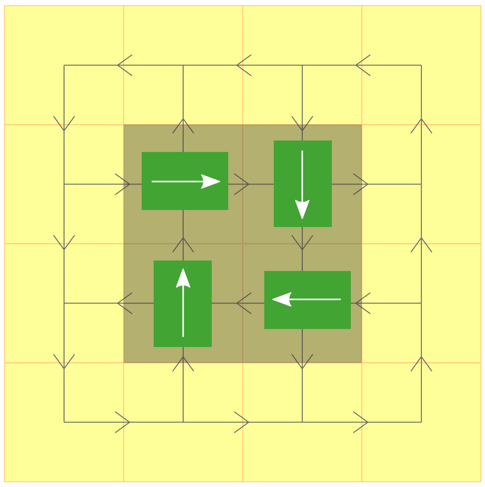
Collision avoidance

As the name suggests, in *collision avoidance* the goal is to avoid collisions between vehicles. A rather straightforward method that can be used is to attach a safety scanner to the front of the vehicle, which keeps up a safety zone. When an object is identified within this zone, the vehicle slows down and eventually stops. As soon as the object is removed from the zone, the vehicle continues its path. More advanced methods can be used to manoeuvre a vehicle around an obstacle or plan a totally different path [De Ryck et al., 2020].

Other methods of collision avoidance are based on the principle that two vehicles are not allowed to use the same space at the same time, an example of which uses a grid-based system. In a grid-based system, the controller keeps a virtual representation of the layout, where the layout is subdivided into a grid of tiles (sometimes referred to as zones). Tiles do not overlap, can only hold one vehicle at the same time, and are large enough such that vehicles can rotate within the borders of the tile [Fransen et al., 2020]. Before a vehicle can use a new tile, the controller must reserve the tile for this vehicle. Because a tile can only be reserved by one vehicle at a time, collisions are avoided [De Ryck et al., 2020]. In literature, this also is referred to as zone-based control.

Deadlock handling

Besides collision avoidance, *deadlock handling* is a crucial responsibility of a traffic controller. Here, *deadlock* is defined as a state where a group of vehicles is waiting for each other forever, also called a deadlock cycle or a circular wait, because the next tile in the path of each vehicle is the last reserved tile of another vehicle. Since it is the last reserved tile of the other vehicle, this vehicle cannot release its reservation before it has reserved the next tile of its own path [van Weert, 2019]. In particular, such a situation is called a *mono-cycle deadlock*. An example of a mono-cycle deadlock can be seen in Figure 2.2a. Additionally, in Figure 2.2b, an example of a *multi-cycle deadlock* has been visualized. A multi-cycle deadlock can occur when a mono-cycle deadlock is avoided, and is characterized by the presence of multiple impending deadlock cycles. In this case, a vehicle could reserve its next tile because it is not occupied, but doing so would lead to deadlock [van Weert, 2019].


(a) Mono-cycle deadlock.

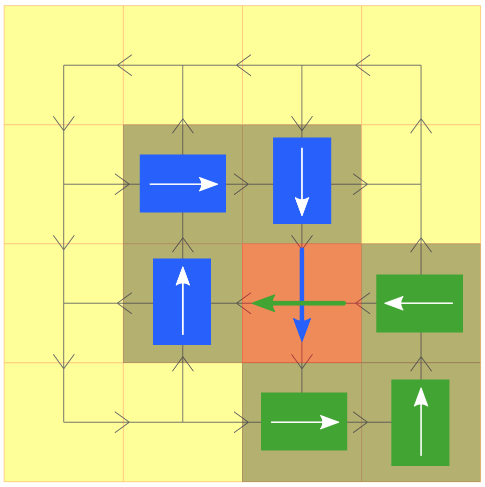

(b) Multi-cycle deadlock.

Figure 2.2: Visualization of the difference between a mono-cycle deadlock and a multi-cycle deadlock.

Deadlocks must be handled appropriately, because otherwise the system permanently remains in a state where no progression is made. Deadlock handling strategies can be distinguished into three categories:

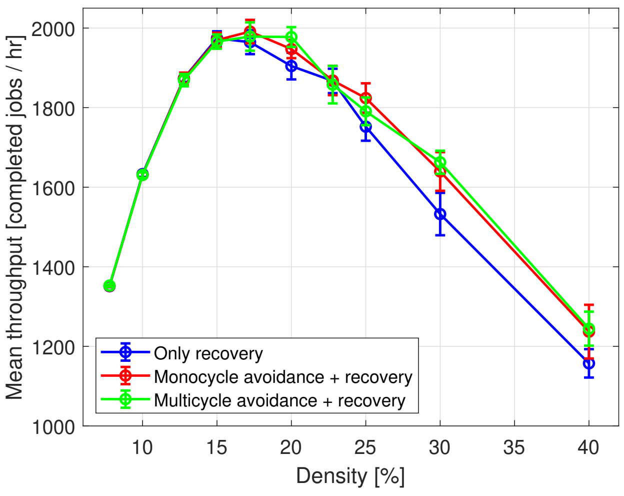
1. Deadlock prevention
2. Deadlock avoidance
3. Deadlock detection and recovery

First of all, *deadlock prevention* aims at preventing deadlocks from occurring by defining a set of rules, which are defined a priori, in an offline manner [Li et al., 2012]. For example, these rules can be applied to the design of the system layout or to the controller of the vehicles. An example of this is the requirement of a well-formed infrastructure, as is defined in [Cáp et al., 2015].

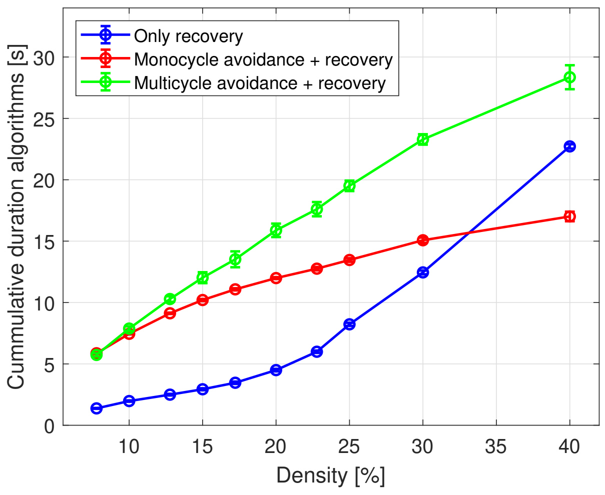
Next, *deadlock avoidance* aims at avoiding deadlocks in an online manner, for example by allocating tiles to vehicles only if the resulting state does not lead to deadlock [van Weert, 2019]. Every time a new tile needs to be reserved for a vehicle, the controller evaluates in advance whether allocating the requested tile to said vehicle will lead to deadlock. If a deadlock situation is foreseen by the controller, the requested tile is not allocated to the vehicle. On the other hand, if the controller foresees that no deadlock will occur, the tile can be allocated to the vehicle [van Weert, 2019].

Lastly, *deadlock detection and recovery* allows deadlocks to occur. However, once a deadlock is detected, it is resolved using online procedures [Somers, 2022]. This can be useful if deadlocks only occur occasionally. Moreover, deadlock detection and recovery is a necessity in situations where unexpected deadlocks occur, caused by disturbances or exceptions that could not be predicted and avoided by deadlock avoidance [Somers, 2022].

[Vis, 2006] believed that only using deadlock detection and recovery leads to a worse performance than using it in combination with deadlock avoidance, since detection and recovery is reactive instead of proactive. In [Somers, 2022], the throughput of a grid-based system was compared for the situation where the system only contained deadlock detection and recovery as opposed to deadlock detection and recovery *and* deadlock avoidance. It was shown that for certain layouts, only using deadlock detection and recovery *can* match the throughput of a system where both deadlock detection and recovery and deadlock avoidance are present. Nevertheless, it is unlikely that throughput can be improved by disabling deadlock avoidance. In order to explain this, in Figure 2.3a the mean throughput is plotted against the vehicle density for different deadlock handling variants on a particular layout. For all variants, the highest throughput is reached for a vehicle density between 15 to 20 [%], and the throughput of all variants is almost identical around a vehicle density of 17.5 [%]. However, as can be seen in Figure 2.3b, the computation time of pure deadlock detection and recovery takes roughly three to four times less time than mono- and multi-cycle deadlock avoidance in combination with deadlock detection and recovery.



(a) Mean throughput against vehicle density.



(b) Cumulative computation time against vehicle density.

Figure 2.3: Comparison of the mean throughput and cumulative computation time of three deadlock handling variants for different vehicle densities [Somers, 2022].

As was concluded in [Somers, 2022], the benefit of only using deadlock detection and recovery is the significant decrease in computation time for deadlock handling. Only using deadlock detection and recovery as a deadlock handling technique is the best choice if layouts are insensitive to deadlocks or computation time needs to be trimmed. However, it was shown that the system throughput greatly depends on the length of the deadlock detection interval. By using deadlock detection and recovery it can at least be guaranteed that the system becomes deadlock-free.

Livelock handling

The next responsibility of a traffic controller is handling livelocks. A *livelock* is defined as a situation where the system state constantly changes, typically in a circular manner, but fails to achieve some desired state in a finite time [Krnjak et al., 2015]. An example of such a situation is a scenario where two persons walk towards each other, but both of them continuously move aside in order to be polite and to let the other person pass. Yet, because both of them are trying to be polite and let the other go first, no actual progress is made.

In order to make sure such a situation does not occur in a multi-vehicle system, a couple of precautions can be taken. Because some deadlock handling strategies involve a particular path planning approach, livelock might be resolved simultaneously [van Weert, 2019]. An example of this is when paths are planned statically, i.e. they are determined before the vehicles start moving and cannot change over time. In contrast, if paths are planned dynamically, the system becomes susceptible to livelocks. Additionally, the risk that livelock occurs becomes smaller when the number of tiles that is reserved ahead increases or when the interval between periodic replanning of the paths increases [van Weert, 2019]. In the most extreme case, the entire path from pick-up to drop-off station is reserved at once, essentially meaning that paths are never replanned, i.e. the time between replanning is infinity.

Congestion avoidance

Congestion avoidance aims at smartly combining path planning and traffic control in order to avoid congested areas, eventually leading to a reduced travel time. In Section 2.1 path planning algorithms were introduced as algorithms that can be used to find a path from one location to another while minimizing some objective function. Minimizing the travel distance of a vehicle might lead to the shortest path distance-wise, but this does not necessarily mean said path is also the shortest path time-wise. An important factor to take into account is the congestion in an environment.

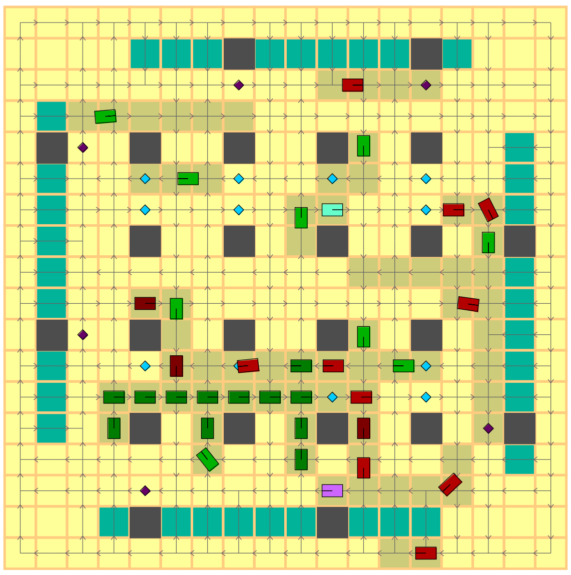


Figure 2.4: Snapshot of congestion in the ParcelGrid layout with 36 vehicles.

As more vehicles are added to an environment, it becomes more likely that at some point a vehicle has to wait for other vehicles while executing its path in order to avoid collisions, deadlocks and livelocks. As an example, a snapshot of the *ParcelGrid* layout with 36 vehicles has been visualized in Figure 2.4. In this snapshot, the green and red rectangles represent loaded and unloaded vehicles, respectively. As can be seen, the bottom region contains a lot of vehicles that are waiting for each other. To reduce the time vehicles waste waiting for each other, some form of congestion avoidance can be implemented. As was already introduced in Section 2.1.1, [Fransen et al., 2020] and [Yuan et al., 2016] introduced adjusted versions of the A* algorithm in an attempt to avoid congested paths.

2.2.2 Centralized and decentralized control

Within traffic control, a distinction between centralized and decentralized (i.e. distributed) control approaches can be made.

As defined in [Fransen, 2019], in a *centralized control approach*, a central controller is present which is in control of all vehicles. In doing so, it may use all available information about all vehicles, such as planned paths, destinations, current locations and speeds of all vehicles. Because global information is available, centralized approaches generally aim at reducing the model complexity and obtaining optimal solutions or increasing the optimality of solutions as much as possible [Liu et al., 2019]. However, as soon as the number of vehicles in a system increases, centralized approaches suffer from significant flaws in terms of computation demands, which can be a critical problem. Moreover, these approaches include a low level of tolerance to unexpected events and dynamic changes in the layout [Krnjak et al., 2015].

Summarizing, centralized controllers can be characterized by the following features [De Ryck et al., 2020]:

- Access to global information
- Globally optimal
- Reduced model complexity
- Computationally expensive; mainly suitable for small scaled systems
- Low level of tolerance to changes

Additionally, in [Fransen, 2019] a *decentralized control approach* (or distributed control approach) is defined as an approach in which there is no central controller which is in control of all vehicles. A distinction can be made between different forms of decentralized control:

a. Every vehicle has its own controller.

b. Every controller is in control of a set of vehicles. This set of vehicles does not change during the simulation.

c. Every controller is in control of a specific area. This area does not change during the simulation.

In all three forms there exist multiple controllers that only have access to local information. Because only local information is available, global optimality of decentralized approaches is less straightforward. Global optimality can be obtained, for example when different controllers are decoupled in such a way that combining their local optima forms a global optimum, but this is not necessarily the case [De Ryck et al., 2020]. Because the computational burden is distributed over multiple controllers, decentralized approaches are usually more suitable for larger scaled systems than centralized approaches. However, because all local controllers have to be coordinated such that no collisions and deadlocks occur, the model usually becomes more complex. Furthermore, decentralized control approaches are characterized by a high level of vehicle autonomy and flexibility [Krnjak et al., 2015], and they mainly focus on fulfilling real-time requirements and locally resolving congestion and deadlock [Liu et al., 2019].

So, decentralized controllers can be characterized by the following features [De Ryck et al., 2020]:

- Access to local information
- Usually not globally optimal
- Increased model complexity
- Computationally less expensive; also suitable for larger scaled systems
- High system flexibility

Centralized control approaches in literature

In literature, there exists a wide variety of centralized control approaches, each with their own advantages, disadvantages, assumptions and objectives. Some control approaches that can serve as inspiration for the hierarchical control approach that is developed in Chapter 4 are briefly introduced in this section, though a lot more centralized control approaches were studied in [Vader, 2022].

In [Małopolski, 2018] a centralized approach is proposed to prevent collisions and deadlocks in grid-based systems, based on a chain of elementary reservations. Whenever a vehicle receives a job, it places a reservation in the queue of all the tiles in its path. A vehicle can only traverse a tile if its reservation is first in queue for said tile. As a result of this, the first vehicle can always finish its path because its reservations are always first in the queue. As soon as the first vehicle has used a tile, its reservation is removed. Now, the vehicle that was second in queue becomes first in queue, and can use the tile. This pattern is repeated for all other vehicles and all other tiles. In this way, collisions and deadlocks are avoided. However, as a result of this approach, the vehicles that are last in the queue have enormous waiting times. In this research, the speed of every vehicle is assumed constant, meaning that acceleration is neglected. This hugely oversimplifies the problem, because in reality vehicles might have to decelerate, accelerate and rotate on certain tiles in order to prevent collisions and deadlocks or make turns.

[Xiao et al., 2020] introduced a centralized collision and deadlock avoidance mechanism for unidirectional guide-path networks. Because unidirectional paths are assumed, only collisions at intersections are of interest here. First, a mutually exclusive semaphore is defined for each node and edge. In essence, such a semaphore is used to control the availability of shared resources in a concurrent system. Simply said, if there is no space left on a node or edge, a vehicle cannot use it. A bidding mechanism-based strategy is used to optimize the traffic sequence of all path intersections, based on the task urgency and traffic urgency of the vehicle. Optimizing this bidding mechanism can greatly increase the average throughput and reduce the congestion and risk of deadlock in the system.

[Yeh and Yeh, 1998] presented a centralized algorithm in order to predict and avoid deadlocks in real time for a unidirectional grid-based layout. Whenever a vehicle wants to travel to the next tile in its path, all other vehicles evaluate the next tile in their own path. If a circular wait is predicted by any of the other vehicles or if the next tile of the vehicle of interest is already occupied, this vehicle has to wait. However, if no circular wait is predicted and the next tile of the vehicle of interest is also not occupied by another vehicle, then the vehicle of interest can enter the next tile. In this way, deadlock is avoided.

In [Moorthy et al., 2003] a centralized deadlock prediction and avoidance algorithm is presented which is used in a container terminal environment. First, mono-cycle deadlock is predicted by checking whether a cyclic request of resource occurs, which implies a deadlock. If deadlock is indeed predicted, the system can choose to reroute the involved AGVs in order to avoid deadlock, or wait until the deadlock is cleared. Next, the algorithm predicts whether multi-cycle deadlock occurs by checking for indirect dependencies between AGVs. However, this severely complicates the algorithm and significantly increases the computational expensiveness.

Decentralized control approaches in literature

Besides the centralized control approaches described in the previous section, there also exists a wide variety of decentralized control approaches in literature which aim at increasing the scalability of control systems. Some control approaches that can serve as inspiration for the hierarchical control approach that is developed in Chapter 4 are briefly introduced in this section, though a lot more centralized control approaches were studied in [Vader, 2022]

In [Fanti et al., 2015] and [Fanti et al., 2018] a decentralized grid-based control approach is introduced in order to avoid collisions and deadlocks. At each timestep, vehicles communicate with all other vehicles that are within certain a communication radius. On the one hand, if a vehicle is the only vehicle that wants to enter a tile, it is free to do so. On the other hand, if another vehicle wants to enter the same tile, the vehicle with the highest priority goes first. This is the vehicle that is the furthest away from its destination. If a vehicle wants to enter a tile that is already occupied by another vehicle, it has to wait on its current tile and sends a relocation request to the other vehicle. In the next timestep, the vehicle that received the relocation request tries to relocate. If the zone that this vehicle wants to relocate to is also already occupied, this vehicle also sends a relocation request to one of its neighbors. Because it is assumed that there is always at least one free zone in the entire system, it is always possible for at least one vehicle to relocate to another zone. This concept is used to ensure that the system stays ‘alive’. Crucially, the time required to travel between two adjacent tiles is assumed constant and equal to the time unit between events in order to allow all vehicles to move synchronously, which is a large oversimplification.

[Bnaya et al., 2013] introduced a multi-agent pathfinding (MAPF) algorithm where agents are considered self-interested. Self-interested agents choose their paths based on what is most beneficial for themselves. Conflicts are detected locally and resolved by replanning the path of one of the vehicles that is in conflict. Additionally, a taxation scheme is introduced which penalizes agents for travelling over a specific edge or node at a specific time. Consequently, self-centered agents aim to avoid paths that have not been penalized, potentially causing fewer conflicts. In this paper, time is discretized such that an agent can move to its next zone or stay in its current zone during each timestep.

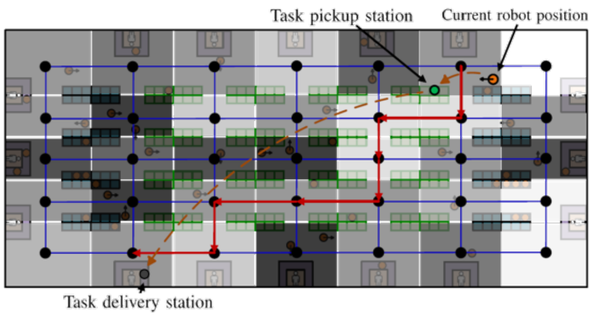
In [Cáp et al., 2015] a decentralized control approach for online multi-robot path planning was introduced, called continuous best-response approach (COBRA). Whenever a robot receives a new job, this robot plans its path such that it avoids collisions with other robots that are already executing their path. Thus, every new path is planned such that it avoids robots whose job was issued earlier. For a well-formed infrastructure, i.e. an infrastructure where any robot standing on an endpoint can never entirely prevent other robots from moving between two other endpoints, this approach is proven to always provide a path where no collisions occur. In this paper, acceleration limits are also neglected.

In [Jäger and Nebel, 2001] decentralized collision avoidance and deadlock detection and resolution for a multi-agent system is introduced. Whenever the distance between two agents drops below a certain value, the agents exchange information about their planned paths and determine whether they are in danger of a collision. If a possible collision is detected, they monitor their movements and, if necessary, insert idle times between certain segments of their paths in order to avoid the collision. Additionally, if an existing deadlock is detected, the agents are asked to plan an alternative path until the deadlock is resolved. Global coordination of all agents is achieved solely by using local communication between agents.

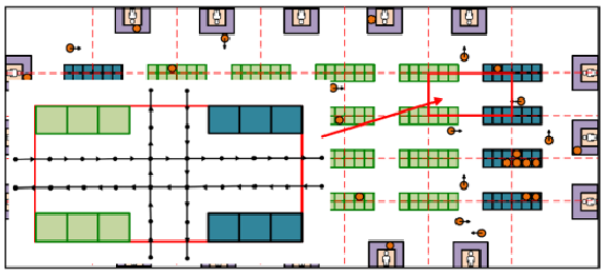
Hierarchical control approaches in literature

So far, a distinction was made between centralized and decentralized algorithms. However, there also exist hierarchical control approaches, in which the control structure is arranged as a hierarchical tree [Sieniutycz, 2020]. In some cases, these hierarchical control approaches are a combination of both a centralized and a decentralized control approach, and aim to exploit the benefits of both approaches [Sabattini et al., 2016].

In [Liu et al., 2019], a hierarchical control strategy is introduced which can be distinguished into two layers: centralized high-level task planning and decentralized low-level motion coordination. On the higher level, the layout is first partitioned into sectors and a sector-level topology graph (i.e. a network of nodes and edges) is generated. Then, a dynamic heat-map is generated which indicates the current distribution of the vehicles over the sectors. Using this heat map, a high-level (i.e. sector-level) path planning is generated using the A* algorithm, which indicates the sequence of sectors to be used. An example of such a sector-level path can be seen in Figure 2.5a, indicated by the red path. The high-level path planning contributes to reducing the probability of congestion, which in turn aims at improving the throughput and avoiding deadlocks. On the lower level, i.e. within each sector, a road-level topology graph is generated which contains only unidirectional roads, as can be seen in Figure 2.5b. Based on this road-level topology graph a conflict-based search (CBS) is performed in order to obtain collision-free paths, as is introduced in [Sharon et al., 2015]. The purpose of this low-level path planning is to reduce the complexity of the coordination problem while maintaining the solvability of the whole life-long problem. It is assumed that at every timestep, vehicles can stay in their current location or move to the next road segment.



(a) Sector-level path in [Liu et al., 2019].



(b) Road map used in [Liu et al., 2019].

Figure 2.5: Heat map and road map used in [Liu et al., 2019].

In [Digani et al., 2014], another two-layer hierarchical control strategy is introduced. A distinction can be made between a centralized high-level topological layer and a decentralized low-level route map layer. Coordination among vehicles is obtained by exploiting centralized information and decentralized coordination. On the topological layer, the layout is first partitioned into sectors and a unidirectional high-level graph is generated. After this, a path is planned from the current sector to the destination sector which contains the goal node. This path is planned using a combination of the D* algorithm and model predictive control (MPC). Because the distribution of vehicles over the sector changes over time, the high-level paths are replanned in an attempt to reduce traffic congestion. On the road map layer, i.e. inside each sector, vehicles are coordinated in a decentralized manner. Path planning is performed using the A* algorithm, where a negotiation mechanism and a resource allocation strategy are used to avoid collisions and deadlocks. In Figure 2.6, the sector division and roads inside each sector have been visualized. The computation time of the algorithm in this research is proven to scale linearly with the number of vehicles in the system. In [Sabattini et al., 2016] the work of [Digani et al., 2014] is continued, and the sector division is performed in a dynamic manner using geodesic Voronoi partitioning.

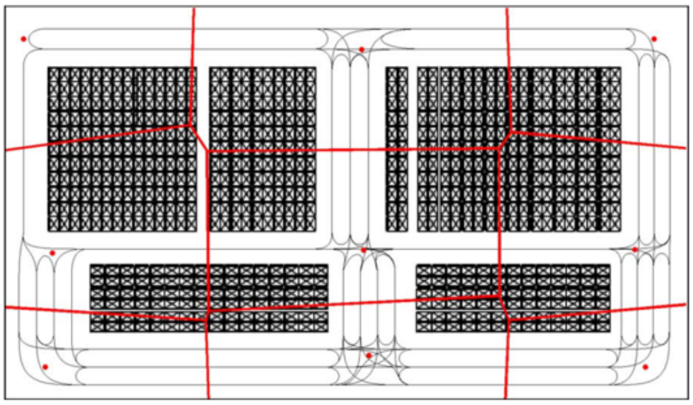


Figure 2.6: Sector division used in [Digani et al., 2014].

2.2.3 Summary on traffic control

In order to coordinate vehicles in a multi-vehicle system and make sure that they can successfully perform their jobs, a traffic controller has three major responsibilities: collision avoidance, deadlock handling and livelock handling. Omission of any of these three responsibilities can lead to damaged vehicles, a system that is permanently stuck in a state where no progression is made, or a system that constantly switches between system states without achieving some desired state. Another optional yet less crucial function of the traffic controller is congestion avoidance.

In general, a distinction between three types of control structures can be made, namely centralized approaches, where a central controller is present which is in control of all vehicles and has access to global information, decentralized approaches, where multiple controllers exist over which the computational burden is distributed and who all have access to local information, and hierarchical control approaches, in which centralized and decentralized control approaches are usually combined.

Because the centralized controller in centralized control approaches has access to global information of all vehicles, a global optimum can be reached relatively easily. However, these approaches are considered to be computationally expensive and unsuitable for larger scaled systems, which is a critical problem. Centralized approaches are also considered less flexible and robust.

Because the computational burden is distributed over multiple controllers in decentralized control approaches, these approaches are generally less computationally expensive and more suitable for larger scaled systems than centralized approaches. However, because of the fact that only local information is available, it is less straightforward to reach a global optimum. Additionally, by using only local information, it is more difficult to avoid congestion and deadlocks in comparison to centralized approaches. Moreover, all local controllers have to be coordinated appropriately, which increases the model complexity and (possibly) communication demands.

2.3 Proposed control approach

In the preceding part of this chapter, a variety of path planning algorithms and traffic control approaches were studied. Based on this research, it can be concluded that most centralized control approaches fall short in situations where scalability is one of the desired properties. Furthermore, although decentralized control approaches are usually a lot less computationally expensive, their model complexity is typically higher. In particular, implementing deadlock and congestion avoidance becomes significantly more complex if only local information is available. To the best of our knowledge, there exists no control approach in literature that satisfies all predetermined requirements introduced in Section 1.2. Although a lot of research can be used to take inspiration from, all approaches have shortcomings.

Based on this conclusion, the decision has been made to create a *two-layer hierarchical control structure*, largely inspired by the ideas presented in [Liu et al., 2019]. Using this hierarchical control structure, the aim is to exploit the scalability of decentralized approaches while limiting the model complexity and typically non-optimal character of decentralized approaches.

The proposed hierarchical control structure is divided into the following two layers, namely *global path planning* and *local traffic control*, which are characterized as follows:

1. On the global level, the existing layout is partitioned into larger sectors which represent the topological relationship between different areas of the layout. Based on this sector partitioning, a global layout is defined that aims at creating a general impression of the full environment, while disregarding most details. Using Dijkstra's algorithm, a centralized global controller determines global paths for all vehicles, which serve as an approximation of the shortest path a vehicle can take to traverse from one location to another location. The main motivation for using Dijkstra's algorithm is its simplicity and computational efficiency. Moreover, the general idea is that the global layout must be simple enough such that Dijkstra's algorithm is able to search through the layout in a short amount of time.

2. On the local level, i.e. within every sector, traffic is controlled using a local controller. Every sector has its own decentralized local controller, which only coordinates the vehicles that are located in its own sector and only has access to local information. Every sector is subdivided into a grid of tiles, which do not overlap and can only hold one vehicle at the same time. Therefore, every sector can be controlled as if it were a grid-based system. The desired properties of the controller in a particular sector depend on the characteristics of that sector. For this reason, it is allowed that the properties of a controller vary per sector, as long as vehicles do not collide, do not end up in deadlock, and are able to successfully perform their jobs.

3 Grid-based control model

As the name suggests, in Vanderlande's grid-based control model vehicles are controlled using a grid-based system. Vehicles are controlled by the centralized supervisory controller, which is largely built on the path planning algorithm presented in [Fransen et al., 2020] and the deadlock avoidance algorithm presented in [Fransen et al., 2022].

In this chapter, an introduction to the grid-based model setup is given first, in which all participants and other objects in the simulation are described. Thereafter, the procedures in the supervisory control cycle are explained in more detail. Lastly, the communication between vehicles and controller is elaborated upon.

3.1 Grid-based model setup

In Section 2.2.1 grid-based control was already introduced as a method that can be used in order to enforce collision avoidance. In a grid-based system a controller keeps a virtual representation of the environment, which is subdivided into a grid of tiles (sometimes referred to as zones). Tiles do not overlap, can only hold one vehicle at the same time, and are large enough such that vehicles can rotate within the borders of the tile [Fransen et al., 2020]. In addition, a vehicle is only allowed to use a tile if the tile is reserved for this particular vehicle. In Vanderlande's grid-based control model the controller only reserves a new tile for a vehicle if it passes the availability check and deadlock check. These checks verify that the tile is not yet reserved by another vehicle and that reserving this tile does not result in a deadlock. As a result of this reservation system, both collisions and deadlock can be enforced.

3.1.1 Discrete-event simulation

The grid-based control model is simulated in the form of a discrete-event simulation. Such a discrete-event simulation simulates the system behavior as a discrete sequence of events. An event list is maintained to keep track of the events that have been planned in the future. As a result of this, a discrete-event model usually does not evolve with equidistant time steps. Each event occurs at a particular time and changes the state of the system, depending on the type of event. Furthermore, it is assumed that no changes occur in the state of the system between two consecutive events [Robinson, 2004]. Whenever the simulator finishes the execution of an event, it determines what is the next event in the event list, updates the simulation time to the time corresponding to the next event, and executes this event at the corresponding time. The pseudocode corresponding to the discrete-event simulation is described in Algorithm 1.

Algorithm 1 Pseudocode of the discrete-event simulation

```
1: Create simulation participants
2: Initialize model
3: while next event time $\leq$ end time simulation do
4:     Update simulation time to next event time
5:     Execute all events that occur at current event time in randomized order
6:     Determine next event time
7: end while
8: Perform post-processing
```

We define a *participant* in the simulation as an object in the simulation that has at least one type of event that it can execute. The relevant participants in the grid-based control model are the controller, mediator, job provider and vehicles (to be more specific: the vehicle logic and vehicle kinematics). Each of these participants is introduced in Section 3.1.2.

Every event is related to a time at which the event is executed, a participant by which the event is executed, and the type of event that must be executed by this participant. It can be the case that the next event of multiple participants occurs at the same time. In that situation, the order in which the events of the participants are executed is randomized in order to prevent bias for specific participants.

3.1.2 Participants in the simulation

In this section, each of the relevant participants in the simulation is introduced.

Controller The *controller* is responsible for controlling (i.e. managing) the vehicles in the system. The controller has two main purposes, which are path planning and traffic control. Furthermore, traffic control can be distinguished into collision avoidance, deadlock handling and livelock handling.

The controller which is currently used in the grid-based control model is referred to as the supervisory controller. This is a centralized controller, which contains the path planning algorithm described in [Fransen et al., 2020], where node weights are updated over time based on the time vehicles spend standing still on their respective nodes, and turning costs are also included. Furthermore, the deadlock avoidance and detection algorithm described in [Fransen et al., 2022] is used, which guarantees that the system remains deadlock-free within a limited number of known future movements. The control cycle that is periodically performed by the supervisory controller is explained in full detail in Section 3.2. During the simulation, *vehicle update messages* are periodically sent from the vehicles to the controller, which can be processed by the controller to determine what control action should be performed upon this vehicle. Based on this controller action, the controller sends back *vehicle commands messages* to the vehicles. Each of these types of messages is explained in more detail in Section 3.1.4.

Mediator The *mediator* acts as the communication network between the controller and the vehicles in the simulation.

The controller and vehicles (to be more specific, the vehicle communication) are considered communication objects, which can send messages to and receive messages from the mediator. Incoming messages always contain a source, destination, message type and the actual message itself. Whenever a message is sent to the mediator, the mediator determines at what time this message should be executed (i.e. forwarded). Depending on whether or not a delay is applied to the message, the time of execution of this next event is equal to the current time, i.e. the time the message is received, or sometime in the future. If the message queue is empty upon arrival of an incoming message, the mediator always adds this message to the front of the message queue, an example of which can be seen in Figure 3.1. Additionally, the mediator sets its next event at the time of execution of the message, which is when the message is forwarded.

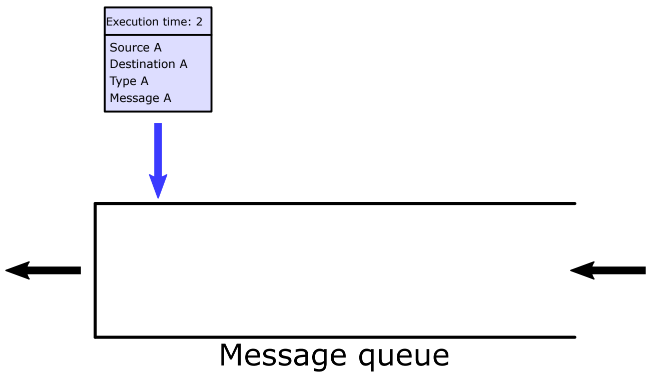
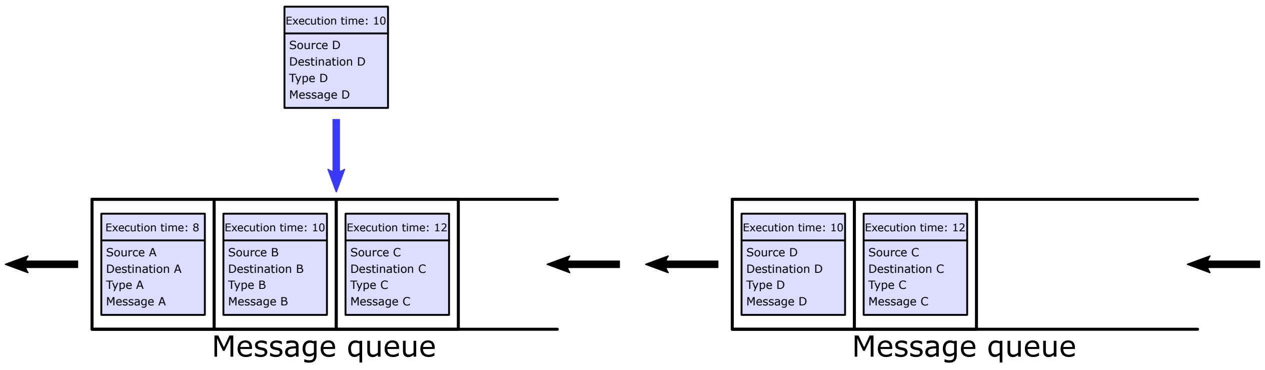


Figure 3.1: The mediator adds an incoming message to the front of an empty message queue.

If the message queue is not empty upon arrival of an incoming message, the mediator adds this message to the message queue behind all messages with the same or lower time of execution. This can be seen in Figure 3.2a, where incoming message D is placed in the message queue *after* messages A and B because it has a later time of execution than message A and the same time of execution as message B, but *before* message C because its time of execution is lower. As soon as *the mediator's next event* is in front of *the event list of the discrete-event simulator*, the mediator forwards all messages in the message queue that are scheduled for the current moment of time to their respective destinations. Messages with the same time of execution are forwarded in the same order as the order in which they were received. In Figure 3.2b a snapshot has been shown of the moment right after message B has been forwarded, meaning that it is now message D's turn to be forwarded. Note that message A was already forwarded in a different mediator cycle at $T = 8$.

Depending on the message type of each message, the corresponding method of the destination object is called. Lastly, all messages that were forwarded in the current mediator cycle are removed from the message queue and the next event of the mediator is set to the message with the smallest time of execution.



(a) The mediator adds an incoming message to a non-empty message queue.

(b) Message A and B have been executed, it is now message D's turn to be executed.

Figure 3.2: Visualization of message queue behavior.

Job provider The *job provider* is responsible for generating jobs and assigning jobs to the vehicles.

During the initialization of the model, an initial batch of jobs is generated. Depending on the model settings, a new job is also generated when a vehicle agent reserves a job, or when a vehicle starts or finishes the execution of a job. As a result of this, it is assumed that there are always available jobs in the remainder of this thesis.

The decision to assign a vehicle to a particular job is made using a job assignment strategy. The goal of job assignment is to efficiently distribute the jobs in the system over the vehicles. Periodically, the job provider checks whether there are any vehicles that are available for a new job. If there is at least one available vehicle, the job provider also checks whether there are any available jobs. If there is at least one available vehicle and one available job, job assignment takes place based on a weighted cost function. The design and implementation of the job assignment strategy are described in full detail in [Jacobs, 2020].

Vehicles The *vehicles* represent the physical behavior of the differential drive vehicles in the model.

Within a vehicle, a distinction can be made between three different functionalities: the *vehicle logic*, *vehicle kinematics* and *vehicle communication*, which are all closely related to each other.

Logic The logic part of the differential drive represents the control logic behind each vehicle. Among other things, it keeps track of the vehicle status (i.e. is the vehicle waiting, moving, loading, unloading or idle), the vehicle capabilities such as its maximum speed, acceleration, deceleration and rotation speed, and the path that has to be driven (i.e. the waypoints that have to be followed). Crucially, the vehicle logic also periodically sends vehicle update messages to the controller.

Kinematics The kinematics functionality of the differential drive represents the vehicle kinematics of each vehicle. Vehicle kinematics keeps track of properties related to the current vehicle position, speed, and acceleration, which event should be executed next by the vehicle logic, and constant information such as vehicle shape, vehicle weight and battery capacity. Every time the vehicle kinematics executes its event, the next kinematic state is determined based on the previous kinematic state using the equations of motion. This kinematic state is later used by the vehicle logic to send vehicle updates to the controller.

Communication The communication part of the differential drive facilitates the communication from the vehicle to the controller and from the controller to the vehicle. On the one hand, every time the vehicle logic sends a vehicle update message to the controller, this message is forwarded by the vehicle communication. On the other hand, every time a vehicle commands message is received from the controller, the vehicle communication processes the incoming message. In a sense, vehicle communication acts as a conduit for messages from and to the logic of the vehicle.

Note that the vehicle communication is technically not a participant in the simulation because the vehicle communication is always called by the vehicle logic. It is included with the vehicle logic and kinematics for the sake of completeness.

Besides the controller, mediator, job provider, vehicle logic and vehicle kinematics, other participants in the simulation are the safety scanner, collision detector and visualizer. These participants are not introduced in more detail, as their workings are irrelevant in the remainder of this thesis.

3.1.3 Other objects in the simulation

Besides the participants in the simulation, there are two more relevant objects that require an introduction, namely the vehicle agents and the layout. These objects cannot be considered participants, as they do not have their own events in the simulation.

Vehicle agents A *vehicle agent* is a digital representation of a vehicle within the controller.

Every vehicle agent is linked to one particular vehicle that it represents. The controller is aware of the existence of the vehicle agents but is unaware of the existence of their vehicle counterparts. Information that is kept within the vehicle agent is, among other things, the previous and current vehicle state (i.e. coordinates, speed and heading), the current vehicle path and the current vehicle job. Based on this available information and incoming vehicle update messages from the vehicle, the controller is able to control a vehicle. Vehicle agents do not perform any actions on their own, but only when called by the controller. Therefore, a vehicle agent is unable to request something of a controller on its own initiative.

A visual representation of the difference between vehicle agents and their vehicle counterparts has been shown in Figure 3.3. Vehicles are physically located inside the layout, whereas the vehicle agents are 'on the sideline' with the controller. This figure also illustrates how messages are sent from the vehicles to the controller and from the controller to the vehicles, with the mediator acting as the communication network.

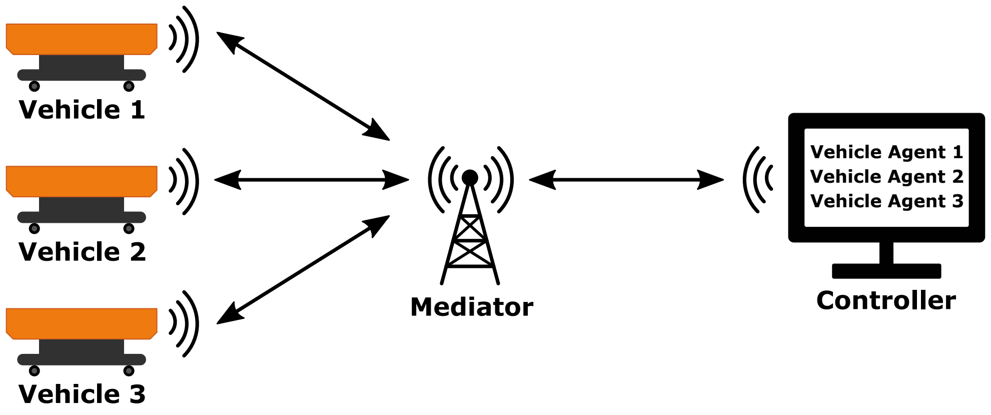


Figure 3.3: Visual representation of the difference between vehicles and vehicle agents, and the communication network between vehicles and the controller.

Layout The *layout* is a digital representation of the environment in which the vehicles find themselves.

Such a digital representation subdivides the environment into a grid of tiles, which are not allowed to overlap and where each tile is designed to hold only one vehicle at a time. Over this grid of tiles a network of nodes and directed edges, i.e. a graph, is placed. The nodes in this graph correspond with the center points of the tiles and the edges in this graph indicate that it is possible to reach one node from another node. Using a graph search algorithm, it is possible to plan paths between the tiles in the layout.

In the layout there exist three types of points of interest, or POI for short. The first type of POI is a pick-up station, which is a location where a vehicle can pick up a load. The starting point of a job is always a pick-up station. After picking up a load, a vehicle travels across the network of nodes and edges to the second type of POI, which is a drop-off station. At a drop-off station, vehicles can drop off a load. The endpoint of a job is always a drop-off station. The last type of POI is a parking spot. A vehicle is sent to one of these spots when it does not have a job to perform, in order to prevent the vehicle from blocking other vehicles from performing their jobs.

An example of a layout can be seen in Figure 3.4.

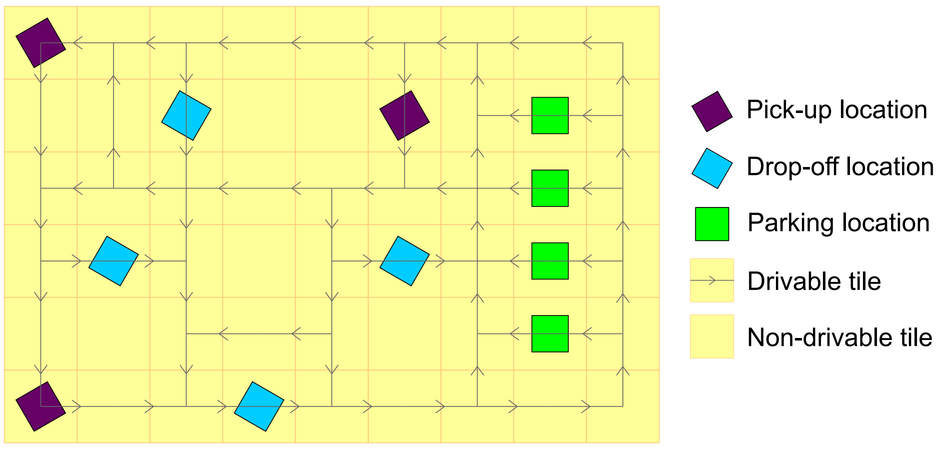


Figure 3.4: Visualization of an example layout.

A layout has a couple of requirements that it must satisfy in order to be a feasible layout:

1. It must be possible to reach every drivable tile in the layout from every other drivable tile in the layout in a finite time.
2. If a tile can be entered, it must also be possible to leave the tile.
3. The layout must have at least one pick-up station.
4. The layout must have at least one drop-off station.
5. Every POI must be located on a drivable tile, i.e. it must be possible to reach every point of interest.

3.1.4 Message types in grid-based control

There exist two types of messages in the grid-based control model, which are vehicle update messages and vehicle commands messages. These two types of messages are used to send information between the controller and the vehicles. Each of the message types is briefly introduced:

1. **Vehicle update message:** Vehicle update messages are periodically sent from the vehicles to the controller. A vehicle update message contains a vehicle update, which consists of the current coordinates, heading and speed of the vehicle, the state of the battery, whether a load is present, and whether an instruction (i.e. load, unload or parking action) was completed or not. Based on these vehicle updates, the controller is able to determine whether a vehicle has reached any nodes or left any tiles, whether it should receive a new target or path, and whether it should reserve any additional tiles for the vehicle.

2. **Vehicle commands message:** Vehicle commands messages are sent from the controller to the vehicles. A vehicle commands message contains vehicle commands, which are essentially the waypoints that have to be driven by a particular vehicle to progress to the next location in its path. Additionally, it includes whether an instruction (i.e. load, unload or parking action) has to be performed at any of these waypoints. Vehicles can process vehicle commands in order to apply the control action that was computed by the controller.

3.2 The supervisory control cycle

The supervisory controller has two types of events: performing periodic deadlock detection and performing its supervisory control cycle. Periodic deadlock detection is left out of scope for this thesis, for more information [Fransen et al., 2022] can be consulted. The control cycle of the supervisory controller is performed periodically and consists of 7 procedures, each of which is briefly introduced now.

1. Processing vehicle updates: The controller determines whether it has received any vehicle updates from any vehicle since the last controller cycle. For every vehicle, the controller evaluates whether any nodes have been reached or any tiles have been left. Additionally, the controller checks for every vehicle update whether a loading action, unloading action, or a parking action was completed. If any of these instructions were completed, this means that the target of the vehicle agent can be cleared such that a new target can be assigned later.

2. Processing left tiles: If any tiles have been left, the reservations of these tiles have to be released in the central tile reservation administration such that other vehicle agents can reserve these tiles in the future. Additionally, if any of these tiles is a parking spot, the corresponding parking reservation needs to be released as well so that other vehicles can use this spot in the future in order to park.

3. Processing reached nodes: If any nodes have been reached, these reached nodes have to be processed. Essentially, this boils down to updating the vehicle agent administration that keeps track of when a vehicle last reached a node. This information is used in path planning and when determining agent priority in traffic control.

4. Assigning new targets and paths: For every vehicle agent, the controller checks whether the agent has a job and a target. Depending on whether the agent has a job and a target, one of the following things happens:

- **Agent has a job but no target:** A new target and path towards this target is determined. If the vehicle is not yet loaded, the new target is the pick-up station of the agent's job. On the other hand, if the vehicle is already loaded, the new target is the drop-off station of the agent's job.
- **Agent has a job and a target:** It is likely that the vehicle is performing an ordinary job. However, it can also be that the vehicle was on the way to a parking spot when receiving a new job. If the vehicle's current target is not a parking spot, the vehicle is indeed performing an ordinary job. Instead, if the target is a parking spot, the parking spot reservation is canceled and a new path is determined towards the pick-up point of the job that the vehicle has just received.
- **Agent has no job and no target:** The vehicle is standing idle in the middle of the layout, waiting for its next job. In order to prevent the vehicle from blocking other vehicles from performing their jobs, a path is determined towards a parking spot if the layout has any parking spots. If the layout has no parking spots, if all parking spots are already occupied, or if the vehicle is already located at a parking spot, the controller does not do anything.
- **Agent has no job but has a target:** The vehicle is already on the way to a parking spot. The controller does not do anything.

More information regarding the exact path planning algorithm that is used can be found in [Fransen et al., 2020].

5. Periodic rerouting: The controller checks whether any of the vehicle agents is candidate for periodic rerouting. If the last time that an agent received a new path is sufficiently long ago, the controller assigns a new path to the agent if there exists a shorter path to the target than the path that the agent currently has. Congested areas in the environment can change over time, and by re-evaluating whether the current path is still the best path, periodic replanning aims to take into account congestion.

6. Providing new tiles: The controller determines for vehicle agents additional tiles need to be reserved. The controller iteratively attempts to reserve the next tile for the agent that is on top of the priority list. This tile can only be reserved for the agent if it passes the availability check and deadlock check, meaning that the tile may not be reserved by another agent (i.e. it is available) and that reserving this tile does not result in a deadlock. In [Fransen et al., 2022] the deadlock avoidance algorithm is explained in full detail.

7. Computing vehicle commands: New vehicle commands (i.e. waypoints) are computed by the controller just in time, such that the vehicle does not have to decelerate if possible. New waypoints are only computed for tiles that have already been reserved by the agent but whose waypoints have not yet been sent. If any new waypoints have been computed, these waypoints are sent to the correct vehicle through the mediator using a vehicle commands message.

3.3 Communication between objects

In Section 3.1.4 two types of messages were introduced: vehicle update messages and vehicle commands messages. In this section, the communication process behind each of these message types is elaborated upon in more detail.

3.3.1 Vehicle updates from vehicles to controller

The vehicle logic of each vehicle is responsible for periodically initiating the propagation of a vehicle update message from the vehicle to the mediator. In sending messages from the vehicle to the mediator, the vehicle communication of the vehicle acts as a conduit. The mediator forwards the vehicle update message to the controller. The communication process of sending a vehicle update message from a vehicle to the controller can be described using the following steps:

1. Vehicle logic triggers vehicle communication to send a vehicle update to the mediator containing the vehicle's most recent kinematic state. Vehicle communication forwards the vehicle update message to the mediator.
2. The mediator adds the incoming vehicle update message to its message queue. The time of execution of this message is equal to the current time unless a delay is applied to the message. We assume that no delay is applied to the message, implying that the timestamp to execute the message is the same as the current time.
3. If there are no other messages in the message queue or if all other messages in the message queue have a larger time to execute than the new message, the next event of the mediator is set to forward the incoming vehicle update message. If there are other messages in the message queue with a lower or identical time to execute, these messages are forwarded first. In this case, the next event of the mediator is not adjusted. This is explained in more detail in Section 3.1.2.
4. As soon as the time of execution of the vehicle update message has been reached in the simulation and the message is in front of the message queue, the mediator forwards the vehicle update message to the controller.
5. The controller receives the incoming message. It adds the incoming vehicle update message to the list of incoming vehicle updates that still have to be processed by the controller.
6. In the next cycle of the controller, all vehicle updates that are in the list of vehicle updates that still need to be processed are processed by the controller.

The sequence diagram corresponding to the communication process from the vehicles to the controller in the grid-based control model can be found in Figure A.1.

3.3.2 Vehicle commands from controller to vehicles

During the performance of its job, a vehicle follows waypoints that are provided by the controller. These waypoints are sent just in time, such that a vehicle does not have to decelerate if possible. As a result of this, the controller continuously has to check whether each vehicle requires new waypoints. If a vehicle requires new waypoints, the controller sends these waypoints to the mediator using a vehicle commands message, which forwards the message to the correct vehicle. The communication process of sending a vehicle commands message from the controller to a vehicle can be described using the following steps:

1. The controller performs its control cycle and computes new waypoints for a particular vehicle. These waypoints are sent from the controller to the mediator using a vehicle commands message.
2. The mediator adds the incoming vehicle commands message to its message queue. The time of execution of this message is equal to the current time unless a delay is applied to the message. We assume that no delay is applied to the message, implying that the timestamp to execute the message is the same as the current time.
3. If there are no other messages in the message queue or if all other messages in the message queue have a larger time to execute than the new message, the next event of the mediator is set to forward the incoming vehicle commands message. If there are other messages in the message queue with a lower or identical time to execute, these messages are forwarded first. In this case, the next event of the mediator is not adjusted. This is explained in more detail in Section 3.1.2.
4. As soon as the time of execution of the vehicle commands message has been reached and the message is at the front of the message queue, the mediator forwards the message to the vehicle communication of the correct vehicle.
5. Vehicle communication receives the incoming message and triggers vehicle logic to receive the incoming vehicle commands message. Vehicle logic immediately processes the new waypoints and appends them to the list of waypoints that it has received before.

The sequence diagram corresponding to the communication process from the controller to the vehicles in the grid-based control model can be found in Figure A.2.

4 Hierarchical control design

In Chapter 2 a variety of control strategies were studied, based on which the decision was made to develop a new hierarchical control strategy. Although this control strategy is largely inspired by the work of [Liu et al., 2019], a lot of new components and procedures still have to be designed.

In this chapter, a distinction between global and local properties and objects is made first. Then, all new components in the hierarchical control strategy are clarified in more detail, as well as all new procedures that are required so that vehicles can successfully perform their jobs. Lastly, multi-sector deadlock is introduced.

4.1 Hierarchical control definitions

As was introduced in Section 2.3, the model is distinguished into two layers of control, which are global path planning and local traffic control. With that in mind, a distinction between global and local properties and objects is made. The most important definitions are introduced in this section.

First of all, a digital representation of the environment is made in the form of a layout. This layout is partitioned into sectors, which are defined according to Definition 4.1. There must be at least two sectors in the layout in order to use the hierarchical control framework.

Definition 4.1 (Sector)**.** A *sector* is a portion of the full layout that has its own local controller in order to control the vehicles in said sector.

There is one centralized global controller, which has access to a global layout. The global controller and global layout are defined in Definition 4.2 and 4.3, respectively.

Definition 4.2 (Global controller)**.** The *global controller* is a centralized controller that plans global paths for all vehicles and forwards vehicle updates to the correct local controller. The global controller is unaware of the existence of any of the vehicles and vehicle agents.

Definition 4.3 (Global layout)**.** The *global layout* is an approximation of the full layout, which is used to determine global paths. The global layout aims at creating a general impression of the full environment while disregarding most details.

Using the global layout, the global controller determines global paths to global targets using Dijkstra's algorithm when it is requested to do so by a local controller. The global path and global target are defined as per Definition 4.4 and 4.5, respectively.

Definition 4.4 (Global path)**.** A *global path* is an approximation of the shortest path that a vehicle can take to traverse from one point of interest to another point of interest.

Definition 4.5 (Global target)**.** A *global target* is the final target of a global path, i.e. the final target that a vehicle must travel to. A global target is always a pick-up station or a drop-off station, depending on whether the vehicle is currently loaded or unloaded.

In the future, global targets may also be parking spots. However, for now this functionality has not yet been implemented.

Each sector has its own decentralized local controller, which is responsible for controlling vehicles in one particular sector. Each local controller has access to a local layout, which corresponds to the environment of the sector that is controlled by this particular controller. The local controller and local layout are defined according to Definitions 4.6 and 4.7, respectively.

Definition 4.6 (Local controller)**.** A *local controller* is a decentralized controller that manages the local path planning and local traffic control in one particular sector, for the vehicles that are currently located in said sector. Each local controller is only aware of the existence of the vehicle agents that are located in its own sector.

Definition 4.7 (Local layout)**.** A *local layout* is a digital representation of the environment in one particular sector. This representation separates the environment into a grid of tiles, over which a network of nodes and edges, i.e. a graph, is placed. The nodes in this graph correspond with the center points of the tiles and the edges in this graph indicate that it is possible to reach one node from another node.

Each local controller plans local paths for the vehicles that are currently located in its own sector using only the tiles in its own sector. A local path is always planned to a local target. The local path and local target are defined in Definition 4.8 and 4.9, respectively.

Definition 4.8 (Local path)**.** A *local path* contains the exact path a vehicle must take within the current sector in order to reach the local target of this sector. This local path is determined by the local controller and can only contain tiles *in its own sector.*

Definition 4.9 (Local target)**.** The *local target* is the final target of a local path, i.e. the path in a particular sector. If the next tile in the global path is the global target, the next local target is identical to the global target.

In Section 4.3.2 an example of all these new definitions is given.

4.2 Hierarchical component design

Following the definitions that have been introduced in Section 4.1, there is a lot of room for interpretation. In this section, several design decisions that have been made regarding the local layout, local controller, global layout and global controller are elaborated upon in some more detail.

4.2.1 Local controller

As per Definition 4.6, a local controller is defined as "a decentralized controller that manages the local path planning and local traffic control in one particular sector". Therefore, each local controller has full information on the local layout of the sector to which it corresponds and of the vehicle agents that are currently located in this sector. Because of this, each sector can be controlled as if it were a grid-based system. However, a local controller has no information about any other controllers, layouts, vehicles or vehicle agents outside its own sector, not even the global controller.

As was already briefly mentioned in Section 2.3, the desired properties of the controller in a particular sector depend on the characteristics of that sector. For example, the desired properties of a controller in a storage area of an airport might be different in comparison to a transportation area between terminals. Furthermore, if a particular sector in the layout is designed in such a way that deadlock simply *cannot* occur, then a deadlock avoidance and detection algorithm is redundant for the local controller corresponding to this sector, which reduces the computational complexity. An example of this can be seen in Figure 4.1, where deadlock is not possible inside sector B due to the design of the local layout[1]. For this reason, the properties of a local controller are allowed to vary per sector, as long as vehicles do not collide, do not end up in deadlock, and are able to correctly travel from sector to sector in order to perform their jobs.

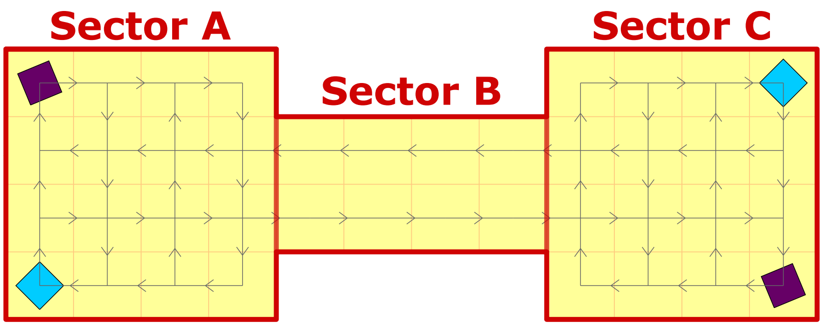


Figure 4.1: Layout illustrating different desired properties in certain sectors.

[1]Deadlock involving sector A, B and C can still occur. This form of deadlock is explained in more detail in Section 4.6.

Although it is allowed to have a different type of local controller in each sector, in the remainder of this thesis it is assumed that all local controllers are the same. Moreover, it is assumed that each local controller is a modified version of the supervisory controller that was introduced in Section 3.1.2, meaning that *within each sector* the path planning algorithm described in [Fransen et al., 2020] and the deadlock avoidance and detection algorithm described in [Fransen et al., 2022] are used. Each of these local controllers has its own local control cycle that is performed periodically, which contains some additional procedures with regard to the regular supervisory control cycle introduced in Section 3.2, in order to facilitate vehicle transitions from one sector to another. The local control cycle is explained in full detail in Section 4.5.

4.2.2 Local layout

Each local controller has access to a local layout which is "a digital representation of the environment" of the sector that is controlled by this particular local controller. Moreover, this digital representation "separates the environment into a grid of tiles, over which a network of nodes and edges, i.e. a graph, is placed". In essence, a local layout is similar to the layout object introduced in Section 3.1.3, except for the fact that a mechanism has to be implemented in order to facilitate transitions between sectors. Two of such mechanisms are introduced shortly.

Entrance and exit tiles

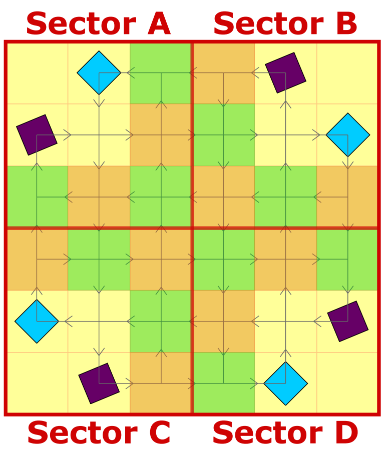
One way to facilitate transitioning between sectors is to use entrance and exit tiles. Here, entrance and exit tiles are defined according to Definition 4.10 and 4.11, respectively.

Definition 4.10 (Entrance tile)**.** An *entrance tile* is a tile through which a sector can be entered from another sector. An entrance tile is connected to at least one exit tile that is located in another sector.

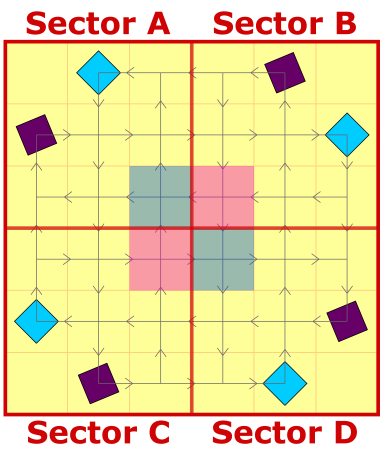
Definition 4.11 (Exit tile)**.** An *exit tile* is a tile through which a sector can be left towards another sector. An exit tile is connected to at least one entrance tile that is located in another sector.

In the layout in Figure 4.2a, the entrance tiles are shaded in green and the exit tiles are shaded in orange. Because an entrance tile is always connected to at least one exit tile in another sector and an exit tile is always connected to at least one entrance tile in another sector, we can always talk in terms of entrance-exit pairs. Moreover, a tile can be part of more than one pair. This is the case when an entrance tile can be reached from multiple exit tiles or when an exit tile can be left towards multiple entrance tiles. In Figure 4.2b such entrance and exit tiles are shaded in blue and pink, respectively. Finally, a tile can serve as both an entrance and exit tile at the same time, which means that it can be entered from an exit tile in another sector and left towards an entrance tile in another sector.

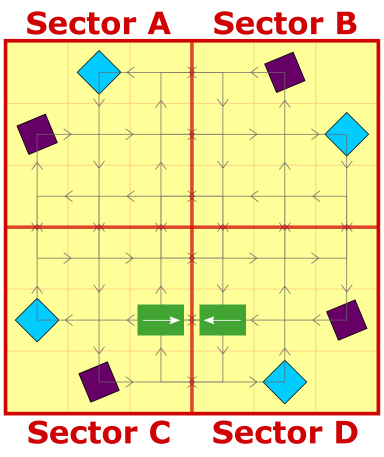
However, a tile can never serve as both an entrance and an exit tile to the same sector, meaning that a connection between an entrance-exit pair can only be unidirectional. As an example, the edges crossing the borders between all sectors in Figure 4.2c are bidirectional. As can be seen, this can lead to a situation where two vehicles in different sectors both want to use the other vehicle's tile in order to transition to the other sector. However, the local controllers of sectors A and B have no information on the tile reservations in the other sector. Essentially, this is a form of deadlock that can only be solved by making both controllers work together in order to replan the path of (at least) one of the vehicles. Although this is a possibility, it significantly increases the probability that a deadlock between multiple sectors occurs. Moreover, the layouts that are considered usually already contain only unidirectional edges because it significantly complicates the traffic control problem if bidirectional edges are used, for relatively little gain.



(a) Entrance and exit tiles are highlighted.



(b) Tiles that are part of more than one pair are highlighted.



(c) Deadlock situation at sector border has been visualized.

Figure 4.2: Layout with four sectors, illustrating the definition of entrance and exit tiles and a deadlock problem that arises when bidirectional pairs of entrance and exit tiles are used.

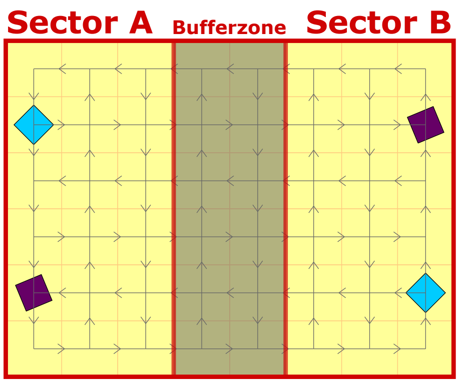
If the notion of entrance and exit tiles is used, a local controller can easily plan a local path for a vehicle to one of the exit tiles that leads to the next sector. A special communication mechanism must be implemented to transition the vehicle to the next sector and the corresponding vehicle agent to the next local controller. As soon as the entrance tile of the next sector has been reached, the local controller of said sector can take over the control.

Buffer zone

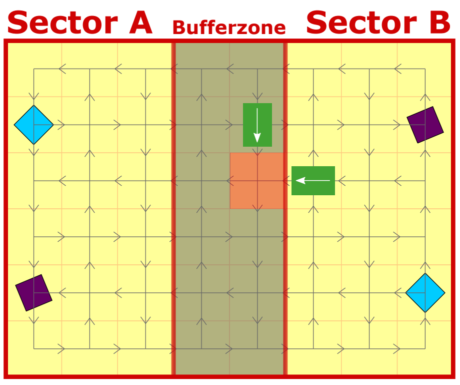
Another way to facilitate transitioning between sectors makes use of a buffer zone between two sectors. The layout in Figure 4.3a contains two sectors and a buffer zone.

Although each sector has its own local controller, both local controllers have full information on the tiles and vehicle agents of the vehicles that are located inside the buffer zone. Because both controllers have full information on the reservation status of the tiles that are located inside the buffer zone, the administration of both controllers needs to be updated whenever a vehicle enters or moves inside the buffer zone. The usage of a buffer zone raises some important questions, like *which controller is responsible for controlling a vehicle that is located in the buffer zone?*

Imagine the situation in Figure 4.3b, where both controllers have full information on the tile shaded in red and each of them can try to reserve this tile if it is available. Ideally, controller B is responsible for controlling both vehicles because then it has full information on both agents and tiles. However, consider the situation where the vehicle in the buffer zone originates from sector A and is still controlled by controller A. If controller A reserves the tile for its vehicle. The tile reservation administration of local controller B also needs to be updated so that the controller is aware that it cannot reserve this tile for its own vehicle. However, in practice there is likely to be a certain delay in communication (though the delay might be extremely small). As a result of this, the administration of local controller B might not yet be updated by the time the controller attempts to reserve the red-shaded tile for its own vehicle. This leads to a situation where both controllers have reserved this tile for their vehicles, which can clearly lead to problems.



(a) Two sectors connected through a buffer zone.



(b) Two vehicles require the same tile in the buffer zone.

Figure 4.3: Layout with two sectors, illustrating the definition of a buffer zone and a situation where two vehicles require the same tile in the buffer zone.

Another question that arises is *when is a vehicle transferred to another controller?* One could say, for example, that a vehicle is transferred to another controller as soon as it has entered the buffer zone, or as soon as the next tile that must be reserved is located in the sector of the next controller. Regardless of the decision, a mechanism must be implemented that verifies whether the tile that a vehicle is located on satisfies the condition for transferring the vehicle to the next controller. The method using entrance and exit tiles contains a mechanism exactly like this, but leaves out the difficulties that come with using a buffer zone. Resulting from this analysis, the decision has been made to make use of pairs of entrance and exit tiles in order to transition from one sector to another.

In Section 3.1.3 it was already introduced that it must be possible to reach every drivable tile in the layout from every other drivable tile in the layout in a finite time. Because of this, it is always possible for a vehicle to reach all other drivable tiles in the entire layout regardless of the sector partitioning, meaning that a vehicle can never get stuck in a sector. Besides the constraints on the full layout that were introduced in Section 3.1.3, a local layout is not subject to any additional constraints.

4.2.3 Global layout

Similar to how local controllers have a local layout, which is a digital representation of the environment of *their own sector*, the global controller has a global layout, which is *an approximation of the full layout.* Based on this global layout, the global controller makes an approximation of the shortest path a vehicle can take towards a pick-up or drop-off station, depending on whether the vehicle is loaded or unloaded.

Although Definition 4.3 states that the global layout "aims at creating a general impression of the full environment, while disregarding most details", the question that still remains is *what exactly does the global layout look like*? In an attempt to keep the global layout as simple as possible, an initial definition of the global layout is as follows:

Global layout In the *global layout*, every sector is represented by its own node, whose coordinates are a weighted average of all tiles located in the sector. If it is possible to travel from sector X to Y, the nodes representing these sectors are connected by a directed edge from X to Y in the global layout.

Global path infeasibility

As a result of this definition, for each layout a global graph can be defined. The global path containing the sequence of sectors from the sector of the pick-up station to the sector of the drop-off station can then be determined using a graph search algorithm. Let us take a look at what happens when attempting to plan a global path from the pick-up station on tile 6 to the drop-off station on tile 8 in Figure 4.4.

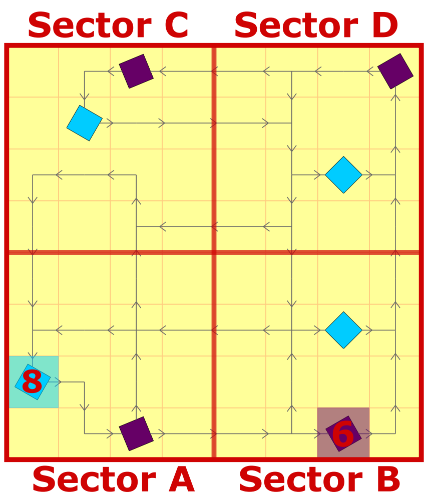
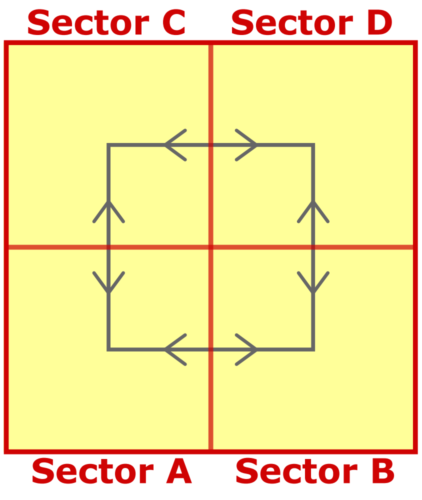
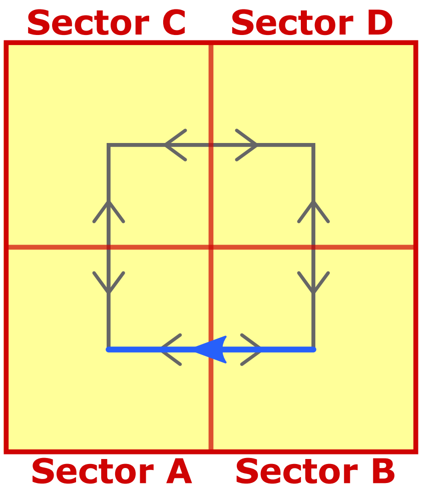


Figure 4.4: Example layout to illustrate global path infeasibility.

In Figure 4.5a the global layout corresponding to the layout in Figure 4.4 can be seen, which consists of four nodes and 4 bidirectional edges between all pairs of neighboring sectors. Because the pick-up station is located in sector B, the drop-off station is located in sector A, and there is an edge leading from sector B into A, the global path in this example is defined as B-A, which has been visualized in Figure 4.5b.



(a) Example of a global layout.



(b) Example of a global path.

Figure 4.5: Example of a global layout and a global path.

Upon closer inspection, it can be seen that the global path that is derived using this definition cannot actually be executed. To be more specific: there is no path, starting from the pick-up up station, that directly leads to sector A for the layout in Figure 4.4. The only way in which sector A can be reached, starting from the pick-up station, is via sector D and possibly sector C, as is indicated in Figure 4.6. In conclusion, using the current definition of the global layout can lead to problems where the global path is infeasible.

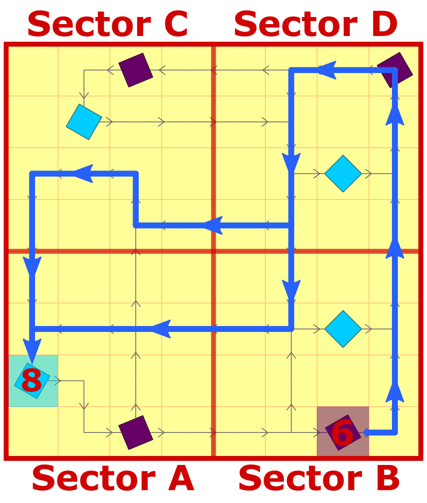


Figure 4.6: Example of feasible paths to reach the drop-off station in sector A starting from the pick-up station in sector B.

In order to ensure that the global path is always feasible using the current definition of the global layout, within every sector it must be possible to travel from every drivable tile to every exit tile and POI in the sector. Note that this is not guaranteed by the constraint introduced in Section 3.1.3, which states that it must be possible to reach every drivable tile in the layout from every other drivable tile in the layout in a finite time. An option to guarantee that the global path is always feasible is to redesign the layout. With this in mind, two things (or a combination of the two) can be done: adding edges or adjusting the location of sector borders.

In Figure 4.7 the layout has been redesigned by adding edges to the layout. The corresponding global layout still looks like the global layout shown in Figure 4.5a and as can be seen, the infeasibility problem has been resolved. However, in reality the environment on which the layout is based might not have space for additional edges or the layout might be tailored to specific requirements. For those reasons, it is undesirable to adjust the original layout simply in order to suit our needs.

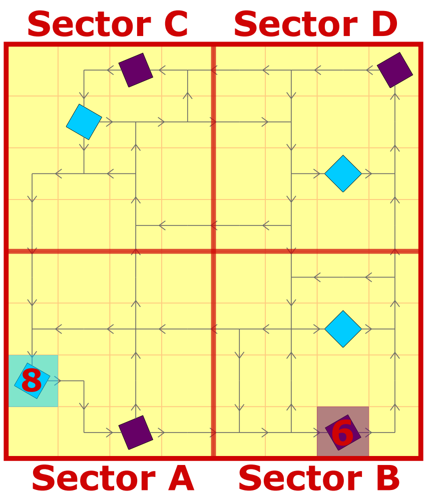
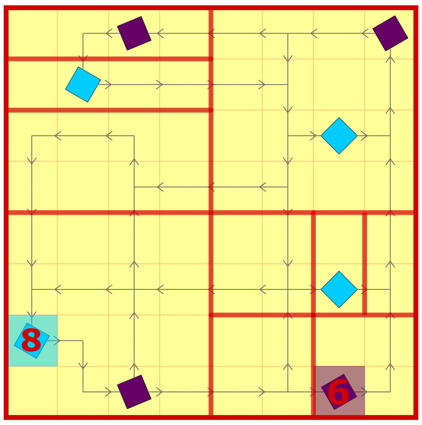


Figure 4.7: Edges have been added to the layout shown in Figure 4.4 to ensure the global path is always feasible.

Adjusting the sector borders, on the other hand, is still a possibility because a sector is essentially an artificial concept[1]. In Figure 4.8a an example of how the sector borders can be adjusted in order to guarantee global path feasibility can be found. However, this leads to a layout with relatively many sectors. In Figure 4.8b an example of a feasible global path from the starting sector to the ending sector can be seen.



(a) The new sector division.

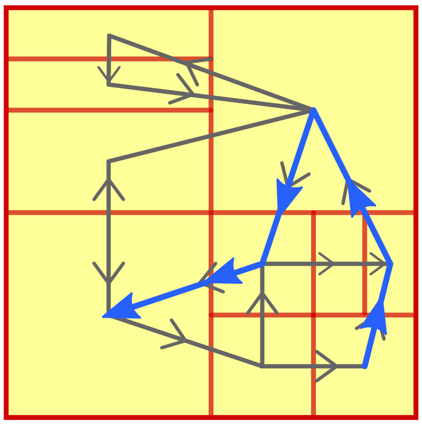
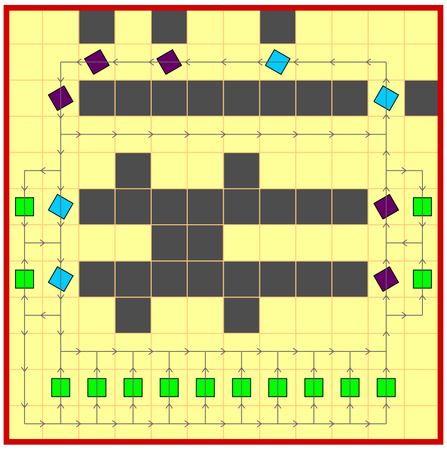

(b) The new global layout and path.

Figure 4.8: Sector borders of the layout shown in Figure 4.4 have been adjusted to ensure the global path is always feasible.

In Figure 4.9a a layout representing Terminal 2 of Marseille Provence Airport has been shown. If an attempt is made to partition this layout into more than one sector where global path feasibility is guaranteed, the result looks something like Figure 4.9b, although there are alternatives. What can be illustrated by this example is that for certain layouts, guaranteeing global path feasibility by adjusting sector borders results in a partitioned layout with so many sectors that it defeats the purpose of aiming to create a general impression of the full environment, while disregarding most details.


(a) Layout before sector partitioning.

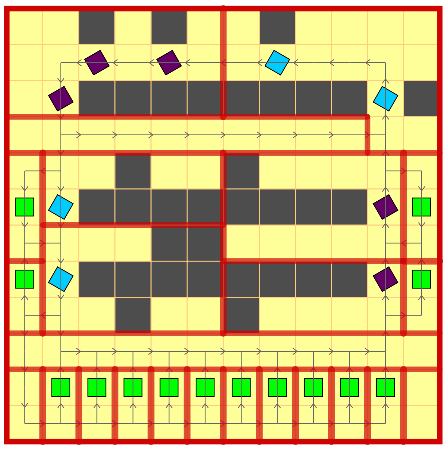

(b) Layout after sector partitioning.

Figure 4.9: Layout representing Terminal 2 of Marseille Provence Airport.

In summary, using the current definition of the global layout and designing the layout such that the global path is always feasible leads to undesirable results. Keeping in mind the problems that arose so far, a new definition of the global layout needs to be devised such that the global path is always feasible. The initial definition of the global layout was too simple and naive, so there is no choice but to add a bit more information to the global layout in favor of always having a feasible global path.

[1]In practice the definition of the sector borders can still depend on the desired controller properties in different areas of the environment.

The new definition of the global layout is as follows:

Global layout For every drivable tile in each sector, it is evaluated which exit tiles and POIs in the same sector can be reached. A directed edge is created from said tile to the exit tile or POI if it is possible to reach the exit tile or POI by *only using tiles in the same sector*. Additionally, directed edges are created from each exit tile to each corresponding entrance tile.

Using this definition, if the start tile and the destination tile (i.e. a pick-up or a drop-off station) are located in the same sector and the destination tile can be reached by only using tiles in this sector, the global path is defined in terms of a start tile and a destination tile. If not, the global path is defined in terms of a start tile, one or more entrance-exit pairs, and a destination tile. This definition also takes into account the fact that the initial position of a vehicle at the start of a simulation can be any drivable tile and that it might be desirable to periodically replan global paths in a future implementation.

To illustrate the implications of this new definition, the layout in Figure 4.10 is examined.

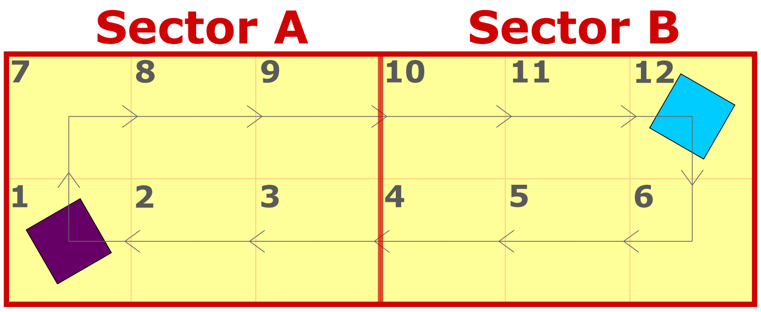


Figure 4.10: Visualization of a simple layout.

As an example, we first evaluate the tiles in sector A. It is possible to reach the exit tile, i.e. tile 9, from tile 1, 2, 3, 7 and 8 using only tiles in sector A, so in the global layout there are directed edges from all of these tiles to tile 9. Additionally, from tiles 2 and 3 it is possible to reach the pick-up station, i.e. tile 1, using only tiles in sector A. Therefore, there are directed edges from tiles 2 and 3 to tile 1 in the global layout. The same procedure can be followed for sector B. Lastly, all exit tiles are connected to the corresponding entrance tiles in the global layout using a directed edge, meaning that directed edges are created from tile 4 to tile 3 and from tile 9 to tile 10. The global layout resulting from this procedure can be seen in Figure 4.11.

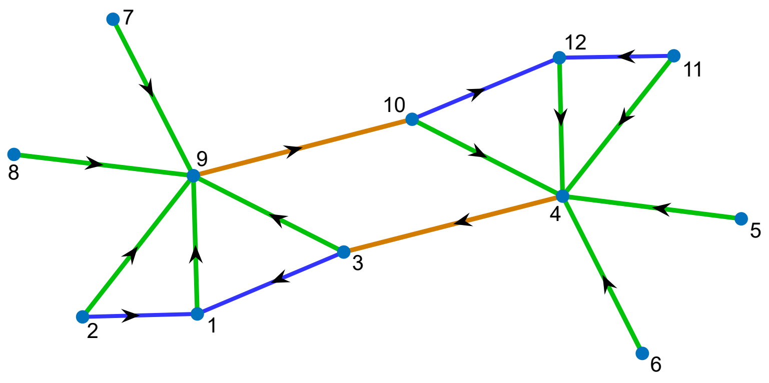


Figure 4.11: Global layout corresponding to the layout in Figure 4.10.

Imagine that a global path must be planned from tile 2 to tile 12. In Figure 4.11 it can be seen that there exist two paths from tile 2 to 12, namely 2-9-10-12 and 2-1-9-10-12. Assuming that all edges in the global layout have equal weight and that Dijkstra's algorithm is used, the shortest path is always the path that goes directly from 2 to 9 instead of via 1. One might argue that the global path is no longer an *approximation* if the path is already fixed for such a large part. However, the global path does not contain any information on how tile 9 is reached from tile 2, nor how tile 12 is reached from tile 10. The local controllers can still determine themselves how these local paths should look. In this particular example, the local controllers do not have a lot of choice, but this is purely due to the nature of this particular layout. In general, though, the local controllers can fully decide the local paths.

Another fair remark is that the size of the global layout also grows significantly large for layouts with a lot of drivable tiles, exit tiles or POIs. As an example, the layout shown in Figure 4.12 contains 656 edges, whereas its corresponding global layout has 1006 edges. However, graph search algorithms such as Dijkstra's algorithm, which is used to determine the global path, are known to be very fast and neighboring nodes are only evaluated if there is a directed edge going *into* them. Even though node 9 in Figure 4.11 is connected to six other nodes, only node 10 is evaluated when determining the distance to neighboring nodes starting from node 9, because nodes 1, 2, 3, 7 and 8 are connected through a directed edge going *into node 9* and not the other way around. Additionally, the original grid-based control model mainly does not scale well as a result of deadlock avoidance and detection, and *not* because of the graph search algorithm.

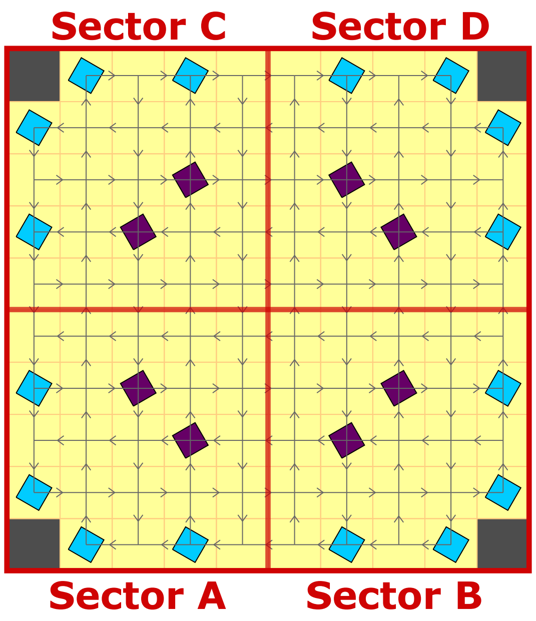


Figure 4.12: Visualization of a larger layout.

Global layout edge weights

So far it was assumed that the weights of all edges in the global layout in Figure 4.11 have equal weight. If this assumption holds and the global path is computed from tile 2 to 12, the graph search algorithm always finds the direct path, which is preferred. However, this assumption can also lead to undesired results.

An alternative layout and the corresponding global layout have been visualized in Figure 4.13 and 4.14, respectively. If the global path is computed from tile 2 to 12, the graph search algorithm finds two equally large paths: path 2-9-10-12 and path 2-17-18-12, which both have a weight of three. However, the global path using tile 2-9-10-12 is clearly shorter than the global path using tile 2-17-18-12. To resolve this problem, edge weights are assigned to the edges in the global layout based on the distance between the two tiles that are connected by the edge. The distance between the two tiles is merely used to estimate how long it will take a vehicle to go from one tile to another. As was already mentioned in Section 2.2.1, the shortest path distance-wise does by no means have to be the shortest path time-wise.

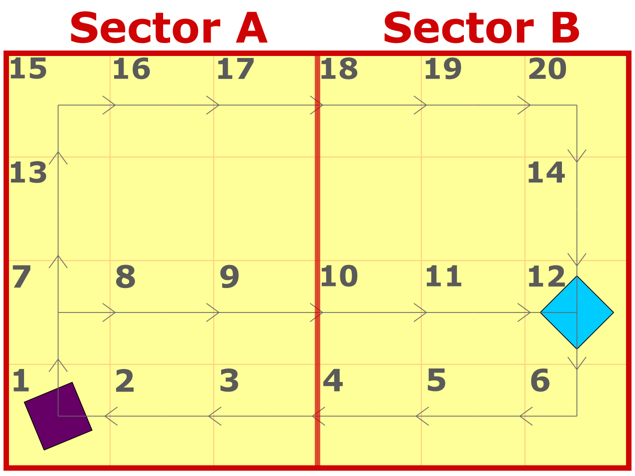


Figure 4.13: Visualization of a simple layout with two entrance-exit pairs to get from sector A to B.

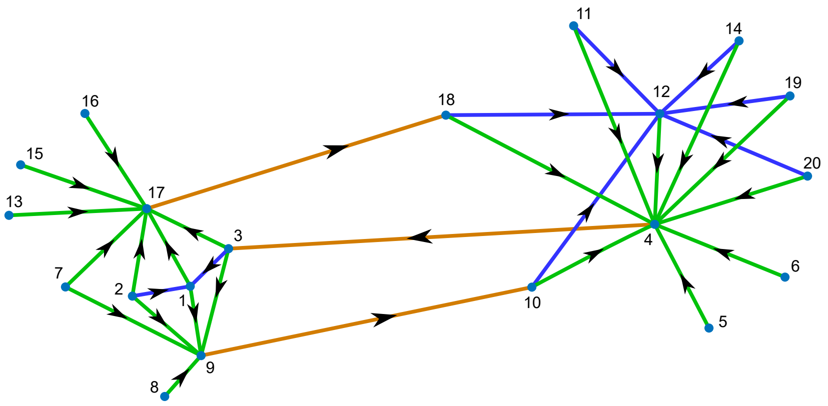


Figure 4.14: Global layout corresponding to the layout in Figure 4.13.

4.2.4 Global controller

There is only one global controller, which has its own global control cycle. This global control cycle is performed periodically and is explained in full detail in Section 4.4. Although the global controller has its own global control cycle, it is unaware of the existence of the vehicles and vehicle agents and it does not initiate any actions on its own. The global controller only forwards vehicle updates when it has received vehicle updates from a vehicle itself, and only determines a global path if one has been requested by one of the local controllers. More on this is explained in Sections 4.3.3 and 4.3.5, respectively. Furthermore, the global controller is not involved with the local traffic control of any of the vehicles.

4.3 Hierarchical control procedures

So far, only the design decisions regarding the local layout, local controller, global layout and global controller have been explained in more detail. In this section, the design decisions regarding the procedures that are required so that vehicles can successfully perform their jobs are clarified.

4.3.1 Message types in hierarchical control

In Section 3.1.4 the two message types that exist in the grid-based control model were introduced, which are vehicle update messages and vehicle commands messages. However, within the hierarchical control strategy, these two types of messages are not sufficient. Therefore, seven additional types of messages are introduced.

1. **Vehicle update delayed message:** Vehicle update delayed messages are sent from the global controller to the local controllers. They contain the same information as vehicle update messages, but arrive at their destination controller with a certain delay, which is caused by the fact that these messages are sent to the destination via the global controller.
2. **Global path request message:** A global path request message is sent from a local controller to the global controller. The local controller sends this message when one of its vehicle agents requires a new global path to the global target.
3. **Global path response message:** A global path response message is sent from the global controller to a local controller in a response to a global path request message that was recently sent by this local controller. A global path response message contains the global path from a vehicle's last reserved tile to its global target.
4. **Sector transition request message:** A sector transition request message is sent from one local controller to another local controller. A local controller sends a sector transition request message to another controller when one of its vehicles needs to transition to the other sector.
5. **Sector transition approval message:** A sector transition approval message is sent from one local controller to another local controller in response to a sector transition request message that was sent recently. A controller only sends a sector transition approval message if it has approved a sector transition request and has reserved the entrance tile that was requested by this sector transition request.
6. **Vehicle registration message:** A vehicle registration message is sent from one local controller to another local controller. A vehicle registration message is sent when a vehicle has physically transitioned from one sector to another so that the controller of the receiving sector is able to register the new agent.
7. **Controller update message:** A controller update message is sent from a local controller to the global controller. This message is sent when a vehicle has physically transitioned from one sector to another to ensure that the global control administration is kept up-to-date.

The mediator always acts as the communication network when one of these messages is sent.

4.3.2 Planning paths using global targets and local targets

Using the definitions of global targets and paths and local targets and paths as defined in Section 4.1, the design of the global and local layouts in Section 4.2, and the message types introduced in Section 4.3.1, the procedure of planning paths using the hierarchical control strategy is now clarified. This is done for the scenario that was already introduced in Figure 4.4, where the pick-up station of the job is located on tile 6 in sector B and the drop-off station is located on tile 8 of sector A.

Assuming that the vehicle is loaded and does not have a global path or target, local controller B sends a global path request message to the global controller. In the next global cycle, the global controller determines the global path using Dijkstra's algorithm and responds with a global path response message. The global path is defined in terms of a start tile, i.e. the pick-up station, multiple entrance-exit pairs, and a destination tile, i.e. the drop-off station. As has been visualized in Figure 4.15, the global path is equal to 6-24-30-29-23-17-16-8.

From this global path, it can be derived that the *global target* is tile 8, whereas all other tiles are 'regular' tiles in the global path. However, these regular tiles alternately serve as the *local target* of the local path. To clarify this, the four separate local paths have been shown in Figure 4.16.

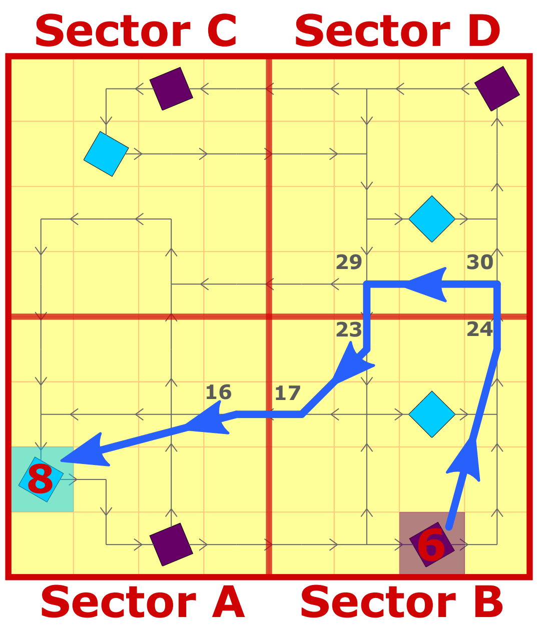


Figure 4.15: Global path from the pick-up station on tile 6 to the drop-off station on tile 8.

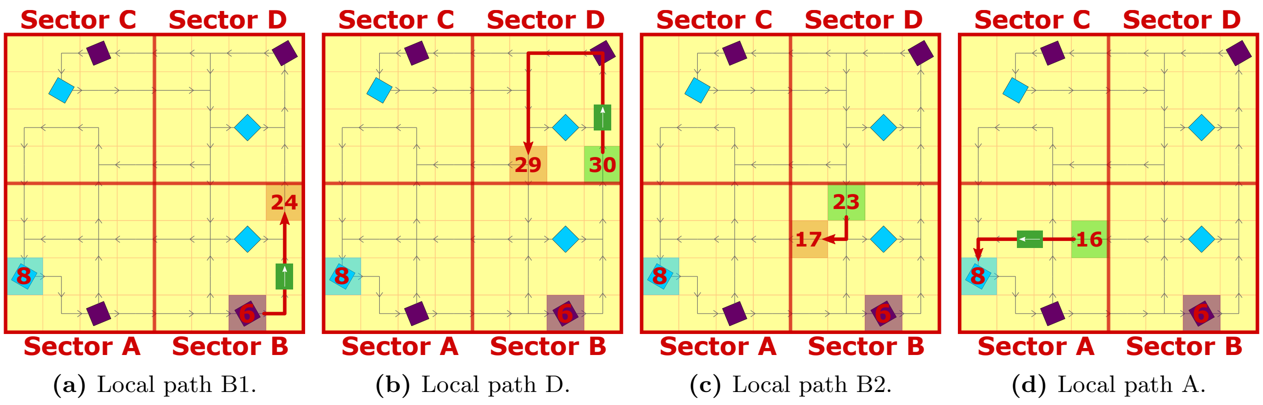


Figure 4.16: Local paths that are computed in each individual sector.

First of all, local controller B plans a local path from the pick-up station on tile 6 to exit tile 24 using the path planning algorithm introduced in [Fransen et al., 2020]. This local path can be seen in Figure 4.16a. As soon as the vehicle has reached tile 24, it must transition to entrance tile 30 in sector D. Therefore, local controller B sends a sector transition request message to local controller D. If entrance tile 30 is available, local controller D responds with a sector transition approval message and the vehicle can transition the sector border. Whenever the vehicle has finished transitioning from sector B to D, the vehicle agent is transferred from local controller B to local controller D using a vehicle registration message and the global controller administration is updated following a controller update message to ensure the administration is correct. How all of this communication takes place exactly is explained in full detail in Section 4.3.6.

As soon as the vehicle has arrived at entrance tile 30 and has been transferred to local controller D, this controller determines the local path from tile 30 to exit tile 29 as can be seen in Figure 4.16b. The vehicle transitions from exit tile 29 to entrance tile 23 in sector B and the agent is transferred back to controller B. Next, controller B determines the local path from entrance tile 23 to exit tile 17, which has been shown in Figure 4.16c. As soon as the vehicle has reached the exit tile, the vehicle transitions to the entrance tile in sector A and the agent is transferred to local controller A. Lastly, controller A determines the local path from entrance tile 16 to the drop-off station, which is located on tile 8 as can be seen in Figure 4.16d. Here, the vehicle unloads and a new global path is planned towards the pick-up station of the vehicle's next job.

4.3.3 Vehicle updates from vehicles to local controllers

In Section 3.3.1, vehicle update messages were already introduced in the context of the grid-based control model. In this model, there was only one controller to which vehicle update messages could be sent. Therefore, vehicles can simply send along 'the controller' as the destination of the message. In contrast, the hierarchical control model contains multiple local controllers to which vehicle update messages can possibly be sent. Crucially, vehicles and vehicle agents do not know in which sector they are located, and neither does the mediator. If vehicles send along 'the controller' as the destination of the message, the mediator does not know which of the local controllers is meant. Moreover, vehicles cannot specify which controller they mean, because they do not know in which sector they are located.

Vehicle update messages should at least arrive at the controller of the sector in which a vehicle is currently located. One way to resolve this problem is to send all vehicle updates to all local controllers. Every local controller can then check whether the vehicle is indeed located in its sector, and if so it can process the vehicle update. However, this unnecessarily stresses the communication network, which is something that is preferably kept to a minimum. Ideally, the local controllers are not bothered with vehicle update messages they do not need.

Instead, to resolve this problem, vehicles send a long 'the global controller' as the destination of their vehicle update messages in the hierarchical control model, which are forwarded by the mediator. The global controller keeps records of which vehicle is located in which sector. Using this administration, the global controller can look up in which sector each of the vehicles is located. In every global control cycle the global controller evaluates for all vehicle update messages that it has received to which local controller the message must be forwarded. For each of these vehicle update messages, the global controller sends a vehicle update delayed message to the mediator, which forwards this message to the correct local controller. This local controller can then process the vehicle update. The global controller essentially forwards the vehicle update to the mediator on behalf of the vehicle that originally sent the vehicle update. Whenever a vehicle transitions to another sector, the global controller is informed by the local controller of the sector which is left and such that it can update its administration.

The communication network in the hierarchical control model and the process of sending a vehicle update from a vehicle, via the global controller, to the correct local controller, has been illustrated in Figure 4.17. The communication process of sending vehicle updates from the vehicles to the local controllers in the hierarchical control model and the corresponding sequence diagram can be found in Appendix B.1.

4.3.4 Vehicle commands from local controllers to vehicles

In Section 3.3.2, vehicle commands messages were already introduced in the context of the grid-based control model. In the hierarchical control strategy, vehicle commands messages are sent in the exact same way. The only difference is that every local controller only computes vehicle commands (i.e. waypoints) for the vehicles that are currently *in its own sector*. In contrast to vehicle update messages, vehicle commands messages can be sent to the correct vehicle directly by the mediator, instead of via the global controller. The reason for this is that the local controllers *do* know which vehicle agents are located in their own sector and which vehicles correspond to these agents, whereas the vehicles and vehicle agents do not know in which sector they are located.

In Appendix B.2 the communication process of sending vehicle commands from the local controllers to the vehicles in the hierarchical control model and the corresponding sequence diagram can be found.

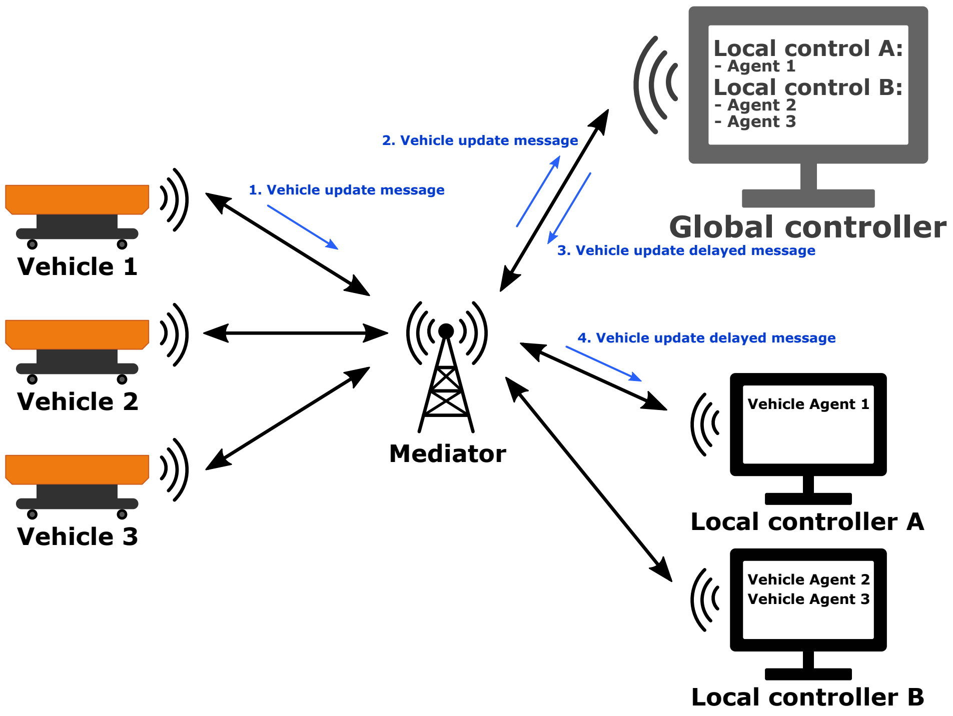


Figure 4.17: Visual representation of the communication network in the hierarchical control strategy and the process of sending a vehicle update to the correct local controller.

4.3.5 Assigning global paths to vehicle agents

In every local control cycle, each local controller checks for all its vehicle agents whether each agent has a job but no global target. If an agent indeed has a job but no global target, the agent needs to be assigned a global target and a global path to this target. Such a global path cannot be determined by the local controller, because the local controller only has information on the local layout of its own sector. Instead, global paths are determined by the global controller.

To facilitate this, a local controller sends a global path request message, via the mediator, to the global controller. If a global path request message has already been sent, no new request is sent to the global controller to reduce unnecessary communication between controllers while the local controller is waiting for a response. If not, the local controller places a global path request message in the mediator's message queue, which is forwarded by the mediator to the global controller when the mediator executes its event. The global target of the requested path is the drop-off or pick-up station of the vehicle's job, depending on whether the vehicle is loaded or not.

In every global control cycle, the global controller processes all global path requests that it has received. Using the global layout, the global controller can determine a global path to any pick-up station or drop-off station. For every global path request, the global controller places a global path response message in the mediator's message queue, which contains the global path. The message is forwarded by the mediator to the controller that requested the global path when the mediator executes its event. Afterwards, during the next local control cycle of this particular local controller, the global path is assigned to the vehicle.

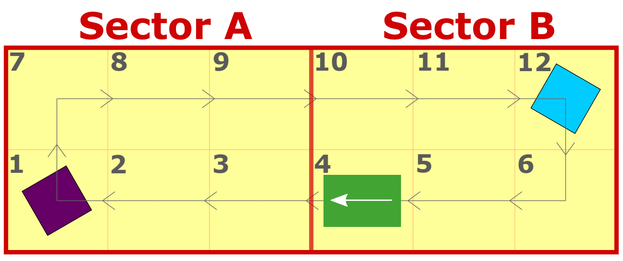
The communication process when assigning global paths to vehicle agents in the hierarchical control model and the corresponding sequence diagram is explained in more detail in Appendix B.3.

4.3.6 Vehicle transitions between local controllers

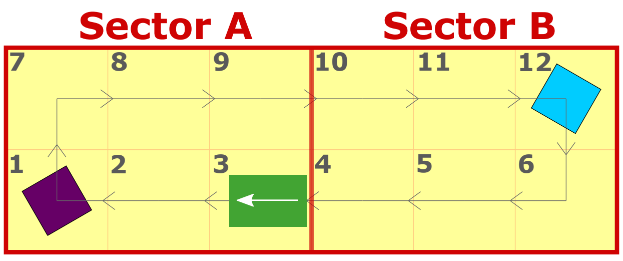
Requesting and approving sector transitions

Whenever a vehicle has reached an exit tile and the next tile of its global path is located in a different sector than the current sector, the vehicle needs to transition to the next sector, both physically and in the administration. In this process two controllers are defined: the *sending controller*, which is the local controller corresponding to the sector the vehicle is leaving, and the *receiving controller*, which is the local controller corresponding to the sector the vehicle is entering.

During the control cycle of the sending controller, the controller checks for all its vehicle agents whether it needs to be transitioned to another sector. If a vehicle needs to be transitioned, the sending controller sends a sector transition request message to the mediator if a request for this vehicle has not yet been sent. The moment when the sector transition request is sent has been visualized in Figure 4.18a.



(a) The moment a sector transition request is sent.



(b) The moment a vehicle is transferred.

Figure 4.18: Visualization of the exact moments that a sector transition request is sent and a vehicle is transferred from local controller B to local controller A.

A sector transition request message contains information on which entrance tile in the next sector is requested and the estimated time until the vehicle must stand still because the vehicle would otherwise surpass its waypoints, which is referred to as *time to standstill* in the remainder of this thesis. The time to standstill depends on the length of the remaining path of the vehicle and the vehicle speed, and *approximates* how long it will take for the vehicle to reach the end of its current path. The sector transition request message is placed in the mediator's message queue by the sending controller and is forwarded by the mediator to the receiving controller when the mediator executes its event.

During the control cycle of the receiving controller, the receiving controller processes all sector transition request messages that it has received.
If, for any sector transition request, the requested entrance tile is already reserved, this sector transition request is turned down for now. If not, the entrance tile is reserved for the vehicle if said vehicle is the highest-priority vehicle that requested this particular entrance tile. Vehicle priority is based on the *moment of time* at which the corresponding vehicle must stand still. This time is defined as

$$T_{\text{standStill}} = T_{\text{message}} + \frac{d_{\text{pathEnd}}}{s_{\text{vehicle}}} \tag{4.1}$$

where T_{message} is the time at which the vehicle sector transition request was sent, d_{pathEnd} the distance to the end of the path, and s_{vehicle} the vehicle speed at the moment the sector transition request was sent. Vehicles with a smaller $T_{\text{standStill}}$ are prioritized over other vehicles with a larger $T_{\text{standStill}}$ which are requesting the same entrance tile.

For every sector transition request that is approved, the receiving controller places a sector transition approval message in the mediator's message queue, which is forwarded to the sending when the mediator executes its event. Such a message contains the coordinates corresponding to the entrance tile which has been reserved. Based on these coordinates, the sending controller can compute waypoints to transition between sectors from the exit tile to the entrance tile. Sector transition requests that were turned down remain in the list until they are approved in a later cycle.

Registering and deregistering a vehicle

If a vehicle agent has received sector transition approval and waypoints have been determined to transition between sectors from the exit tile to the entrance tile, the vehicle agent needs to be deregistered from sending controller the and registered with the receiving controller. This is only done when the vehicle has physically left the exit tile of the sending sector, which also implies that the vehicle has physically entered the entrance tile due to the squared shape of the tiles. This moment has been visualized in Figure 4.18b. At this moment, the receiving controller must take over from the sending controller, such that the vehicle agent can participate in tile assignment and deadlock avoidance and detection of the new controller.

The sending controller first deregisters the vehicle agent from its own controller and then sends a vehicle registration message to the mediator, which contains the information required to register the agent with the receiving controller. At this moment in time, the agent is temporarily not registered with any controller. The vehicle registration message is placed in the mediator's message queue by the sending controller and is forwarded by the mediator to the receiving controller when the mediator executes its event. In contrast to the vehicle update messages and global path requests, the vehicle registration message is *not* processed in the control cycle of the receiving controller. Instead, the mediator immediately triggers the vehicle registration mechanism of the receiving controller. This is done to make sure that the vehicle agent is not registered with any controller for the shortest amount of time. If the agent would first be registered with the receiving controller and then deregistered with the sending controller, the agent would temporarily be registered with two controllers at the same time. This raises new questions, for example *which of the two controllers is responsible for actually controlling the vehicle?*

As was already mentioned, the global controller keeps track of which vehicle is located in which sector in order to forwards vehicle update messages to the correct local controller. If a vehicle transfers from one sector to another, this administration is updated such that vehicle update delayed messages are sent to the correct local controller in the future. Therefore, the global controller administration needs to be updated if the vehicle has physically left the exit tile of the sending sector. This is done using a controller update message, which is placed in the mediator's message queue and is forwarded by the mediator at the exact same moment as a vehicle registration message. The controller update message is not processed in the control cycle of the global controller. Instead, the mediator immediately triggers the administration update mechanism of the global controller. This is done to ensure that no vehicle updates are sent to the wrong controller as a result of the global controller administration being outdated.

In Appendix B.4 the communication process corresponding to vehicle sector transitioning from one sector to another in the hierarchical control strategy is explained in more detail. Additionally, the corresponding sequence diagram can be found in this appendix.

4.4 The global control cycle

The global controller only has one type of event, which is to perform its global control cycle. This cycle is performed periodically and consists of 2 procedures, each of which is introduced shortly.

1. **Forwarding vehicle updates:** The global controller checks whether it has received any vehicle updates from any of the vehicles since the last global control cycle. If so, for each of these vehicle updates the global controller determines in which sector the corresponding vehicle is currently located using its global administration. Then, the global controller forwards this vehicle update to the corresponding local controller via the mediator using a vehicle update delayed message.

2. **Processing global path requests:** The global controller verifies whether any local controllers have requested a global path for one of its agents since the last global control cycle. For every request, the global controller determines the global path from the last reserved tile of the agent to the target that is specified in the request. The global controller sends a global path response message via the mediator to the controller that requested the global path.

4.5 The local control cycle

Each local controller has two types of events; performing periodic deadlock detection and performing its local control cycle. Periodic deadlock detection is left out of scope for this thesis, for more information [Fransen et al., 2022] can be consulted. Every local controller periodically performs its local control cycle, independent of all other local controllers. The local control cycle consists of a total of 12 procedures, some of which are similar or identical to some of the procedures introduced in Section 3.2. Each of these procedures is briefly presented next.

1. Processing vehicle updates: Similar to the supervisory control cycle introduced in Section 3.2, the local controller processes all vehicle updates that it has received from the vehicles in its own sector since the last control cycle. For every vehicle, the controller evaluates whether any nodes have been reached and any tiles have been left. If the vehicle reached its local target but not its global target (i.e. a pick-up station or drop-off station), only the local target and path of the vehicle agent are cleared such that the vehicle can receive a new local target. However, if the vehicle reached its global target and has just finished a loading or unloading action, both the global and local targets and paths are cleared such that the vehicle can receive both a new global and local target.

2. Processing left tiles: Identical to the supervisory control cycle introduced in Section 3.2, the local controller releases the reservations of any tiles that have been left such that other vehicle agents can reserve these tiles in the future.

3. Processing reached nodes: Identical to the supervisory control cycle introduced in Section 3.2, the local controller processes any nodes that have been reached. The vehicle agent administration that keeps track of when a vehicle last reached a node is updated, such that the most recent information can be used in local path planning and when determining vehicle agent priority.

4. Processing global path responses: The local controller checks whether it has received any global path responses since the last local control cycle. If so, the local controller assigns the global path to the vehicle agent for which the global path is meant. Additionally, the global target is updated to the correct point of interest.

5. Assigning new global targets and global paths: Similarly to the supervisory control cycle introduced in Section 3.2, the local controller determines for each vehicle agent whether it has a job and whether it has a global target. Depending on whether the agent has a job and global target, one of three things can happen:

- **Agent has a job but no global target:** If the vehicle agent is not yet waiting for a global path, i.e. a global path request has not yet been sent to the global controller in one of the previous cycles, a new global target and global path towards this target has to be determined. If the vehicle is not yet loaded, the new global target is the pick-up station of the agent's job. On the other hand, if the vehicle is already loaded, the new global target is the drop-off station of the agent's job. The local controller sends a global path request to the global controller, who will determine a global path during the next global control cycle.
- **Agent has a job and a global target:** The vehicle might need a local target and path. For now, the local controller does nothing.
- **Agent has no job and no global target:** The vehicle is standing still and is waiting for a new job to arrive. The local controller does nothing.

In contrast to the supervisory control model that was introduced in Chapter 3, the functionality to plan paths to parking spots has not yet been implemented in the hierarchical control model due to time limitations. As a result of this, vehicles that have no job currently stand still in the middle of the layout, possibly blocking other vehicles. In the future, however, it is recommended that this functionality is implemented, as is discussed in more detail in Section 6.2.

6. Assigning new local targets and local paths: The local controller determines for every vehicle agent whether it has a local target and a global target. Depending on whether the agent has a local and global target, the controller does one of the following things:

- **Agent has a global target but no local target:** The vehicle agent has a global target, but needs to be assigned a local target. We define the *current sector* as the sector in which the vehicle is currently located, and the *next sector* as the sector in which the next tile in the global path, i.e. the tile to which a new local path must be determined, is located. Two cases can be distinguished:
    - **The current and next sector are identical:** The local controller determines an ordinary local path to the next local target. In [Fransen et al., 2020], the path planning algorithm used to determine the local path is explained in detail.

    - **The current and next sector are different:** The vehicle must currently be located on an exit tile and the next tile to which a path should be planned is an entrance tile in another sector. If the vehicle agent is not yet waiting for sector transition approval, i.e. a sector transition request has not yet been sent, the sending controller sends a sector transition request to the receiving controller.
- **Agent has a global target and a local target:** The vehicle is performing an ordinary job. The local controller does nothing.
- **Agent has no global target:** The vehicle must first be assigned a global target before a local target and local path can be assigned. For now, the local controller does nothing.

7. Processing sector transition requests: The local controller checks whether it has received any sector transition requests since its last cycle. If it has received more than one sector transition request, vehicle priority is determined based on the moment of time at which a vehicle must stand still, which is computed using Equation 4.1. The lower the estimated time, the higher the corresponding vehicle agent is placed on the priority list. The local controller iteratively attempts to reserve the entrance tile that is requested by each vehicle agent, starting with the highest-priority agent. If a requested entrance tile is already reserved by another vehicle agent, this request is turned down for now. On the other hand, if a requested entrance tile is not yet reserved, it is reserved for the requesting vehicle and a sector transition approval message is sent to the controller that requested the entrance tile. This approval message contains the coordinates of the entrance tile, so that waypoints to transition between sectors from the exit tile to the entrance tile can be generated by the sending controller. Other sector transition requests that are treated after this request and that request the same entrance tile are turned down because the entrance tile is no longer available. Finally, the sector transition requests that have been approved are removed from the list of sector transition requests, so that they are no longer evaluated in the future.

8. Processing sector transition approvals: The local controller verifies whether it has received any sector transition approvals since the last local control cycle. Based on the coordinates that were sent along with the sector transition approval message, the controller computes waypoints to transition between sectors from the exit tile to the entrance tile. These waypoints are sent to vehicle communication using a vehicle commands message, such that vehicle logic can append them to its drivepath.

9. Transferring vehicle agents to the next sector: The local controller transfers all vehicle agents that have received sector transition approval, have obtained waypoints, and *have physically left the exit tile*, to their next local controller. If the vehicle has physically left the exit tile, it is implied that it has also physically entered the entrance tile due to the squared shape of the tiles. The local controller deregisters the agent from its own controller and sends a vehicle registration message to the receiving controller. The receiving controller registers the agent in its own administration after the message is forwarded by the mediator. Lastly, a controller update message is sent to the global controller. As soon as this message is forwarded by the mediator, the global controller administration is updated.

10. Periodic rerouting: Identical to the supervisory control cycle introduced in Section 3.2, the local controller checks whether any of the vehicle agents is candidate for periodic rerouting. If the last time that an agent received a new path is sufficiently long ago, the controller assigns a new local path to the agent if there exists a shorter path to the local target than the path that the agent currently has.

11. Providing new tiles: Identical to the supervisory control cycle introduced in Section 3.2, the local controller determines for which vehicle agents additional tiles need to be reserved. The controller iteratively attempts to reserve the next tile for the agent that is on top of the priority list. This is only possible if the attempt tile reservation passes the free tile check and deadlock check. In [Fransen et al., 2022] the deadlock avoidance algorithm is explained in full detail.

12. Computing vehicle commands: Identical to the supervisory control cycle introduced in Section 3.2, new vehicle commands (i.e. waypoints) are computed by the local controller. New waypoints are only computed for tiles that have already been reserved by the agent but whose waypoints have not yet been sent. If any new waypoints have been computed, these waypoints are sent to the correct vehicle through the mediator using a vehicle commands message.

4.6 Multi-sector deadlock

In Section 2.2.1 deadlock was defined as a state where "a group of vehicles is waiting for each other forever [...] because the next tile in the path of each vehicle is the last reserved tile of another vehicle". In this section a closer look is taken at *multi-sector deadlock*, as is defined according to Definition 4.12. An example of such a multi-sector deadlock can be seen in Figure 4.19.

Definition 4.12 (Multi-sector deadlock)**.** A *multi-sector deadlock* is a deadlock situation in which vehicles from multiple sectors are involved.

As is stated in [van Weert, 2019], a mono-cycle deadlock situation can be identified as a situation in which the circular wait condition valid. The most straightforward method to recover from a mono-cycle deadlock is to replan the path of (at least) one of the involved vehicles such that the circular wait is broken. Furthermore, a multi-cycle deadlock can be defined as a situation in which the modified circular wait condition is valid and its cycle is closed. Because multi-cycle deadlock is significantly more complicated, for now the focus is specifically on mono-cycle deadlock.

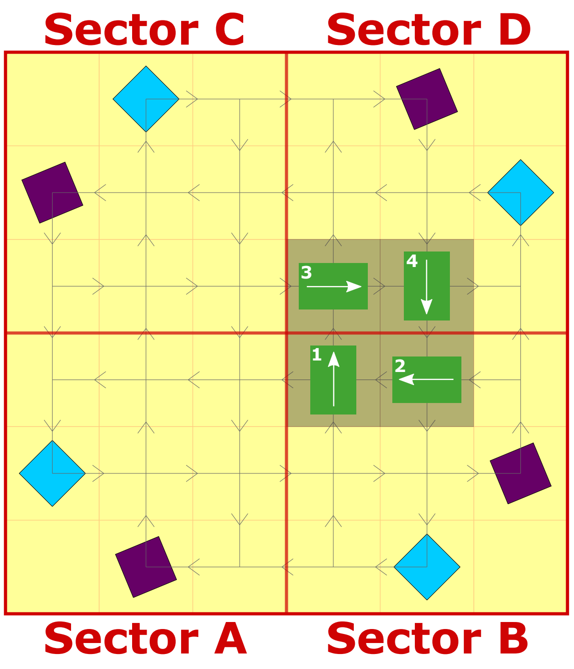


Figure 4.19: Example of a multi-sector deadlock.

Because vehicles from multiple sectors are involved in a multi-sector deadlock and each local controller only has access to local information, no circular wait can be found by the deadlock avoidance and detection algorithm. Therefore, this deadlock cannot be identified using the circular wait method. As an example, consider the example in Figure 4.19, where vehicles 1 and 2 from sector B and vehicles 3 and 4 from sector D are involved in a multi-sector deadlock. Local controllers B and D both only have access to local information, i.e. information on the vehicle agents and layouts in their own sector. If the deadlock avoidance and detection algorithm of local controller B is called, it cannot find a circular wait because it is unaware of the existence of vehicles 3 and 4. Likewise, local controller D also cannot find a circular wait because it is unaware of the existence of vehicles 1 and 2. This is a crucial consequence of local controllers only being able to use local information.

As a result of this, two types of problems arise:

1. **Multi-sector deadlock cannot be avoided:** If the system is not yet in a multi-sector deadlock and a vehicle agent wants to reserve a tile that would lead to multi-sector deadlock, the deadlock avoidance algorithm cannot predict that a circular wait would be formed.
2. **Multi-sector deadlock cannot be detected:** If the system is already in multi-sector deadlock, the deadlock detection algorithm cannot detect that a circular wait is present. Because of this, it is also not possible to recover from a multi-sector deadlock.

Of course, being unable to avoid and detect a multi-sector deadlock is a problem that can significantly damage the performance of the system. However, a relevant question is *how often does multi-sector deadlock occur?* In Chapter 5, more research on the occurrence of multi-sector deadlock is performed.

5 Comparison of results

In the previous chapter, the design of the hierarchical control strategy was elaborated upon in full detail. In order to estimate how well this strategy performs, the hierarchical control strategy that is designed in Chapter 4 is compared to the supervisory controller introduced in Chapter 3.

In this chapter, simulations are performed on three different layouts with a varying level of difficulty: a simple layout, an intermediate layout and an advanced layout. For each of these layouts, 10 simulations of 1800 [s] are performed for a variety of vehicle densities.

5.1 Simple layout

The first layout that is simulated is the *simple layout*, which has been visualized in Figure 5.1. As can be seen, this layout consists of two sectors, which each have one exit tile that leads to the entrance tile of the other sector. Moreover, sector A contains a pick-up station and sector B contains a drop-off station. As a result of the nature of this layout, vehicles will constantly drive around in circles from pick-up to drop-off station and from drop-off to pick-up station.

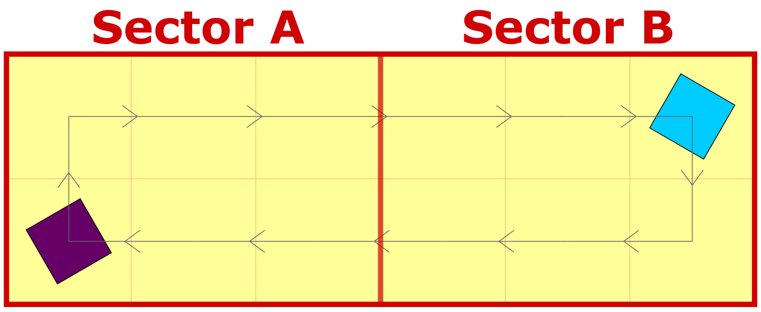


Figure 5.1: Visualization of the simple layout.

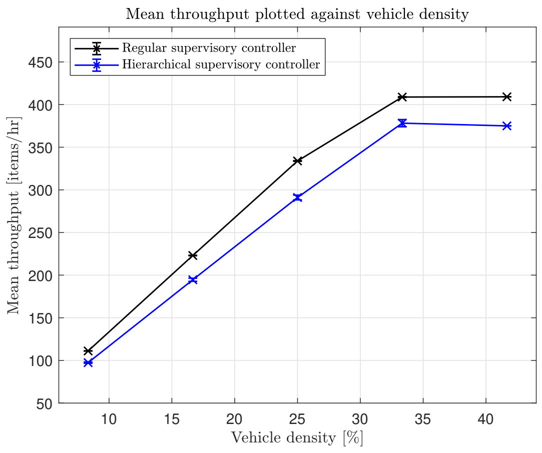
In Figure 5.2a and 5.2b the mean throughput respectively the average item lead time have been visualized for the simple layout and a vehicle density of approximately 8.33, 16.67, 25, 33.33 and 41.47 [%], corresponding to 1, 2, 3, 4 and 5 vehicles. As can be seen in these two figures, using the hierarchical control strategy as opposed to the supervisory controller leads to a slight decrease in the mean throughput and a slight increase in the average item lead time of the system for this particular layout, regardless of the vehicle density.

Every time a vehicle wants to transition to another sector, the sending controller sends a sector transition request message to the receiving controller. The receiving controller must approve this request before responding with a sector transition response message. Because it takes time for this process to take place, vehicles have to decelerate, come to a stop, and accelerate when transitioning between sectors. This largely explains the difference in average item lead time. If it is assumed that the maximum acceleration of a vehicle is 1 [m/s^2], the maximum deceleration of a vehicle is -1 [m/s^2], and the maximum vehicle speed is 1.5 [m/s], then the distance traveled while decelerating and accelerating can be computed as

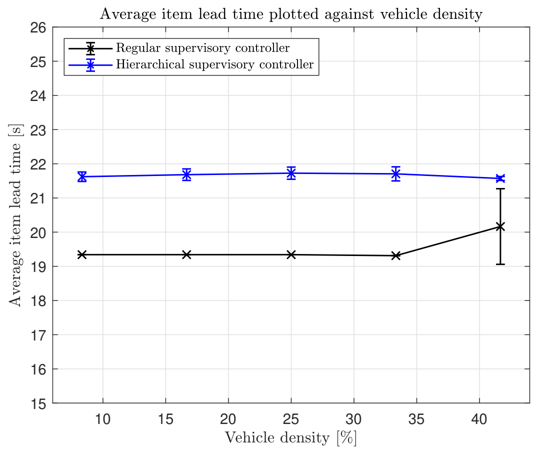
$$s = 2 \cdot \frac{1}{2}\frac{v_{\max}^2}{a_{\max}} = \frac{1.5^2}{1} = 2.25\,[\mathrm{m}] \tag{5.1}$$

where s is the traveled distance, $v_{\max}$ the maximum velocity and $a_{\max}$ the maximum acceleration.

So, when using the hierarchical control strategy a total distance of 2.25 [m] is spent decelerating and accelerating per sector transition, which takes a total of $2 \cdot (1.5/1) = 3$ [s]. In contrast, if a vehicle does not have to decelerate and accelerate it only takes a vehicle $2.25/1.5 = 1.5$ [s] to travel the same distance. As an example, the difference in average item lead time when only using one vehicle, i.e. at a vehicle density of 8.33 [%], is roughly 2.280 [s]. For this particular layout, a vehicle always needs to transfer once from sector A to B in order to go from pick-up to drop-off station, meaning that roughly $100 \cdot (3 - 1.5)/2.280 = 65.8$ [%] of the increase in lead time when using one vehicle is caused by deceleration and acceleration when crossing the border. Here it is assumed that a vehicle can accelerate up to maximum speed and does not have to decelerate again upon reaching the entrance tile, as a result of not enough tiles being reserved ahead. The remainder of the increase in lead time is likely caused by the fact that a vehicle spends time standing still while waiting for sector transition approval as a result of communication between the involved local controllers. As the item lead time increases, the throughput is expected to decrease as a result.



(a) Mean throughput against vehicle density.



(b) Average item lead time against vehicle density.

Figure 5.2: Mean throughput and average item lead time plotted against vehicle density for the simple layout.

In Figure 5.3 the *cumulative controller cycle time* has been plotted for all controllers, which is the total time a particular controller spends performing its cycle during a simulation. As can be seen in this figure, the total controller cycle time of all hierarchical controllers (i.e. all local controllers *plus* the global controller) exceeds the cumulative controller cycle time of the supervisory controller. It can also be seen that the cumulative controller cycle time of local controllers A and B are both lower than that of the supervisory controller. It is difficult to draw any conclusions on the performance of the hierarchical controller based on this figure, because it depends on the context in which the controller is used and how the system behaves in practice. If the control cycles of the two local controllers are mainly performed in parallel, then the distribution of the computational burden seems to be relatively effective (though at the cost of a slight decrease in throughput and a slight increase in lead time). On the other hand, if the control cycles of the two local controllers are mainly performed in series, the benefit of using a hierarchical control strategy is questionable, because the total cumulative controller cycle time of all hierarchical controllers is higher than that of the supervisory controller.

For example, in the extreme case where there is only one vehicle in the system, the local controllers are mainly run in series. If the vehicle is located in sector A, the cumulative controller cycle time of local controller A increases because time is spent performing its control cycle. In the meantime, the cumulative controller cycle time of local controller B does not increase because it does not have any vehicles to control[1]. If the number of vehicles in the system increases, the probability that a sector does not contain any vehicles decreases, and the probability that the two controllers are largely run in parallel increases. Whether the controllers are mainly used in parallel or in series during a simulation is difficult to identify based on the results of the simulation.

[1]In reality, local controllers also perform their control cycle if there are no vehicles located in their sector. However, because there are no vehicle updates to process, messages to send, messages to process, or vehicle commands to compute, the cycle time of such a cycle is significantly lower than when a controller does have vehicles to control.

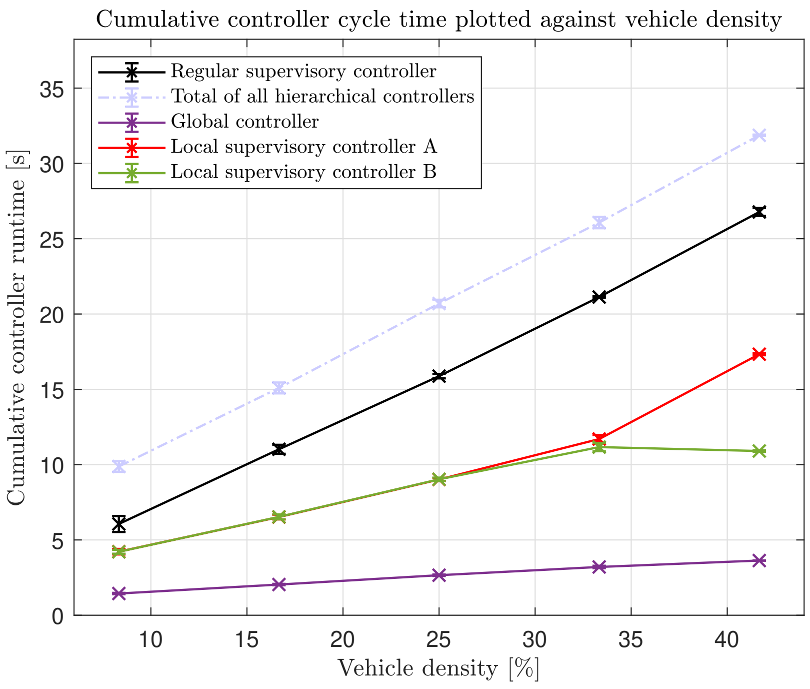


Figure 5.3: Cumulative controller cycle time plotted against vehicle density for the simple layout.

Lastly, a significant difference can be observed in the cumulative controller cycle time of local controllers A and B when using five vehicles (i.e. at a vehicle density of 41.67 [%]). This result can be explained by the fact that the layout is asymmetrical. At the beginning of the simulation, vehicles are placed at a random initial location. However, the first global target of a vehicle in a simulation is *always* the pick-up station. As a result of this, during the first few seconds of the simulation all vehicles drive past the drop-off station without stopping. On the contrary, all vehicles *do* stop at the pick-up station in order to load the vehicle. This leads to a situation where vehicles end up in a queue to use the pick-up station. As soon as every vehicle has picked up its first load the system ends up in a steady state because the number of pick-ups and drop-offs and the duration of pick-ups and drop-offs are identical.

An example of this can be seen in Figure 5.4, where the pink vehicle is currently being loaded at the pick-up station. Because there are constantly two vehicles waiting in sector A, this eventually leads to a higher cumulative controller cycle time for controller A. If the simulation is short enough such that this steady state where two vehicles are waiting in sector A cannot be reached, this difference in cumulative controller cycle time does not occur.

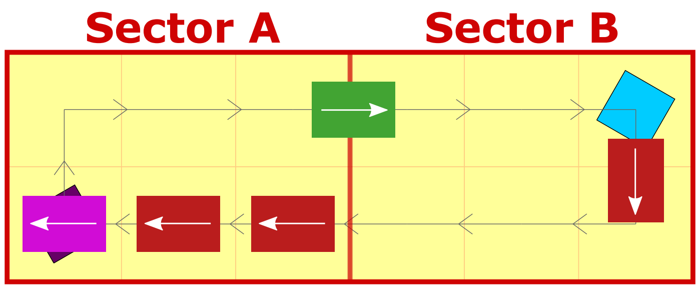


Figure 5.4: Visualization of the simple layout where vehicles end up waiting for the pick-up station.

5.2 Intermediate layout

In Figure 5.5 the *intermediate layout* has been shown. The layout has been partitioned into four sectors, each having a pick-up station and a drop-off station. For this layout, simulations are performed at a vehicle density of approximately 2.04, 4.08, 10.20, 14.29, 20.41 and 24.49 [%], corresponding to 1, 2, 5, 7, 10 and 12 vehicles.

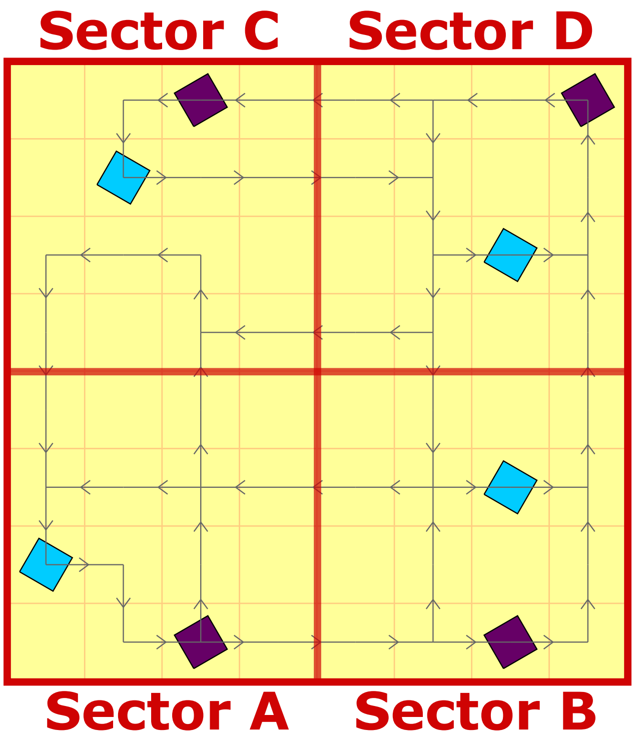


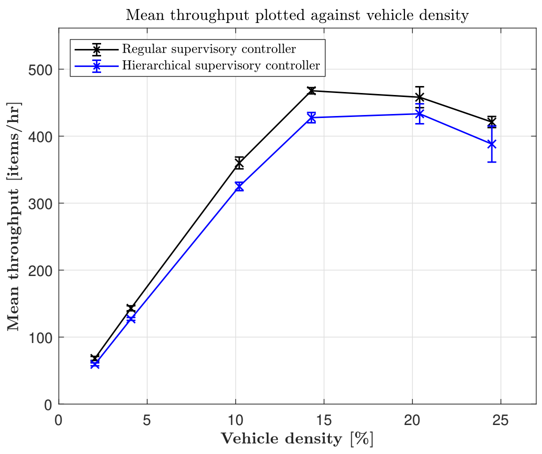
Figure 5.5: Visualization of the intermediate layout.

In Figure 5.6a and 5.6b the mean throughput respectively average item lead time have been shown as a function of vehicle density. Similar to the simple example, the mean throughput slightly decreases and the average item lead time slightly increases as a result of using the hierarchical control strategy, which is a result of a combination of factors.

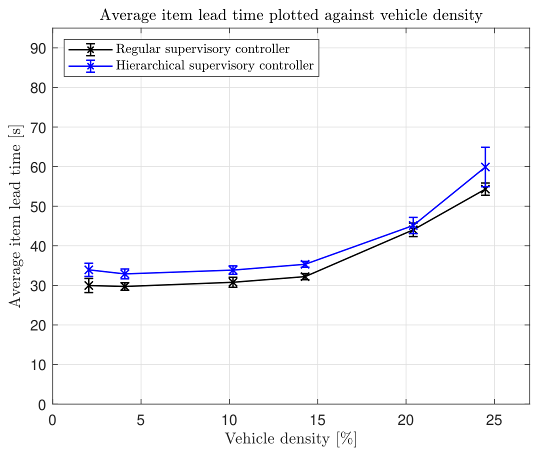
First of all, as was already discussed, vehicles have to decelerate and accelerate when transitioning between sectors. For this layout, the difference in average item lead time when only using one vehicle is roughly 3.960 [s]. There exist 16 combinations of pick-up and drop-off stations, of which the average number of sector transitions when travelling from a pick-up to a drop-off station is two. Using the same calculation as in Section 5.1, it can be computed that roughly $100 \cdot 2(3 - 1.5)/3.960 = 75.8$ [%] of the increase in lead time when using one vehicle is caused by deceleration and acceleration when crossing the border. Here it is again assumed that a vehicle can accelerate up to maximum speed and does not have to decelerate again upon reaching the entrance tile, as a result of not enough tiles being reserved ahead. In addition, vehicles also spend time waiting when transitioning between sectors are a result of communication between local controllers.

Furthermore, global paths from POI to POI might not be optimal. Using the hierarchical control strategy, only an approximation is made on what is the best possible global path. In this approximation congestion in the layout is not taken into account. Even though local controllers are able to take into account node weights based on the path planning algorithm introduced in [Fransen et al., 2020], they can only do so on a local scale. When using the regular supervisory controller, congestion in the layout can be taken into account for the entire layout.

Finally, periodic rerouting of vehicles can also only be performed on a local scale using the hierarchical control strategy. To be more specific, a local controller can only attempt to periodically reroute a vehicle as long as the local target remains the same. If there were to be a shorter path that makes use of a different exit tile, the local controller cannot plan to this exit tile because the global path is fixed. As can be derived from Figure 5.5, in this layout it is not possible periodically reroute the path in any of the sectors because there do not exist any alternative routes.



(a) Mean throughput against vehicle density.



(b) Average item lead time against vehicle density.

Figure 5.6: Mean throughput and average item lead time plotted against vehicle density for the intermediate layout.

In Figure 5.7 the cumulative controller cycle time has been plotted for all controllers. For this layout, the total cumulative controller cycle time of all controllers in the hierarchical control strategy is higher than the cumulative cycle time of the supervisory controller, except for a vehicle density of 24.49 [%]. This can be explained by the fact that for a higher vehicle density, the deadlock avoidance and detection algorithm in the regular supervisory controller has a lot more possible deadlock cycles to evaluate and possibly resolve. Therefore, the cumulative cycle time of the supervisory controller tends to explode as the vehicle density becomes too high. Using the hierarchical control strategy, such a cycle contains fewer vehicles because vehicle agents are split over multiple controllers. In practice, however, the system is unlikely to ever be run at a vehicle density of 24.49 [%]. Judging from Figure 5.6a, the peak throughput using both controllers is reached at a vehicle density somewhere between 15 and 20 [%]. Using a higher vehicle density only increases the cost because vehicles are expensive to produce.

Furthermore, the cumulative controller cycle time of controller D is approximately 5 to 6 times larger than that of the other local controllers. The main reason for this is the fact that sector D serves as a hotspot in the layout. If a vehicle needs to go from a pick-up to a drop-off station, in a lot of cases it has to go through sector D. As a result of this, sector D is usually filled with a lot of vehicles, which significantly increases the cumulative cycle time of the controller in this sector because the deadlock avoidance and detection algorithm in this sector has a lot more vehicles to evaluate. Consequently, if a lot of vehicles are located in sector D it is implied that little vehicles are located in sector A, B and C, which leads to a reduced cumulative cycle time of the controllers corresponding to these sectors.

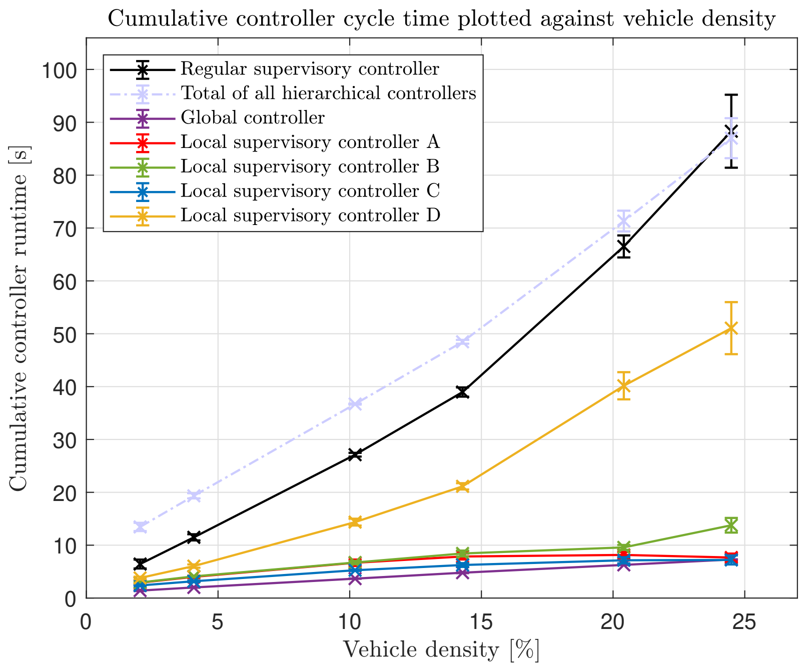


Figure 5.7: Cumulative controller cycle time plotted against vehicle density for the intermediate layout.

5.3 Advanced layout

Finally, the *advanced layout* is simulated, which is the ParcelGrid layout that was already introduced in Section 2.2.1. The advanced layout and an initial sector partitioning can be seen in Figure 5.8. The advanced layout is partitioned into four sectors, where each sector has two pick-up stations and four drop-off stations. Simulations on this layout are performed for 1, 9, 18, 27, 36 and 45 vehicles, which corresponds to a vehicle density of 0.56, 5, 10, 15, 20 and 25 [%] if only the tiles in the working area are regarded.

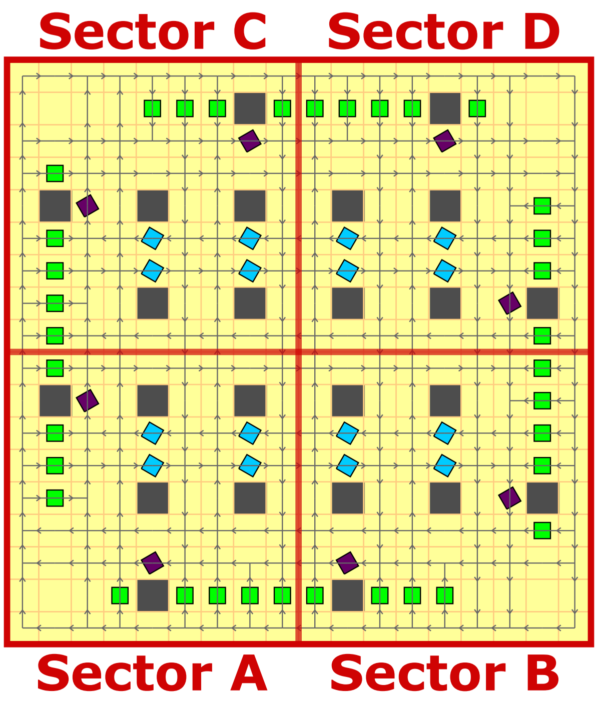
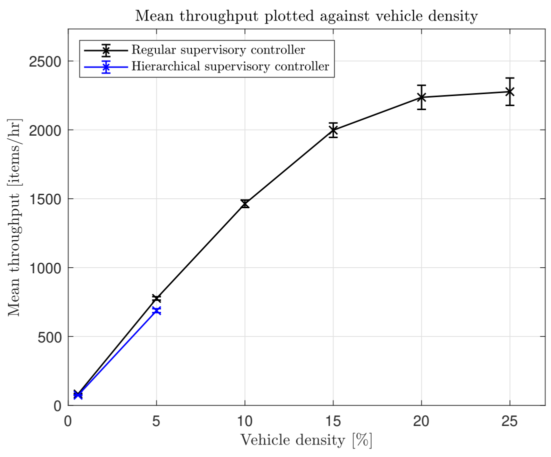
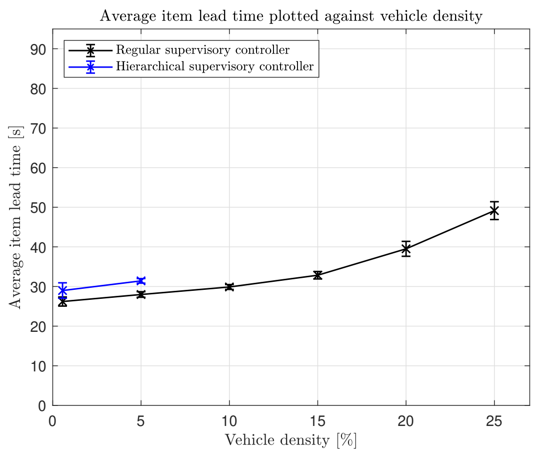


Figure 5.8: Visualization of the advanced layout.

In Figure 5.9a and 5.9b the mean throughput and average item lead time have been shown for the advanced layout at different vehicle densities. However, as can be seen, for the hierarchical control strategy only the results at a vehicle density of 0.56 and 5 [%] have been plotted. For a vehicle density of 10, 15, 20 and 25 [%] the hierarchical control model namely always ends up in a multi-sector deadlock, which essentially leads to a mean throughput of zero as the simulation time goes to infinity.



(a) Mean throughput against vehicle density.



(b) Average item lead time against vehicle density.

Figure 5.9: Mean throughput and average item lead time plotted against vehicle density for the advanced layout.

As an example, the moving average throughput of a single simulation of this layout with 27 vehicles has been plotted in Figure 5.10, in which a time window of 30 [s] is used to average the throughput. From this figure it can be derived that the moving throughput significantly drops as the simulation progresses because more and more vehicles end up in deadlock. Eventually, after a simulation time of roughly 200 [s], no more jobs are finished as a result of deadlock in entire the system.

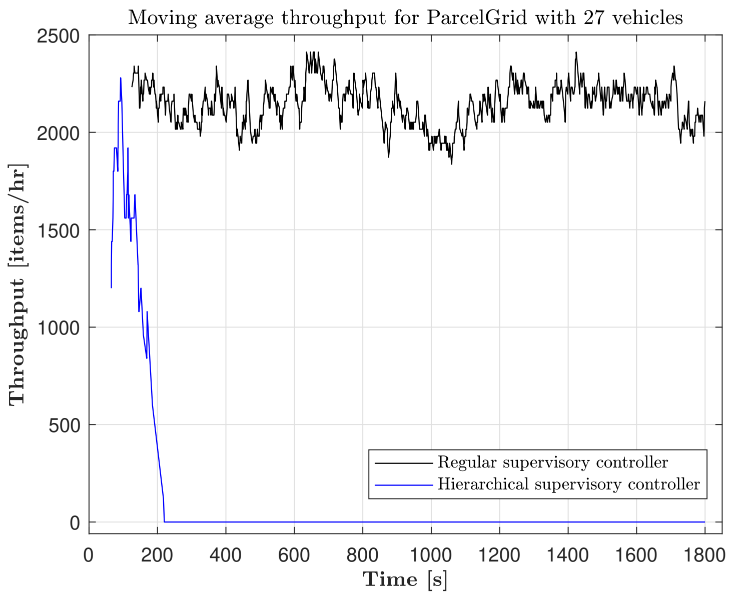


Figure 5.10: Moving throughput for a single run on the advanced layout using 27 vehicles and a time window of 30 [s].

Upon closer inspection of the layout and corresponding sector partitioning, it can be identified that this problem is largely caused by the critical regions shaded in blue in Figure 5.11. Each of these critical regions indicates a two-by-two area where only four vehicles are required in order to form a circular wait in the region. In addition, for each of these regions, the tiles are split among multiple sectors, which results in a situation where each of the involved controllers is never able to avoid or detect a circular wait. Moreover, vehicles practically always use at least one of these regions when performing their job[1] and because they do so, these regions are usually crowded.

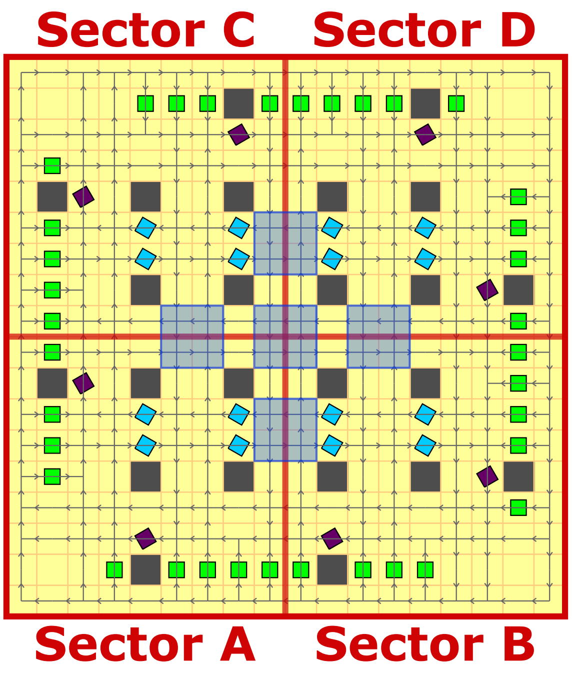
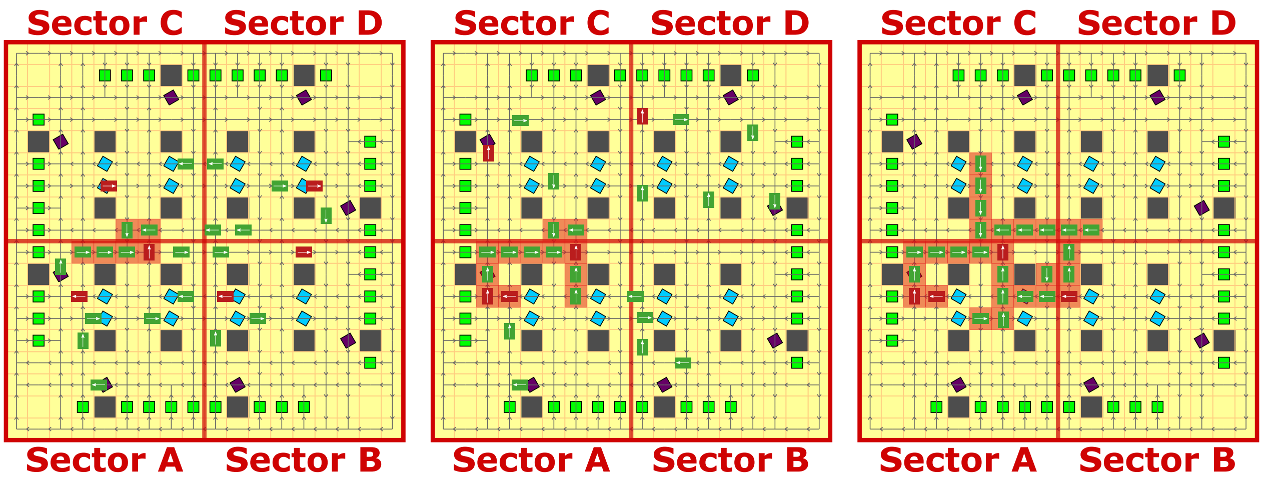


Figure 5.11: Visualization of the advanced layout with critical regions indicated in blue.



(a) Snapshot after 105 [s]. **(b)** Snapshot after 140 [s]. **(c)** Snapshot after 205 [s].

Figure 5.12: Visualization of how a small multi-sector deadlock situation can evolve into a large multi-sector deadlock in a simulation with 27 vehicles.

[1]Vehicles can also use the tiles that are located further to the outside, but it unlikely that none of the blue-shaded areas is part of the shortest global path.

On top of that, as soon as the first four vehicles end up in a multi-sector deadlock situation, it is usually a matter of time before other vehicles end up being blocked by this deadlock. An example of how one small multi-sector deadlock can lead to a situation where all vehicles end up in deadlock can be seen in Figure 5.12. In this figure, a snapshot of the system state at respectively 105, 140 and 205 [s] has been shown for a simulation with 27 vehicles. The tiles that are occupied by vehicles that are involved in a multi-sector deadlock are shaded in red. As can be seen, the multi-sector deadlock gradually expands in size. This effect is reinforced if more vehicles are added to the environment because the probability that a vehicle is blocked by one of these deadlocks increases. Remember, none of the involved local controllers is able to identify that the system is stuck in this state forever.

The cumulative controller cycle time of each controller has been plotted in Figure 5.13. Because of the near-symmetrical nature of the layout the cumulative controller cycle times of all local controllers are roughly the same over 10 simulations. Although the total cumulative controller cycle time of the hierarchical control strategy is lower than the cumulative cycle time of the supervisory controller for all vehicle densities except 5 [%], it is unfair to draw any conclusions based on this because the hierarchical control model ended up in a multi-sector deadlock situation at all these vehicle densities.

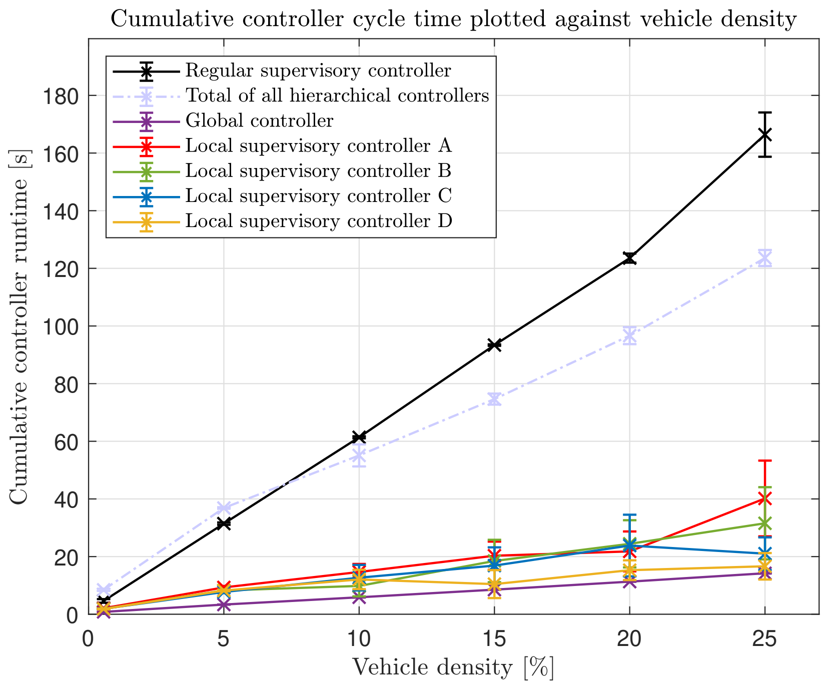


Figure 5.13: Cumulative controller cycle time plotted against vehicle density for the advanced layout.

In an attempt to reduce the probability that the system ends up in a multi-sector deadlock, the locations of the sector borders are adjusted. The advanced layout and its new sector partitioning can be seen in Figure 5.14. As can be seen, the sector borders have been shifted such that the tiles of each critical region indicated in Figure 5.11 are not split over multiple sectors. Because of this, there is always one local controller that has full information on all tiles involved in such a critical region, meaning that it can at least always detect if deadlock has occurred. Additionally, all entrance tiles only have one incoming edge, which is the edge coming from the corresponding exit tile, such that reserving an entrance tile can never *directly* lead to a (mono-cycle) deadlock.

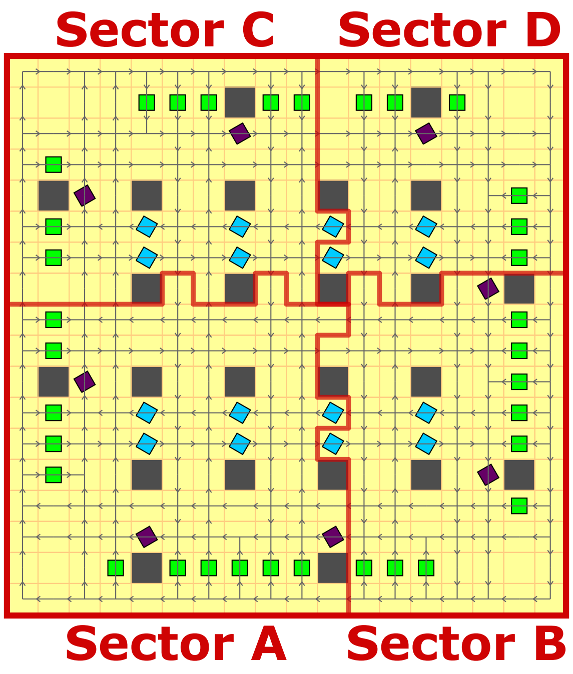


Figure 5.14: Visualization of the advanced layout with an alternative sector partitioning.

It is still possible that a multi-sector deadlock occurs as a result of vehicles blocking each other, but this is a consequence of local controllers only being able to use local information. For example, reserving the red-shaded entrance tile in Figure 5.15 for the vehicle that is on the exit tile in sector B does not directly lead to multi-sector deadlock. Nevertheless, as soon as the two vehicles that are behind the red-shaded vehicle also move forward, a multi-sector deadlock occurs. In contrast to the sector partitioning in Figure 5.8, there are always more than four vehicles required to form a multi-sector deadlock using this sector partitioning, because there are no more two-by-two regions crossing the border in which a non-detectable circular wait could occur.

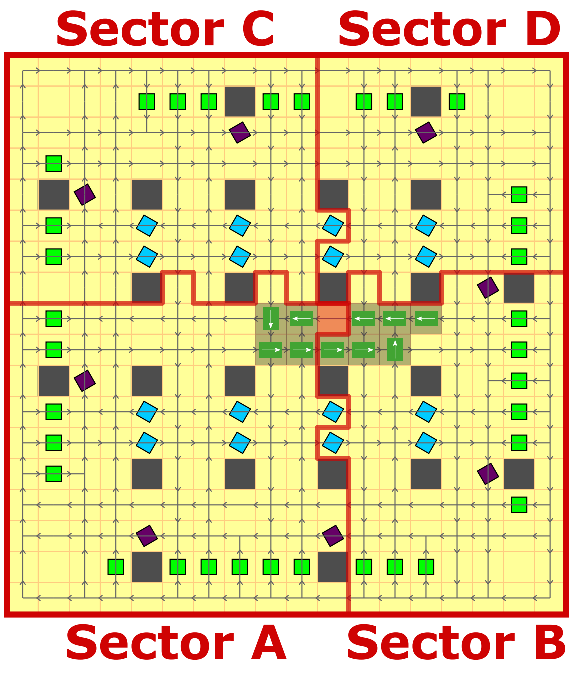
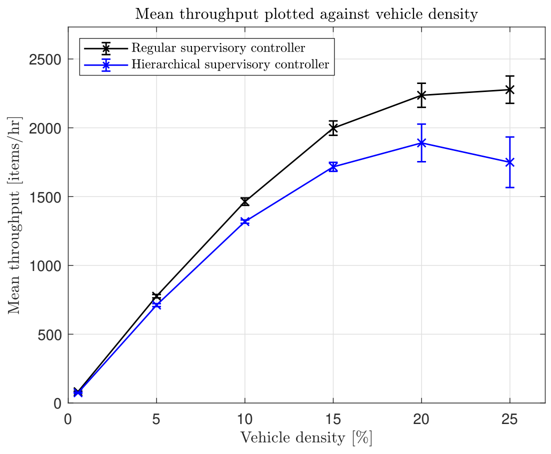
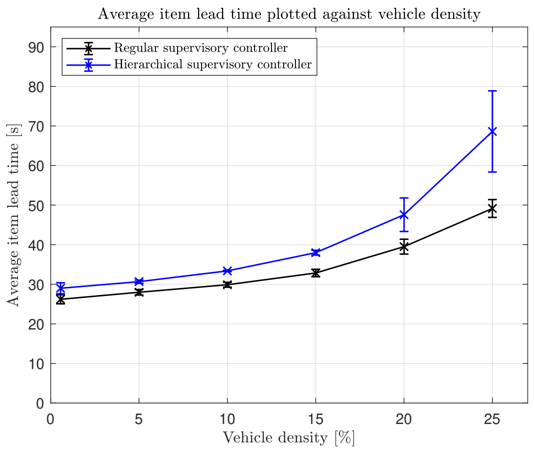


Figure 5.15: Visualization of the advanced layout where multi-sector deadlock can occur as a result of vehicle blocking.

In Figure 5.16a and 5.16b the mean throughput and average item lead time using the *new sector partitioning* have been plotted for a vehicle density of 0.56, 5, 10, 15, 20 and 25 [%], corresponding to 1, 9, 18, 27, 36 and 45 vehicles. Using this sector partitioning, the system never ended up in multi-sector deadlock in any of the simulations for a vehicle density of 0.56, 5, 10, 15 and 20 [%]. For a vehicle density of 25 [%], a total of 16 runs were performed out of which 6 resulted in a multi-sector deadlock, resulting in 10 runs where no multi-sector deadlock occurred. Only the non-deadlock cases have been plotted in Figure 5.16a and 5.16b.



(a) Throughput against vehicle density.



(b) Average item lead time against vehicle density.

Figure 5.16: Throughput and item lead time plotted against vehicle density for the advanced layout using the new sector partitioning.

As expected, the mean throughput decreases and the average item lead time increases when using the hierarchical control strategy as opposed to the supervisory controller. The difference in average item lead time when only using one vehicle is roughly 2.784 [s]. For this layout, there exist 128 combinations of pick-up and drop-off stations, of which the average number of sector transitions when travelling from a pick-up to a drop-off station is 0.969. Using the same calculation as in Section 5.1 and 5.2, it can be computed that roughly $100 \cdot 0.969(3 - 1.5)/2.784 = 52.2$ [%] of the increase in lead time when using one vehicle is caused by deceleration and acceleration when crossing the border.

For the supervisory controller, the peak throughput is likely to be reached at a vehicle density of roughly 25 [%]. The hierarchical controller, on the other hand, already reaches its peak throughput at a vehicle density of about 20 [%]. Because using more vehicles only increases the expenses and does not yield a higher throughput, the hierarchical control model is expected to be run at a vehicle density of 20 [%]. Since a multi-sector deadlock was not observed during any of the simulations at a vehicle density of 20 [%], it is assumed that multi-sector deadlocks do not pose an enormous problem at this vehicle density. However, because only 10 simulations were run for 1800 [s], this cannot be guaranteed based on these simulations.

Finally, in Figure 5.17 the cumulative controller cycle time of each controller can be seen as a function of vehicle density using the new sector partitioning. Interestingly, the total cumulative controller cycle time of the hierarchical control strategy is slightly *lower* than the cumulative controller cycle time of the supervisory controller for vehicle densities above 17.5 [%]. At the vehicle density where the peak throughput is obtained for both controllers, the hierarchical control model *always* has a lower cumulative controller cycle time than the supervisory controller, even in the worst case where all local controllers perform their control cycle in series. This does, however, come at a cost of a decrease in mean throughput and an increase in average item lead time.

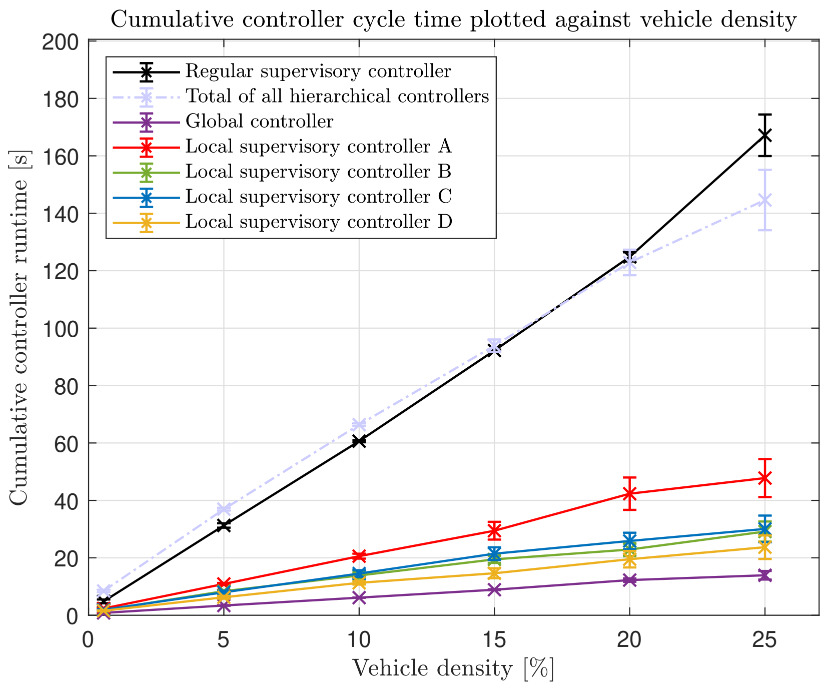


Figure 5.17: Cumulative controller cycle time plotted against vehicle density for the advanced layout using the new sector partitioning.

6 Conclusion and recommendation

In this thesis, a new hierarchical control strategy was developed, which aimed at exploiting the scalability of decentralized approaches while limiting the model complexity and typically non-optimal character of decentralized approaches. Although Vanderlande already has an existing control strategy in place for their real-life AGV system called FLEET, Vanderlande is also exploring alternative routing controllers using their grid-based control model. The performance, potential and problems of the new hierarchical control strategy are compared with respect to the existing controller in Vanderlande's grid-based control model, called the supervisory controller. The main research question to be answered is *what does the design of a hierarchical control strategy look like and how can its performance be compared with the supervisory controller?*

In the first section of this chapter, conclusions are drawn regarding the design and performance of the hierarchical control strategy in comparison with the supervisory controller. Thereafter, a couple of recommendations are given based on which further research can be performed.

6.1 Conclusion

To begin with, a literature study was performed in order to obtain an overview of existing path planning and traffic control techniques in literature. Although optimization-based and metaheuristic algorithms can be useful under specific circumstances, both of them generally do not scale that well. Because Dijkstra's algorithm, the A* algorithm and the D* algorithm are all very computationally efficient, complete and optimal (given that the heuristic cost of the A* and D* algorithm is admissible), these algorithms are typically used as graph search algorithms in path planning. In order to coordinate vehicles in a multi-vehicle system and to make sure that these vehicles can successfully perform their jobs, a traffic controller has three major responsibilities: collision avoidance, deadlock handling and livelock handling. Omission of any of these three responsibilities can lead to damaged vehicles, a system that is permanently stuck in a state where no progression is made, or a system that constantly switches between system states without achieving some desired state.

In general, a rough distinction between three types of control structures can be made, namely centralized approaches, decentralized approaches and hierarchical control approaches. Because centralized controllers have access to global information on all vehicles, a global optimum can be reached relatively easily. However, centralized control approaches are considered to be computationally expensive and unsuitable for larger scaled systems, which is a critical problem. Because the computational burden is distributed over multiple controllers in decentralized control approaches, these approaches are generally less computationally expensive and more suitable for larger scaled systems than centralized approaches. However, because of the fact that only local information is available, it is less straightforward to reach a global optimum. Additionally, by using only local information, it is more difficult to avoid congestion and deadlocks in comparison to centralized approaches.

The hierarchical control strategy that is designed in this thesis can be distinguished into a centralized and a decentralized layer. Therefore, a distinction between global and local properties and objects is made. Global paths are determined by a centralized global controller and serve as an approximation of the shortest path that each vehicle must take to traverse from one location to another location. The layout is partitioned into multiple sectors in which traffic is controlled using a decentralized local controller, which only has access to local information. Because the desired properties of the controller in a particular sector depend on the characteristics of that sector, the properties of a controller are allowed to vary per sector. For example, the desired properties of the controller in a storage area of an airport might be different in comparison to a transportation area between terminals. Furthermore, a handful of new message types are introduced so that vehicles can successfully switch between sectors and can correctly perform their jobs.

Using the current implementation of the hierarchical control strategy, the mean throughput of the system typically slightly decreases and the average item lead time typically slightly increases in comparison to the supervisory controller. This can largely be explained by the fact that vehicles have to decelerate, come to a stop, and accelerate every time they transition from one sector to another. Additionally, a global path is only an approximation of the shortest path a vehicle must take in order to travel from one point of interest to another. Moreover, periodic rerouting of vehicles can only be performed on a local scale because global paths are fixed. Each of these limitations is *not* a consequence of using a hierarchical control strategy in general, but merely of the current implementation of the hierarchical control strategy. If a solution to these limitations can be implemented, it is expected that the decrease in mean throughput and increase in average item lead time can be reduced.

The cumulative controller cycle time of each individual local controller, i.e. the total time the controller spends performing its cycle during a simulation, is consistently significantly lower than the cumulative controller cycle time of the supervisory controller. However, in most cases. the total controller cycle time of all hierarchical controllers, i.e. all local controllers plus the global controller, exceeds the cumulative controller cycle time of the supervisory controller. It is difficult to draw any conclusions on the performance of the hierarchical control strategy based on this because it depends on the context in which the controller is used. Because the grid-based control model is simulated using a discrete-event simulation, during the simulation all controllers perform their control cycle in series. In practice, however, the controllers can perform their control cycle in parallel with other controllers. If the control cycles are mainly performed in parallel, then the distribution of the computational burden seems to be relatively effective, though typically at the cost of a slight decrease in throughput and a slight increase in lead time. On the other hand, if the control cycles are mainly performed in series, the benefit of using a hierarchical control strategy is questionable, because in most cases the total cumulative controller cycle time of all hierarchical controllers is higher than that of the supervisory controller. Whether the local control cycles are mainly performed in parallel or in series during a simulation is difficult to identify and depends on the distribution of the vehicles over the controllers, which changes over the course of a simulation. Even if the local control cycles are mainly performed in series, the hierarchical control mechanism can still be used as a means to successfully introduce different types of local controllers with varying properties, depending on the desired properties per region, i.e. per sector, of the layout.

A significant problem that can arise using the designed hierarchical control strategy is the formation of a multi-sector deadlock. Such a deadlock is a situation where a group of vehicles is waiting for each other forever because their next tile is already occupied or reserved by another vehicle, and the involved vehicles are located in different sectors. Because each local controller only has access to local information, such a deadlock cannot easily be identified. As soon as a small number of vehicles end up in a multi-sector deadlock situation, it is usually a matter of time before other vehicles start being blocked by this deadlock, possibly leading to a situation where all vehicles end up in deadlock. The way in which a layout is partitioned into sectors can make a major difference in the probability that multi-sector deadlock occurs. By increasing the number of vehicles that is required to form a cross-border circular wait, this probability can be significantly reduced. Additionally, by allowing entrance tiles to only have one incoming edge, which is the edge coming from the corresponding exit tile, reserving an entrance tile can never directly lead to a deadlock because reserving an entrance tile can never lead to a circular wait.

6.2 Recommendation

During the development of the hierarchical control strategy, several problems arose and a couple of features were considered out of the scope of the project. In this section, possible future improvements are introduced and recommendations are given based on which further research on the feasibility of the hierarchical control strategy can be performed.

Multi-sector deadlock

As was discovered in Chapter 5, multi-sector deadlock is a crucial problem that can arise when using the hierarchical control strategy. However, it was also discovered that whether multi-sector deadlock occurs strongly depends on the layout design, sector partitioning and vehicle density of a system. Therefore, additional research has to be performed on how multi-sector deadlock can be dealt with.

First of all, an estimate must be made on how the probability that multi-sector deadlock occurs depends on the vehicle density in the system and the partitioning of the layout. As was shown for the advanced layout in Section 5.3, the probability that deadlock occurs in a critical region can significantly be reduced by limiting the vehicle density in the system. The upside is that in most scenarios, the peak throughput is already reached at a lower vehicle density than where multi-sector deadlock occurs. This is, however, not the case for the layout in Figure 5.8. So, limiting the vehicle density is no guarantee because there can always be a small group of vehicles that just happens to end up in multi-sector deadlock, and as soon as a small group of vehicles has ended up in deadlock, it is usually a matter of time before these vehicles start blocking other vehicles. However, the probability that multi-sector deadlock occurs can be further reduced by using a sector partitioning where critical regions are not split over multiple sectors, such that one controller always has full information over all tiles in a critical region. The question that remains is *how big is the probability that a multi-sector deadlock occurs?*

Additionally, before a local controller approves a sector transition request it could also check whether reserving the requested entrance tile will lead to a deadlock. Currently, the local controller only checks whether the entrance tile is already reserved. By doing this, it is at least ensured that mono-sector deadlock does not occur after the vehicle transitions from exit to entrance tile. However, this does not mean that multi-sector cannot result afterwards, because another vehicle might reserve the exit tile that was just released which can still result in multi-sector deadlock, as was explained using Figure 5.15.

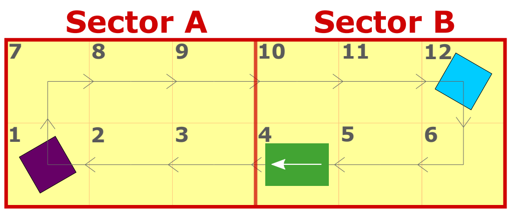
Finally, a form of multi-sector deadlock avoidance and/or deadlock detection and recovery can be implemented. If multi-sector deadlocks only occur occasionally, periodically checking whether the system finds itself in a multi-sector deadlock situation and recovering from this situation might be sufficient. On the other hand, if multi-sector deadlock occurs more often, it might be necessary to perform a multi-sector deadlock avoidance check. However, because deadlock avoidance algorithms are generally relatively computationally expensive, it can be expected that such a multi-sector deadlock avoidance algorithm is extremely expensive as well. Furthermore, it raises questions such as *what does a multi-sector deadlock avoidance and detection algorithm look like? how can multi-sector deadlocks be recovered?* and *who is responsible for initiating a deadlock check?*

Requesting sector transition ahead of time

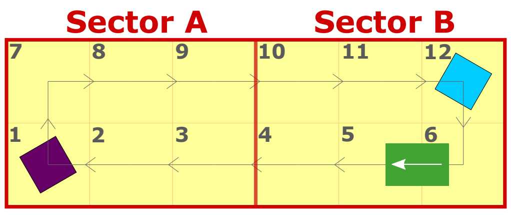
In Section 5.1, 5.2 and 5.3 it was computed that for the simple layout, intermediate layout and advanced layout, the increase in average item lead time when using one vehicle was caused for approximately 65.8, 75.8 and 52.2 [%] by vehicles having to decelerate, come to a stop, and accelerate when crossing the border. This does not necessarily mean this is the case for every layout or every vehicle density, but it does show that this *can* significantly reduce the performance of the hierarchical control strategy.

This problem can be resolved by sending a sector transition request ahead of time to prevent vehicles from having to slow down. The difference between the current moment the sector transition request is sent and when the sector transition request is sent ahead of time is shown in Figure 6.1. One way in which this can be achieved is by sending a sector transition request when the distance needed to come to a stop threatens to become smaller than the distance towards the center of the exit tile. Communication delays between controllers can be taken into account by including a small margin. A similar mechanism is currently already used to provide vehicles with waypoints just in time. Alternatively, a sector transition request can be sent when the exit tile has been *reserved*. Each local controller typically tries to reserve several tiles ahead for their vehicles. Because a vehicle must always be able to come to a stop on a tile that is reserved for itself, the distance (i.e. number of tiles) that a controller must reserve ahead is equal to or larger than the distance required to come to a stop. If an exit tile is reserved for a vehicle, it is guaranteed that all tiles leading up to this exit tile are also reserved for the same vehicle. Assuming that the exit tile is used to leave the sector and not just as part of a path through the sector, the next tile in the path of said vehicle must always be the entrance tile. Regardless of the method that is used, if a sector transition request is not approved in time, the vehicle still has to decelerate and possibly come to a stop.

Similarly, in the next sector enough tiles need to be reserved ahead to ensure that a vehicle does not have to decelerate, come to a stop and accelerate as soon as it arrives at the entrance tile. Therefore, the receiving controller must also reserve enough tiles to at least cover the distance required to come to a stop. However, tiles can only be reserved if a local path for this sector has already been determined. Consequently, it is necessary to plan ahead the local path of the next sector upon approving a sector transition request. In order to do so, the preceding sector transition request must contain sufficient information, such as the next tile in the global path of the vehicle and the sector of this tile. The receiving controller can include the computed local path when responding with the sector transition approval message.



(a) Sector transition request is sent upon reaching the center of the exit tile.



(b) Sector transition request is sent ahead to prevent the vehicle from decelerating.

Figure 6.1: Two different moments at which a sector transition request can be sent for a vehicle transitioning from sector B to sector A.

Basing the global path on congestion

In Section 2.2.1 it was introduced that congestion can significantly damage the performance of a system, as a result of vehicles spending time waiting in order to avoid collisions, deadlocks and livelocks. Though local controllers are able to take into account congestion on a local level following the path planning algorithm introduced in [Fransen et al., 2020], congestion is *not* taken into account when planning global paths.

In order to take into account congestion when planning global paths, [Liu et al., 2019] introduced a sector-level heat map. This heat map aims to identify congested areas and penalizes sectors in which a high level of congestion is present. In an attempt to identify congested areas, heat values can for example be based on the average speed of all vehicles in a sector, the total waiting time of all vehicles in a sector, or the fraction of drivable tiles in a sector that is occupied.

For the layouts that are simulated in Chapter 5 penalizing congested areas might not make a lot of difference since these layouts do not contain a lot of sectors. For larger layouts, however, this could prove to make a more significant difference.

Periodic rerouting of vehicles on a global scale

Besides planning global paths based on congestion, an additional manner in which congestion can be considered is to periodically reroute vehicles on a global scale. In the current hierarchical control model, vehicles receive a global path from POI to POI only once. Determining a global path based on a heat value might take into account congestion *at the moment of planning the path*, but it is not guaranteed that congestion in the layout does not change over time. A sector that has a lot of congestion at one moment might be entirely empty a few moments later. Therefore, it might be useful to periodically replan a vehicle's global path to verify whether the current global path is still the best possible global path.

Furthermore, it is worth investigating whether periodically rerouting vehicles on a global scale is (part of) the solution to multi-sector deadlock. As an example, 10 simulations were run on the intermediate layout of Section 5.2 with a vehicle density of 30.61 [%], i.e. 15 vehicles. In 80 [%] of the simulations, the system ended up in a multi-sector deadlock. A snapshot of the final state of the system of one of these simulations can be seen in Figure 6.2. Neither of the involved local controllers is able to detect this multi-sector deadlock because both local controllers only have access to local information and cannot find a circular wait. Moreover, periodic rerouting on a local scale cannot resolve this multi-sector deadlock because there do not exist any alternative paths in any of the sectors. On the other hand, if the global path of the top left or bottom left vehicle in the chain is replanned, this multi-sector deadlock situation is resolved, at least for now.

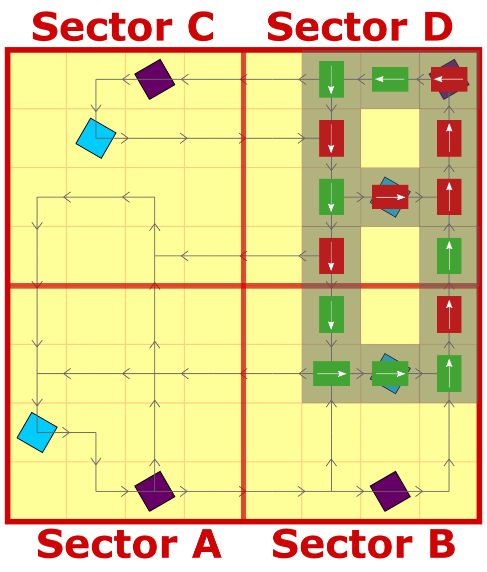


Figure 6.2: Multi-sector deadlock situation when using 15 vehicles on the intermediate layout.

Increasing freedom in the global path

Besides periodically replanning global paths, additional freedom can be added to the global path such that local controllers can choose from multiple exit tiles. For example, if a local controller has multiple exit tiles to choose from that lead to the same sector, it can select the exit tile whose expected travel time is the shortest. As an example, in Figure 6.3 local controller A can choose to plan a local path to exit tile 4 or 28. Upon arrival in sector B, local controller sector B can choose to plan a local path to exit tile 8 or 32, regardless of the entrance tile through which the vehicle entered the sector. In sector C, local controller C just plans a local path to the pick-up station on tile 36. In this way, congested areas or possibly deadlocks can be avoided. How exactly such a global path is determined and defined is something to be studied.

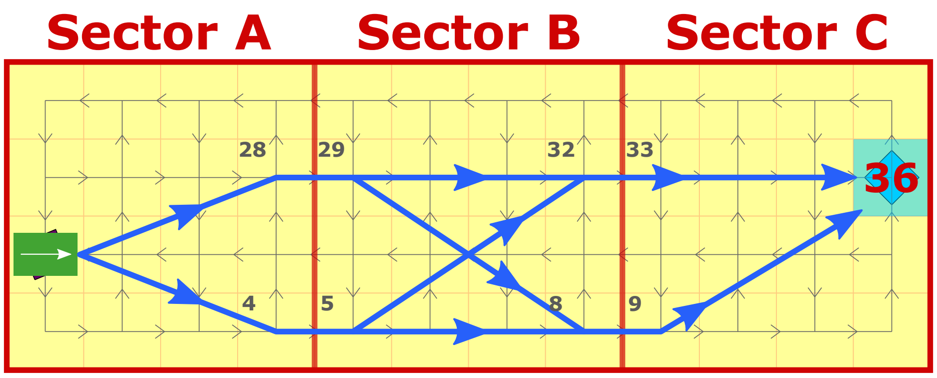


Figure 6.3: Example of a global path where multiple exit tiles are allowed.

This idea can also be used to periodically replan a local path in a sector that contains a lot of congestion or deadlock. However, one has to be careful that the remainder of the global paths always remains feasible. Consider the layout in Figure 6.4, which finds itself in a deadlock situation. The vehicle indicated in red in has to pick up a load at the pick-up station in sector C, but the path to exit tile 46 is blocked by the deadlock. Locally replanning the path to exit tile 46 is not possible, because there is no alternative path to this tile. Exit tile 28 also leads to sector C, but if this tile is selected as an alternative exit tile and a local path is planned, it is no longer possible to reach the pick-up station upon arriving in sector C. In other words, even if an alternative exit tile leads to the same sector, it is no guarantee that the remainder of the path is feasible. Therefore, it must always be checked whether using an alternative exit tile does not lead to an infeasibility problem.

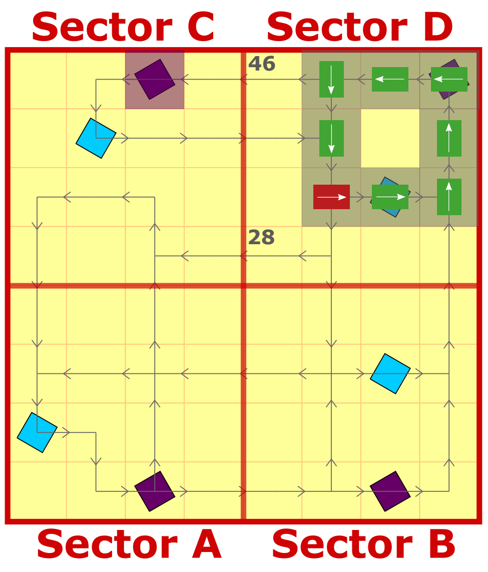


Figure 6.4: If the red-shaded vehicle exits sector D towards sector C through exit tile 28 it is not possible to reach the pick-up station in sector C.

Increasing the global controller cycle interval
Next, more research has to be performed on the consequence of increasing the interval between each global control cycle. As was explained in Section 4.4, during the global control cycle the global controller does two things: it forwards vehicle updates to the correct local controller and it determines a global path when a global path has been requested.

In the current implementation of the hierarchical control strategy, new global paths are only requested and assigned twice. Therefore, increasing the global controller cycle interval is expected to make a modest amount of difference with regard to determining global paths. If a vehicle has to wait for a new global path for a longer period of time, all it means is that the vehicle occupies its tile for an unnecessary amount of time, possibly blocking other vehicles while doing so.

An increased global controller cycle interval might be more detrimental to the system's performance with regard to forwarding vehicle updates. If the interval is increased, local controllers receive vehicle updates with a significant delay, meaning that new paths and targets are determined with a delay, tile reservations are released with a delay, and new tiles are reserved with a delay. What this means for the performance of the system still has to be studied. As was mentioned in Section 4.3.3, an alternative to vehicles sending their vehicle updates to the correct local controller via the global controller is vehicles sending their vehicle updates to all local controllers. Every local controller can then filter out the vehicle updates that correspond to the vehicles that are currently located in its sector, and disregard all other vehicle updates. In this way, the process of sending vehicle updates can be decoupled from the global controller. However, this does come at the cost of unnecessarily stressing the communication network, which is something that is preferably kept to a minimum.

Paths to parking spots
As was briefly mentioned in Section 4.5, the functionality to plan paths to parking spots has not yet been implemented in the hierarchical control model, as opposed to the supervisory control model, due to time limitations. This currently leads to a situation where jobless vehicles stand still in the middle of the layout until they are assigned a new job, while possibly blocking other vehicles. Even though it was assumed in Section 3.1.2 that there are always available jobs in the system, it is recommended that this functionality is implemented in case this assumption does not hold.

When implementing this functionality a rough distinction between two types of approaches can be made: either the local controllers are responsible for planning a path to a parking spot, or the global controller. An example of each of the types of approaches is:

1. **Local controllers are responsible for planning paths to parking spots:** Because local controllers only have access to local information, they can only plan a local path to a parking spot that is located in their own sector. If a sector has no parking spots or all parking spots are taken, no path can be planned and the vehicle keeps standing still, possibly blocking other vehicles.

2. **The global controller is responsible for planning paths to parking spots:** Whenever a vehicle has no job, the local controller of the sector in which the vehicle is located sends a global path request message to the global controller, where the global target is one of the parking spots. The global controller responds using a global path response message.

References

[Blum and Roli, 2003] Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35:268–308.

[Bnaya et al., 2013] Bnaya, Z., Stern, R., Felner, A., Zivan, R., and Okamoto, S. (2013). Multi-agent path finding for self interested agents. In *Proceedings of the 6th Annual Symposium on Combinatorial Search, SoCS 2013*.

[Cáp et al., 2015] Cáp, M., Vokrínek, J., and Kleiner, A. (2015). Complete decentralized method for online multi-robot trajectory planning in valid infrastructures. *CoRR*, abs/1501.07704.

[De Ryck et al., 2020] De Ryck, M., Versteyhe, M., and Debrouwere, F. (2020). Automated guided vehicle systems, state-of-the-art control algorithms and techniques. *Journal of Manufacturing Systems*, 54:152–173.

[Digani et al., 2014] Digani, V., Sabattini, L., Secchi, C., and Fantuzzi, C. (2014). Hierarchical traffic control for partially decentralized coordination of multi AGV systems in industrial environments. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6144–6149.

[Dijkstra, 1959] Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271.

[Fanti et al., 2015] Fanti, M. P., Mangini, A., Pedroncelli, G., and Ukovich, W. (2015). Decentralized deadlock-free control for AGV systems. In *2015 American Control Conference*, pages 2414–2419.

[Fanti et al., 2018] Fanti, M. P., Mangini, A. M., Pedroncelli, G., and Ukovich, W. (2018). A decentralized control strategy for the coordination of AGV systems. *Control Engineering Practice*, 70:86–97.

[Fazlollahtabar and Saidi-Mehrabad, 2013] Fazlollahtabar, H. and Saidi-Mehrabad, M. (2013). Methodologies to optimize automated guided vehicle scheduling and routing problems: A review study. *Journal of Intelligent & Robotic Systems*, 77:525–545.

[Fragapane et al., 2021] Fragapane, G., de Koster, R., Sgarbossa, F., and Strandhagen, J. O. (2021). Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda. *European Journal of Operational Research*, 294:405–426.

[Fransen, 2019] Fransen, K. (2019). A path planning approach for AGVs in the dense grid-based AGV sorter. Master's thesis, Eindhoven University of Technology.

[Fransen et al., 2022] Fransen, K., Reniers, M., and van Eekelen, J. (2022). Deadlock avoidance algorithm for AGVs on a tessellated layout. *IEEE International Conference on Automation Science and Engineering.*

[Fransen et al., 2020] Fransen, K., van Eekelen, J., Pogromsky, A., Boon, M., and Adan, I. (2020). A dynamic path planning approach for dense, large, grid-based automated guided vehicle systems. *Computers & Operations Research*, 123:105046.

[Hart et al., 1968] Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100–107.

[Jacobs, 2020] Jacobs, J. (2020). The design and implementation of a job assignment strategy in automated guided vehicle systems. Master's thesis, Eindhoven University of Technology.

[Jäger and Nebel, 2001] Jäger, M. and Nebel, B. (2001). Decentralized collision avoidance, deadlock detection, and deadlock resolution for multiple mobile robots. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium*, volume 3, pages 1213–1219.

[Kim et al., 2018] Kim, D., Hai, N., and Joe, W. (2018). A guide to selecting path planning algorithm for automated guided vehicle (AGV). In *Duy V., Dao T., Zelinka I., Kim S., Phuong T. (eds) AETA 2017 - Recent Advances in Electrical Engineering and Related Sciences: Theory and Application. AETA 2017. Lecture Notes in Electrical Engineering*, pages 587–596. Springer International Publishing, Cham.

[Koenig and Likhachev, 2002] Koenig, S. and Likhachev, M. (2002). D* lite. In *Proceedings of the National Conference on Artificial Intelligence*, pages 476–483.

[Krnjak et al., 2015] Krnjak, A., Draganjac, I., Bogdan, S., Petrović, T., Miklić, D., and Kovačić, Z. (2015). Decentralized control of free ranging AGVs in warehouse environments. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2034–2041.

[Li et al., 2012] Li, Z., Wu, N., and Zhou, M. (2012). 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)*, 42:437–462.

[Liu et al., 2019] Liu, Z., Wang, H., Zhou, S., Shen, Y., and Liu, Y. (2019). Coordinating large-scale robot networks with motion and communication uncertainties for logistics applications. *CoRR*, abs/1904.01303.

[Malacad, 2022] Malacad, M. (2022). Comparing Dijkstra's and A* search algorithm. https://medium.com/@miguell.m/dijkstras-and-a-search-algorithm-2e67029d7749. [Online: Accessed 27 September 2022].

[Małopolski, 2018] Małopolski, W. (2018). A sustainable and conflict-free operation of AGVs in a square topology. *Computers & Industrial Engineering*, 126:472–481.

[Moorthy et al., 2003] Moorthy, R., Hock-Guan, W., Wing-Cheong, N., and Chung-Piaw, T. (2003). Cyclic deadlock prediction and avoidance for zone-controlled AGV system. *International Journal of Production Economics*, 83:309–324.

[Osman and Laporte, 1996] Osman, I. H. and Laporte, G. (1996). Metaheuristics: A bibliography. *Annals of Operations Research*, 63:513–623.

[Robinson, 2004] Robinson, S. (2004). Simulation: the practice of model development and use. *John Wiley & Sons*, 1.

[Sabattini et al., 2016] Sabattini, L., Digani, V., Secchi, C., and Fantuzzi, C. (2016). Hierarchical coordination strategy for multi-AGV systems based on dynamic geodesic environment partitioning. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4418–4423.

[Sharon et al., 2015] Sharon, G., Stern, R., Felner, A., and athan R. Sturtevant, N. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66.

[Sieniutycz, 2020] Sieniutycz, S. (2020). Systems design: Modeling, analysis, synthesis, and optimization. *Complexity and Complex Thermo-Economic Systems*, pages 85–115.

[Somers, 2022] Somers, J. (2022). Development of a deadlock recovery algorithm for grid-based AGV systems. Master's thesis, Eindhoven University of Technology.

[Stentz, 1994] Stentz, A. (1994). Optimal and efficient path planning for partially-known environments. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, volume 4, pages 3310–3317.

[Stentz, 1995] Stentz, A. (1995). The focussed D* Algorithm for real-time replanning. In *Proceedings of 14th International Joint Conference on Artificial Intelligence*, volume 2, page 1652 – 1659.

[Vader, 2022] Vader, R. M. (2022). Analysis, design and implementation of a new routing controller for a grid-based AGV system. Graduation preparation report, Eindhoven University of Technology.

[van Weert, 2019] van Weert, M. (2019). Deadlock avoidance and detection for the grid-based AGV-sorter system. Master's thesis, Eindhoven University of Technology.

[Vanderlande, 2022] Vanderlande (2022). Fleet. https://www.vanderlande.com/evolutions/fleet/. [Online: Accessed 3 February 2022].

[Vis, 2006] Vis, I. F. (2006). Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research*, 170:677–709.

[Xiao et al., 2020] Xiao, H., Wu, X., Qin, D., and Zhai, J. (2020). A collision and deadlock prevention method with traffic sequence optimization strategy for UGN-based AGVs. *IEEE Access*, 8:209452–209470.

[Yeh and Yeh, 1998] Yeh, M.-S. and Yeh, W.-C. (1998). Deadlock prediction and avoidance for zone-control AGVs. *International Journal of Production Research*, 36:2879–2889.

[Yuan et al., 2016] Yuan, R., Dong, T., and Li, J. (2016). Research on the collision-free path planning of multi-AGVs system based on improved A* algorithm. *American Journal of Operations Research*, 6:442–449.

A Appendix A

This appendix contains the sequence diagrams of the communication processes in the grid-based control model.

A.1 Vehicle updates from vehicles to controller

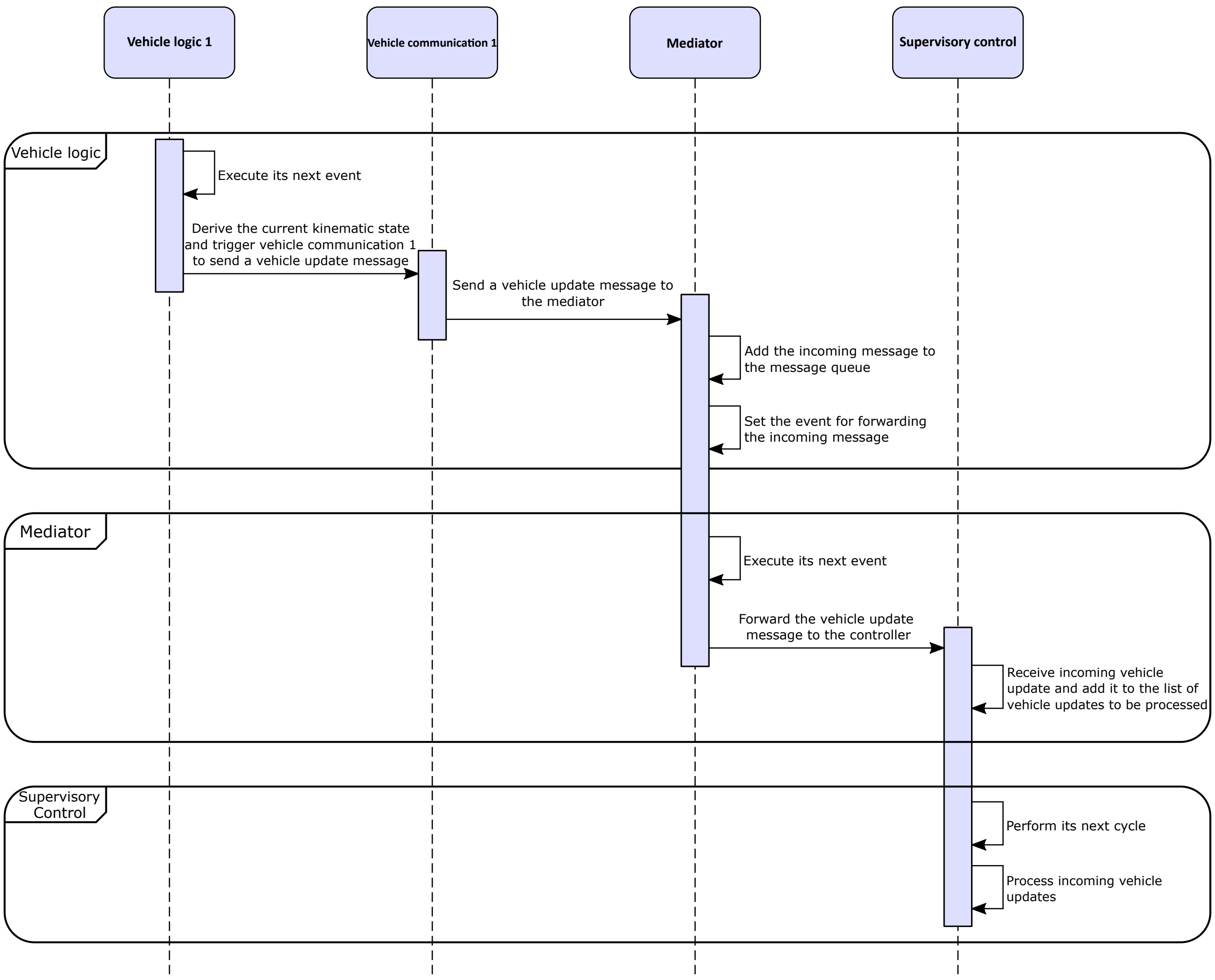

Vehicle logic 1
Vehicle communication 1
Mediator
Supervisory control
Vehicle logic
Execute its next event
Derive the current kinematic state and trigger vehicle communication 1 to send a vehicle update message
Send a vehicle update message to the mediator
Add the incoming message to the message queue
Set the event for forwarding the incoming message
Mediator
Execute its next event
Forward the vehicle update message to the controller
Receive incoming vehicle update and add it to the list of vehicle updates to be processed
Supervisory Control
Perform its next cycle
Process incoming vehicle updates

Figure A.1: Sequence diagram corresponding to the communication process from vehicles to the controller in the grid-based control model.

A.2 Vehicle commands from controller to vehicles

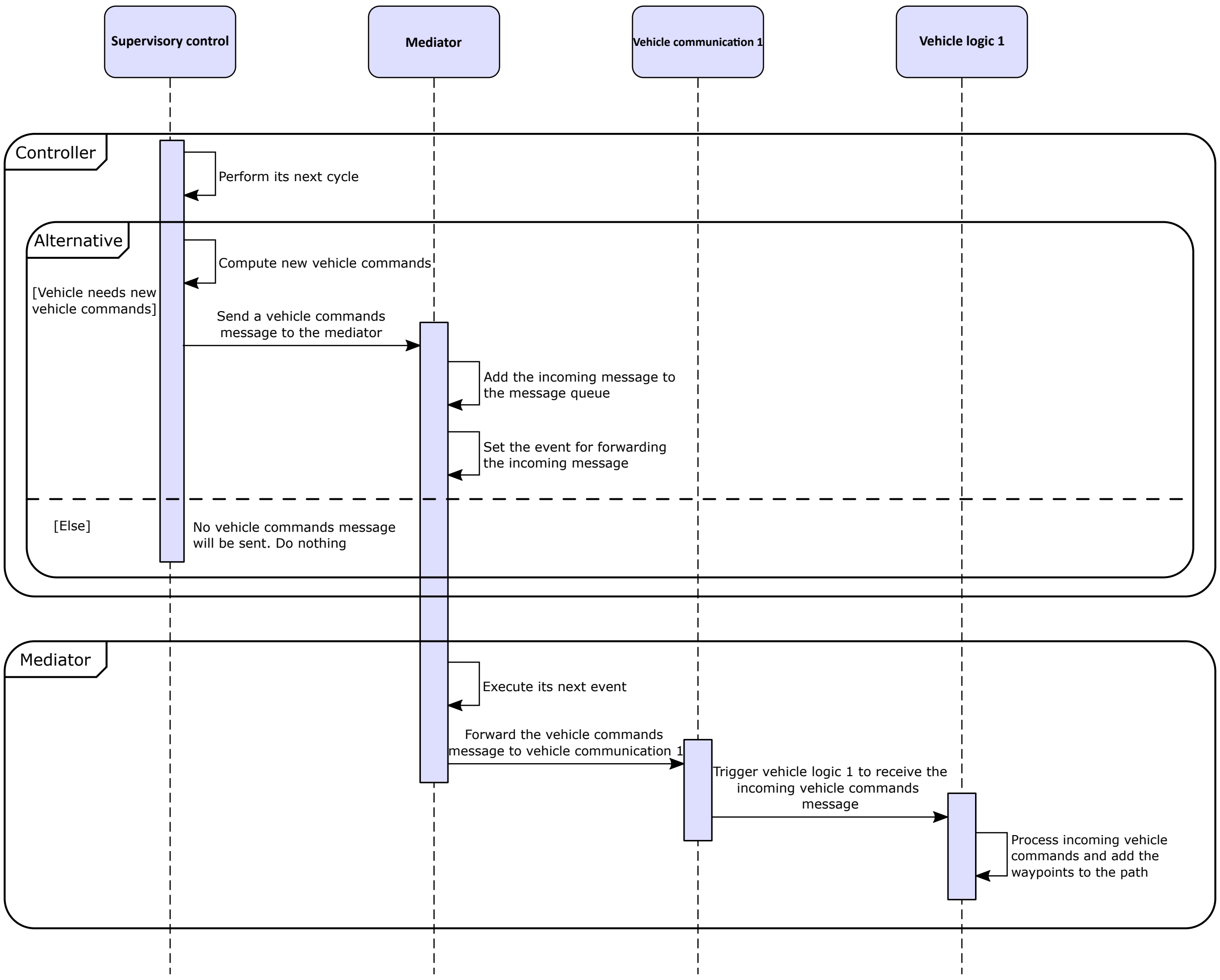

Supervisory control
Mediator
Vehicle communication 1
Vehicle logic 1
Controller
Perform its next cycle
Alternative
Compute new vehicle commands
[Vehicle needs new vehicle commands]
Send a vehicle commands message to the mediator
Add the incoming message to the message queue
Set the event for forwarding the incoming message
[Else]
No vehicle commands message will be sent. Do nothing
Mediator
Execute its next event
Forward the vehicle commands message to vehicle communication 1
Trigger vehicle logic 1 to receive the incoming vehicle commands message
Process incoming vehicle commands and add the waypoints to the path

Figure A.2: Sequence diagram corresponding to the communication process from the controller to the vehicles in the grid-based control model.

B Appendix B

This appendix contains the sequence diagrams of the communication processes in the hierarchical control model.

B.1 Vehicle updates from vehicles to local controllers

The vehicle logic of each vehicle is responsible for initiating vehicle update messages from the vehicle to their local controller. However, as was stressed in Section 4.3.3, vehicles do not know to which controller their vehicle updates must be sent because they do not know in which sector they are located. As a solution to this problem, the global controller acts as an intermediary who forwards the incoming vehicle updates to the correct local controller. Communication from a vehicle, via the global controller, to the correct local controller can be described using the following steps:

1. Vehicle logic triggers vehicle communication to send a vehicle update to the mediator containing the vehicle's most recent kinematic state. Vehicle communication forwards the vehicle update message to the mediator.
2. The mediator adds the incoming message to its message queue. The time of execution of this message is equal to the current time unless a delay is applied to the message. We assume that no delay is applied to the message, implying that the timestamp to execute the message is the same as the current time.
3. If there are no other messages in the message queue or if all other messages in the message queue have a larger time to execute than the new message, the next event of the mediator is set to forward the incoming vehicle update message. If there are other messages in the message queue with a lower or identical time to execute, these messages are forwarded first. In this case, the next event of the mediator is not adjusted. This is explained in more detail in Section 3.1.2.
4. As soon as the time of execution of the vehicle update message has been reached in the simulation and the message is in front of the message queue, the mediator forwards the vehicle update message to the *global controller*.
5. The global controller receives the incoming message. It adds the incoming vehicle update message to the list of incoming vehicle updates that still have to be forwarded by the global controller in its next global control cycle.
6. In the next cycle of the controller, all vehicle updates that are in the list of vehicle updates that still need to be processed are processed. The global controller determines for every vehicle update in which sector the corresponding vehicle is currently located. For every vehicle update, the global controller sends the vehicle update delayed message to the mediator.
7. Repeat steps 2 and 3.
8. As soon as the time of execution of the vehicle delayed update message has been reached and the message is at the front of the message queue, the mediator forwards the vehicle update delayed message to the local controller of the sector in which the vehicle is currently located.
9. The local controller receives the incoming message. It adds the incoming vehicle update to the list of incoming vehicle updates that still have to be processed, such that the vehicle update can be processed during the next cycle of the *local controller*.
10. In the next cycle of the local controller, all vehicle updates that are in the list of vehicle updates that still need to be processed are processed by the local controller.

The sequence diagram corresponding to the communication process from the vehicles to the local controllers in the hierarchical control model can be found in Figure B.1.

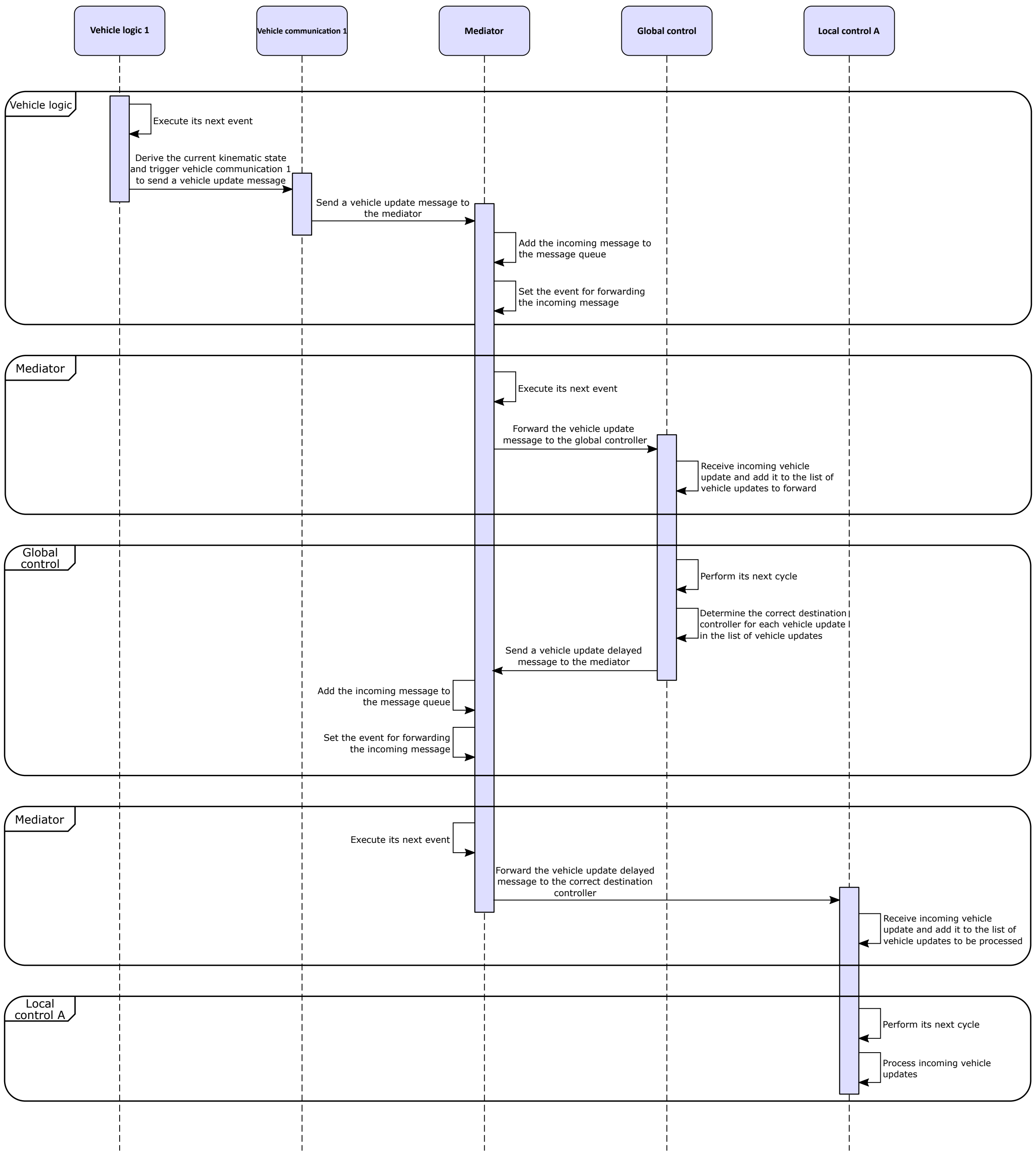

Vehicle logic 1
Vehicle communication 1
Mediator
Global control
Local control A
Vehicle logic
Execute its next event
Derive the current kinematic state and trigger vehicle communication 1 to send a vehicle update message
Send a vehicle update message to the mediator
Add the incoming message to the message queue
Set the event for forwarding the incoming message
Mediator
Execute its next event
Forward the vehicle update message to the global controller
Receive incoming vehicle update and add it to the list of vehicle updates to forward
Global control
Perform its next cycle
Determine the correct destination controller for each vehicle update in the list of vehicle updates
Send a vehicle update delayed message to the mediator
Add the incoming message to the message queue
Set the event for forwarding the incoming message
Mediator
Execute its next event
Forward the vehicle update delayed message to the correct destination controller
Receive incoming vehicle update and add it to the list of vehicle updates to be processed
Local control A
Perform its next cycle
Process incoming vehicle updates

Figure B.1: Sequence diagram corresponding to the communication process from vehicles to the local controllers in the hierarchical control model.

B.2 Vehicle commands from local controllers to vehicles

During the performance of its job, a vehicle follows waypoints that are provided by its local controller. These waypoints are sent just in time, such that a vehicle does not have to decelerate if possible. As a result of this, the local controller continuously has to check whether each vehicle requires new waypoints. If a vehicle requires new waypoints, the controller sends these waypoints to the mediator using a vehicle commands message, which forwards the message to the correct vehicle. The communication process of sending a vehicle commands message from the controller to a vehicle can be described using the following steps:

1. The local controller performs its local control cycle and computes new waypoints for a particular vehicle. These waypoints are sent from the controller to the mediator using a vehicle commands message.

2. The mediator adds the incoming vehicle commands message to its message queue. The time of execution of this message is equal to the current time unless a delay is applied to the message. We assume that no delay is applied to the message, implying that the timestamp to execute the message is the same as the current time.

3. If there are no other messages in the message queue or if all other messages in the message queue have a larger time to execute than the new message, the next event of the mediator is set to forward the incoming vehicle commands message. If there are other messages in the message queue with a lower or identical time to execute, these messages are forwarded first. In this case, the next event of the mediator is not adjusted. This is explained in more detail in Section 3.1.2.

4. As soon as the time of execution of the vehicle commands message has been reached and the message is at the front of the message queue, the mediator forwards the message to the vehicle communication of the correct vehicle.

5. Vehicle communication receives the incoming message and triggers vehicle logic to receive the incoming vehicle commands message. Vehicle logic immediately processes the new waypoints and appends them to the list of waypoints that it has received before.

The sequence diagram corresponding to the communication process from the local controller to the vehicles in the hierarchical control model can be found in Figure B.2.

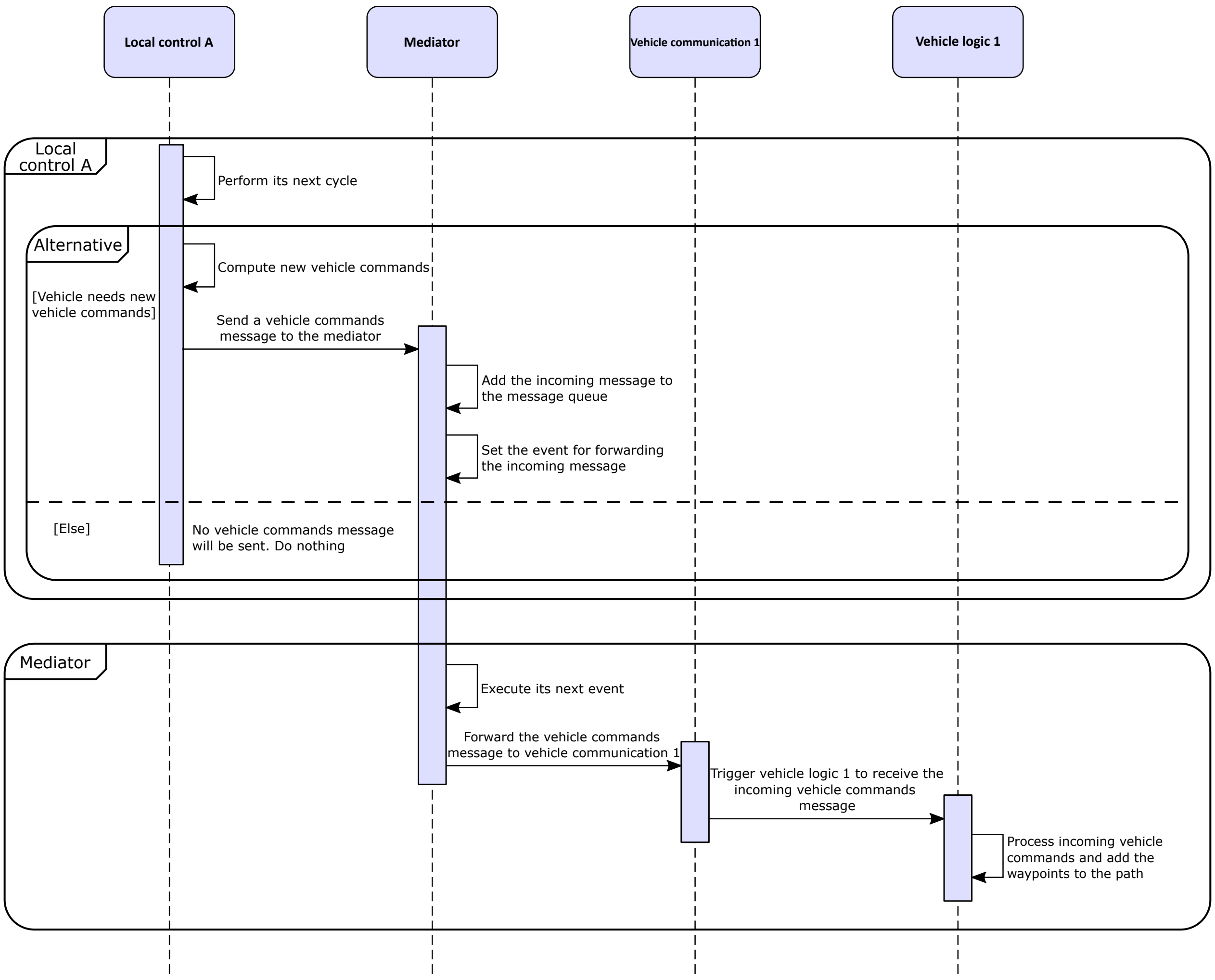

Local control A
Mediator
Vehicle communication 1
Vehicle logic 1
Local control A
Perform its next cycle
Alternative
Compute new vehicle commands
[Vehicle needs new vehicle commands]
Send a vehicle commands message to the mediator
Add the incoming message to the message queue
Set the event for forwarding the incoming message
[Else]
No vehicle commands message will be sent. Do nothing
Mediator
Execute its next event
Forward the vehicle commands message to vehicle communication 1
Trigger vehicle logic 1 to receive the incoming vehicle commands message
Process incoming vehicle commands and add the waypoints to the path

Figure B.2: Sequence diagram corresponding to the communication process from the local controllers to the vehicles in the hierarchical control model.

B.3 Assigning global paths to vehicle agents

During every local control cycle, each local controller checks for all its vehicle agents whether each agent has a job but no global target. If an agent has a job but no global target, a global target is selected and a global path request is sent from the local controller to the global controller. In the global control cycle, the global controller determines a global path from the vehicle's latest reserved tile to the global target and sends a global path response message back to the local controller. Then, during the next local control cycle of this particular local controller, all incoming global path responses are processed. Communication between a local controller and the global controller regarding global paths can be described using the following steps:

1. The local controller performs its local control cycle. A vehicle agent has a job but no global target and is not already waiting for a global path. A new global target is selected and a global path request message is sent to the mediator.
2. The mediator adds the incoming message to its message queue. The time of execution of this message is equal to the current time unless a delay is applied to the message. We assume that no delay is applied to the message, implying that the timestamp to execute the message is the same as the current time.
3. If there are no other messages in the message queue or if all other messages in the message queue have a larger time to execute than the new message, the next event of the mediator is set to forward the incoming message. If there are other messages in the message queue with a lower or identical time to execute, these messages are forwarded first. In this case, the next event of the mediator is not adjusted. This is explained in more detail in Section 3.1.2.
4. As soon as the time of execution of the global path request message has been reached in the simulation and the message is in front of the message queue, the mediator forwards the global path request message to the global controller.
5. The global controller receives the incoming message and adds the incoming global path request message to the list of global path requests that still have to be processed by the global controller in its next global control cycle.
6. In the next cycle of the global controller, all global path requests that are in the list of global path requests that still need to be processed are processed. The global controller determines for every global path request the shortest path from the last reserved tile of the vehicle to the global target using Dijkstra's algorithm. The global controller sends a global path response message to the mediator containing the global path.
7. Repeat steps 2 and 3.
8. As soon as the time of execution of the global path response message has been reached and the message is at the front of the message queue, the mediator forwards the global path response message to the local controller that requested the global path.
9. The local controller receives the incoming message. It adds the incoming global path response message to the list of incoming global path responses, such that the global path response can be processed during the next cycle of the local controller.
10. In the next cycle of the local controller, all global path responses that are in the list of global path responses that still need to be processed are processed by the local controller.

The sequence diagram corresponding to the communication process between the local and global controller when assigning global paths to vehicle agents in the hierarchical control model can be found in Figure B.3.

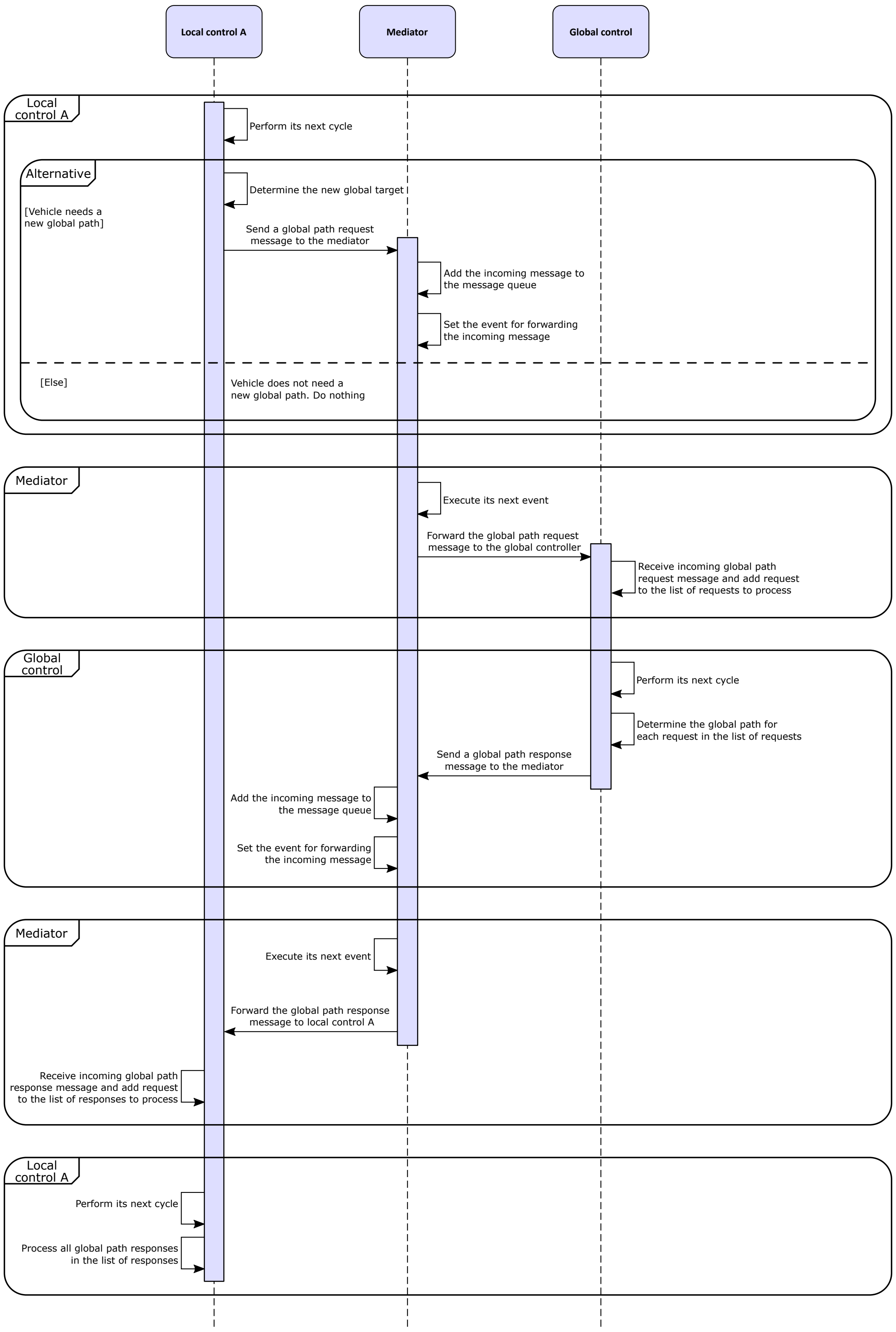

Local control A
Mediator
Global control
Local control A
Perform its next cycle
Alternative
Determine the new global target
[Vehicle needs a new global path]
Send a global path request message to the mediator
Add the incoming message to the message queue
Set the event for forwarding the incoming message
[Else]
Vehicle does not need a new global path. Do nothing
Mediator
Execute its next event
Forward the global path request message to the global controller
Receive incoming global path request message and add request to the list of requests to process
Global control
Perform its next cycle
Determine the global path for each request in the list of requests
Send a global path response message to the mediator
Add the incoming message to the message queue
Set the event for forwarding the incoming message
Mediator
Execute its next event
Forward the global path response message to local control A
Receive incoming global path response message and add request to the list of responses to process
Local control A
Perform its next cycle
Process all global path responses in the list of responses

Figure B.3: Sequence diagram corresponding to the communication process between the local and global controller when assigning global paths in the hierarchical control

B.4 Vehicle transitions between local controllers

1. The local controller performs its local control cycle. A vehicle agent has a global target but no local target, the agent is not already waiting for sector transition approval, and the current sector of the vehicle is different from the sector in which the next tile of the global path is located. The local controller sends a sector transition request message to the mediator.

2. The mediator adds the incoming message to its message queue. The time of execution of this message is equal to the current time unless a delay is applied to the message. We assume that no delay is applied to the message, implying that the timestamp to execute the message is the same as the current time.

3. If there are no other messages in the message queue or if all other messages in the message queue have a larger time to execute than the new message, the next event of the mediator is set to forward the incoming message. If there are other messages in the message queue with a lower or identical time to execute, these messages are forwarded first. In this case, the next event of the mediator is not adjusted. This is explained in more detail in Section 3.1.2.

4. As soon as the time of execution of the sector transition request message has been reached in the simulation and the message is in front of the message queue, the mediator forwards the sector transition request message to the local controller in which the entrance tile is located, i.e. the receiving controller.

5. The receiving controller receives the incoming message and adds the incoming sector transition request message to the list of sector transition requests that still have to be approved by the receiving controller.

6. In the next cycle of the receiving controller, all sector transition requests that have not yet been approved are evaluated. If the requested entrance tile is available and the requesting agent has priority, the sector transition request is approved. The receiving controller sends a sector transition approval message to the mediator.

7. Repeat steps 2 and 3.

8. As soon as the time of execution of the sector transition approval message has been reached in the simulation and the message is in front of the message queue, the mediator forwards the sector transition approval message to the local controller in which the exit tile is located, i.e. the sending controller.

9. The sending controller receives the incoming message and adds the incoming sector transition approval message to the list of sector transition approvals that still have to be processed by the sending controller.

10. In the next cycle of the sending controller, all sector transition approvals that have not yet been processed are processed. The vehicle agent corresponding to the approval computes the waypoints for transitioning from the exit tile to the entrance tile. The sending controller sends a vehicle commands message to the mediator.

11. Repeat steps 2 and 3.

12. As soon as the time of execution of the vehicle commands message has been reached and the message is at the front of the message queue, the mediator forwards the message to the vehicle communication of the correct vehicle.

13. Vehicle communication receives the incoming message and triggers vehicle logic to receive the incoming vehicle commands message. Vehicle logic immediately processes the new waypoints that are required to transition from the exit tile to the entrance tile and appends them to the list of waypoints that it has received before.

14. In the next cycle of the sending controller, all sector transition approvals that have not yet been processed are processed.

15. In the successive cycles of the sending controller it checks whether the vehicle has physically left the exit tile. As soon as the vehicle has left the exit tile, it must have also physically entered the entrance tile. The sending controller sends a vehicle registration message to the mediator.

16. Repeat steps 2 and 3.

17. If the sending controller has just sent a vehicle registration message to the mediator it also sends a controller update message to the mediator.

18. Repeat steps 2 and 3.

19. As soon as the time of execution of the vehicle registration message has been reached and the message is at the front of the message queue, the mediator forwards the message to the receiving controller.

20. The receiving controller receives the incoming message and registers the new agent to its administration.

21. As soon as the time of execution of the controller update message has been reached and the message is at the front of the message queue, the mediator forwards the message to the global controller.

22. The receiving controller receives the incoming message and updates the global administration.

The sequence diagram corresponding to the communication process during vehicle sector transitioning from one sector to another in the hierarchical control model can be found in Figure B.4.

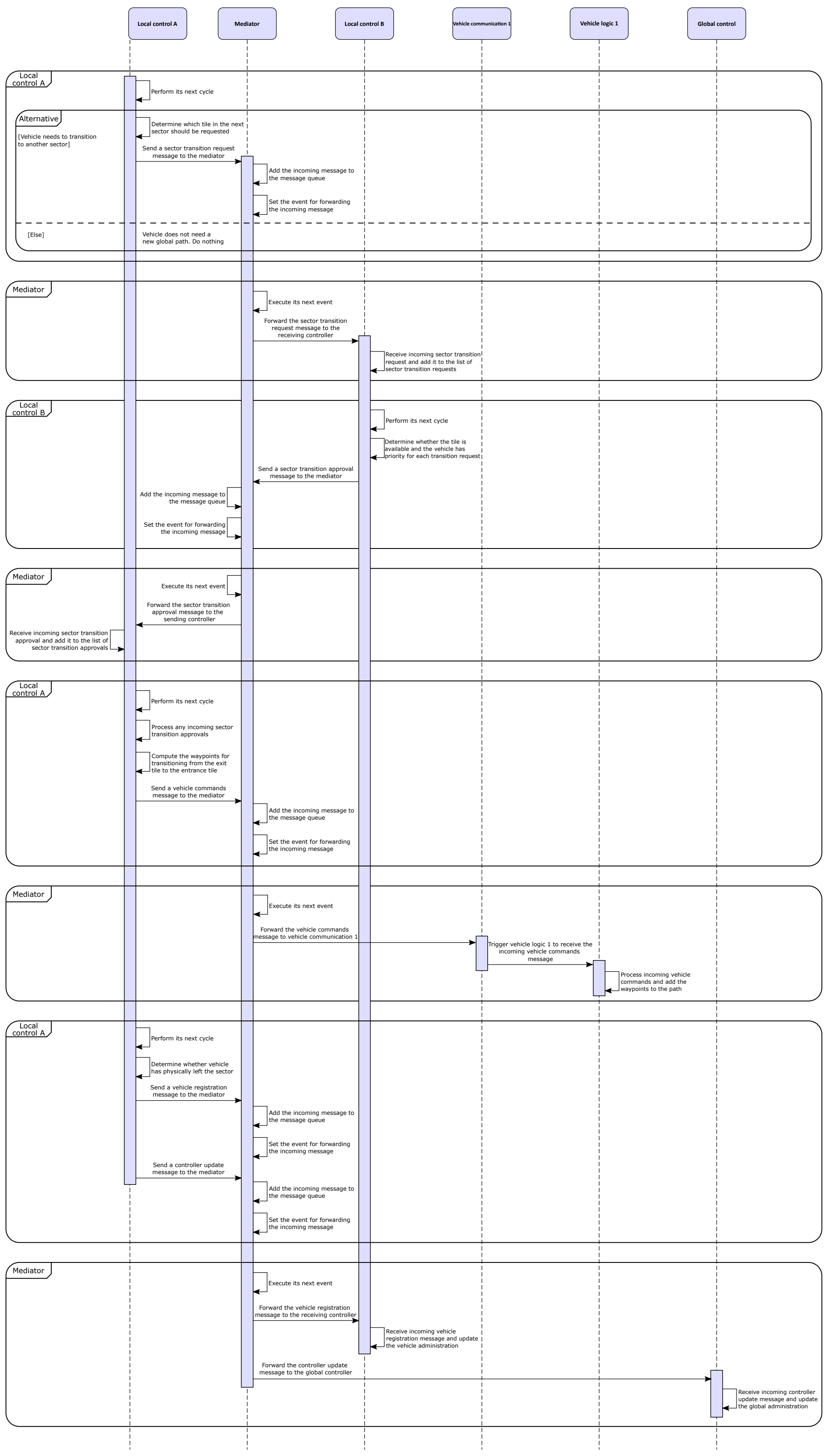

Local control A
Mediator
Local control B
Vehicle communication 1
Vehicle logic 1
Global control
Local control A
Perform its next cycle
Alternative
[Vehicle needs to transition to another sector]
Determine which tile in the next sector should be requested
Send a sector transition request message to the mediator
Add the incoming message to the message queue
Set the event for forwarding the incoming message
[Else]
Vehicle does not need a new global path. Do nothing
Mediator
Execute its next event
Forward the sector transition request message to the receiving controller
Receive incoming sector transition request and add it to the list of sector transition requests
Local control B
Perform its next cycle
Determine whether the tile is available and the vehicle has priority for each transition request
Send a sector transition approval message to the mediator
Add the incoming message to the message queue
Set the event for forwarding the incoming message
Mediator
Execute its next event
Forward the sector transition approval message to the sending controller
Receive incoming sector transition approval and add it to the list of sector transition approvals
Local control A
Perform its next cycle
Process any incoming sector transition approvals
Compute the waypoints for transitioning from the exit tile to the entrance tile
Send a vehicle commands message to the mediator
Add the incoming message to the message queue
Set the event for forwarding the incoming message
Mediator
Execute its next event
Forward the vehicle commands message to vehicle communication 1
Trigger vehicle logic 1 to receive the incoming vehicle commands message
Process incoming vehicle commands and add the waypoints to the path
Local control A
Perform its next cycle
Determine whether vehicle has physically left the sector
Send a vehicle registration message to the mediator
Add the incoming message to the message queue
Set the event for forwarding the incoming message
Send a controller update message to the mediator
Add the incoming message to the message queue
Set the event for forwarding the incoming message
Mediator
Execute its next event
Forward the vehicle registration message to the receiving controller
Receive incoming vehicle registration message and update the vehicle administration
Forward the controller update message to the global controller
Receive incoming controller update message and update the global administration

Figure B.4: Sequence diagram corresponding to the communication process during sector transition from one sector to another in the hierarchical control model.