1 MEI 2018
AUTHOR: D. HAAK
SUPERVISOR: Ir. M.B. DUINKERKEN
Specialization: Transport Engineering and Logistics

Report number: 2018_TEL_8237

Title: AGV Laboratory Control Architecture

Author: D. Haak

Title (in Dutch) AGV Laboratorium Control Architectuur

Assignment: Research Assignment

Confidential: No

Initiator: Ir. M.B. Duinkerken

Supervisor: Ir. M.B. Duinkerken

Date: 1 Mei, 2018

This report consists of 98 pages. It may only be reproduced literally and as a whole. For commercial purposes only with written authorization of Delft University of Technology. Requests for consult are only taken into consideration under the condition that the applicant denies all legal rights on liabilities concerning the contents of the advice.
Preface

This research assignment covers the design of a distributed control architecture for the automated guided vehicle (AGV) laboratory of the Delft University of Technology. To improve the current loosely coupled control structure for future extension a careful and stepwise analysis is carried out and described in this report. The final architecture is implemented in the AGV laboratory to verify the functionality and feasibility. From the first step of obtaining insight in the system this report elaborates to a final design of a control architecture that provides a framework for future development.

I would like to thank Ir. M.B. Duinkerken for the provided support and research directions during the regular meetings. Besides that, the availability of the AGV laboratory of the Delft University of Technology has been of great value.
Summary

Over the years the amount of containers handled worldwide has increased significantly. To maintain steady operations, the increased automation of handling equipment became more important over the years. Especially in ship to shore (STS) operations this automation has already improved the overall system efficiency. The STS operations can be divided in quay crane (QCs) operations, quay transport operations and stacking operations by yard cranes (YCs). On both sides crane automation is experiencing fast development, but transport operation automation has become even more important. These operations are nowadays progressively carried out by automated guided vehicles (AGVs). The Delft University of Technology has a laboratory where such quay transport operations with scaled AGVs are simulated. Although over the years this laboratory and its systems developed it can still be considered as a loosely coupled network of singular components. Therefore, the objective of this research assignment was to design, implement and test a distributed control architecture to couple the existing components in one structured network. A preliminary implementation is provided to proof the concept.

An AGV is defined as a robot that can navigate autonomously based on several navigation methods. Together with the QC and YC operations and navigation instruments the AGVs form the AGV system. The AGV system thus requires the equipment as described above to be controlled by a one or multiple computers. On these computers logistic functions as job scheduling, dispatching, routing and navigation make sure that the system can carry out its required actions in a coordinated and efficient manner. In industrial AGV systems both this equipment as well as the logistic functions are extensively developed and complex. In the AGV laboratory of the Delft University of Technology a simplified version and scaled version of an industrial AGV system is present. Scaled AGVs (1:25) are used to simulate and test actions regarding the logistic functions. The current control architecture is divided in a tactical, operational and physical layer with actions carried out sequential from the top to the bottom layer. Each of the developed system components has been proven to function in an isolated manner, but not as part of an overall system. It can thus be stated that the AGV laboratory is partly based on industrial application, but that the design of an overall control architecture was necessary to eventually simulate according to reality.

To design and implement a suitable control architecture for the AGV laboratory a wide range of options were reviewed. These options were divided in global control architectures, subsystem collaboration methods and ways of communication. Firstly, as global control architectures (de)centralized, distributed, coordinated and multilevel control were evaluated with multi agent systems (MAS) as side note to distributed control. Then for subsystem collaboration the coupled and decoupled and the cooperating or non-cooperating methods were weighted. For all four methods advantages and disadvantages were outlined. Lastly, the communication possibilities were reviewed. A main division was made between the push and pull principles and direct and indirect communication. Each of these options brought advantages as well as disadvantages.

As follow up on this literature study a choice for these three main system segments was made. To cope with future complexity of the system a multilevel, distributed system setting with the MAS concept implemented was chosen. Besides that, the coordinated control theory was added to assign the subsystems in the highest level as coordinator for the lower levels to restrict the actions and calculations in those levels. After defining the global control architecture, the choice was made to partly couple the components in subsystems and let them collaborate on cooperative basis. Lastly, the decision was made to apply a hybrid push and pull communication mechanism to allow for different operational frequencies in the system. To accommodate this hybrid model indirect communication suited best, which resulted in the application of a message board in the format of an external database. Yet, it was also decided to carry out communication between components within a subsystem with direct interaction to expedite data transfers within subsystems.
Applying the above described selections a control architecture was designed. First a strategic, tactical and operational level division was made with in each level one or more subsystems that interact via a central database. These subsystems all have specific tasks that together form the overall system. Within these subsystems agents and a master are present. The agents can be considered as calculative programs that execute logistic functions, while the masters arrange the data streams within and between the subsystems. An important distinction in the design is made between static and dynamic methods. A static method implies that calculations have been done beforehand and the system operates based on these one time calculations. A dynamic method implies that calculations are frequently updated based on actual performance. The control architecture was designed in such a way that it suites the static method but can be used for dynamic methods in the future too. Figure S.1 outlines the three layers with the subsystems and their components.

Figure S.1: Three-layer system structure with subsystems and their components

Each of these subsystems is connected to the central database that can be consulted at different operational frequencies. The specifications of each subsystem, its agents, the database and an illustration of the comprehensive system layout can be found in section 5.1 and figure 5.9 in the remainder of this report. Lastly, the communication structure was defined based on the chosen push and pull principles. The agents push their data via the subsystem master to the database and after a request data is pulled by the master commissioned by the agents. Exception to the data push and pull strategy are emergency messages, these are always pushed to the related system components.

Next step for this research was to follow up the design with a preliminary implementation of the input subsystem in the strategic layer to provide a proof of concept and fundament for further development of the system. As preparation for this implementation a chapter with a more detailed design of the strategic layer was established. In this detailed design component and state diagrams were established to specify how the programs of the input subsystem should be implemented. Based on the current static setup of the AGV lab it was decided that the implementation would focus on the master of the input system and the task agent. These components together should turn a delivered load plan into a task chart that is usable for the lower layers of the system. As experimental setup three computers were used with on the first a MySQL database and on the second and third the input master and task agent MATLAB® files respectively. With this segregated arrangement the distributed setting was tested by using a Java Database Connectivity (JDBC) driver and toolbox for the MySQL database connection with MATLAB® and the TCP/IP toolbox for MATLAB® to MATLAB® communication. With slight adaptions to the predetermined detailed design as consequence of the functionality of MATLAB® the implementation was prepared for the experiments.

To get a thorough insight in the functionality and feasibility of the implemented control architecture five experiments with qualitative as well as quantitative outcomes were formulated. The first test focussed on the functionality of the system and showed that it worked according the detailed design and carried out its actions without interruptions. The second test concerned the flexibility of the control architecture and showed that adaptions or additions in the delivered load plan were accepted and adjusted in the
system. Subsequently, the third test addressed the robustness of the system by testing its reaction to wrong or no messaging. Each of the error attempts resulted in the rightly displayed error message and a retry conform the system design. Thereafter, the fourth and fifth test both accounted for the feasibility of the implementation to lower system layers and dynamic methods. The fourth test showed that the time difference between a single data request and two or three parallel data request from MATLAB® to MySQL was limited to 4.54ms on average. This implicated that the MySQL database is able to process parallel request at a high frequency making it a suitable message board for the designed control architecture. The fifth test involved measurements of the operational frequency of the implementation with a load plan of 10 and 100 rows. The results clearly showed that the current implementation did not fulfil the requirements for the lower system layers and dynamic methods. It was shown that the basic tools provided by MATLAB®, JDBC and the standard TCP/IP toolbox, could not ensure sufficiently fast processing. This resulted in the recommendation to use improved program and tools and setup a more stable isolated network.

After the design, implementation and testing it was concluded that the constructed control architecture can facilitate functioning of the system in the AGV lab, but that with the current tools and network it was not feasible for real time operations. Yet, the current available static system components in the AGV lab now have an assigned position in the overall system structure, with the opportunity to improve towards a fully dynamic system. Future research should continue to extend and advance this fundamental arrangement with top down developments and implementation of components with better computational and processing capabilities connected via a more stable network.
List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGV</td>
<td>Automated Guided Vehicle</td>
</tr>
<tr>
<td>ALTOS</td>
<td>AGV Lab Terminal Operating System</td>
</tr>
<tr>
<td>CC</td>
<td>Centralized Control</td>
</tr>
<tr>
<td>DC</td>
<td>Distributed Control</td>
</tr>
<tr>
<td>DEL</td>
<td>Delay</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>HAZ</td>
<td>Hazardous material</td>
</tr>
<tr>
<td>HUM</td>
<td>Human interface priority call</td>
</tr>
<tr>
<td>IDL</td>
<td>Interface Description Language</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>INT</td>
<td>Integer</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java Database Connectivity</td>
</tr>
<tr>
<td>KPI</td>
<td>Key Performance Indicator</td>
</tr>
<tr>
<td>MAGV</td>
<td>Mini Automated Guided Vehicle</td>
</tr>
<tr>
<td>MAS</td>
<td>Multi-Agent System</td>
</tr>
<tr>
<td>ODBC</td>
<td>Open Database Connectivity</td>
</tr>
<tr>
<td>PDL</td>
<td>Programming Design Language</td>
</tr>
<tr>
<td>QC</td>
<td>Quay Crane</td>
</tr>
<tr>
<td>RISP</td>
<td>Receiver-Intent-Based Sender-Push</td>
</tr>
<tr>
<td>ROS</td>
<td>Robot Operating System</td>
</tr>
<tr>
<td>RTG</td>
<td>Rubber Tyred Gantry Crane</td>
</tr>
<tr>
<td>RMG</td>
<td>Rail Mounted Gantry Crane</td>
</tr>
<tr>
<td>RVIZ</td>
<td>ROS Visualization</td>
</tr>
<tr>
<td>SIRP</td>
<td>Sender-Intent-Based-Receiver-Pull</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Socket Layer</td>
</tr>
<tr>
<td>STS</td>
<td>Ship-to-Shore</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TEU</td>
<td>Twenty-foot Equivalent Unit</td>
</tr>
<tr>
<td>TOS</td>
<td>Terminal Operating System</td>
</tr>
<tr>
<td>TXT</td>
<td>Text</td>
</tr>
<tr>
<td>URDF</td>
<td>Unified Robot Description Format</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>YC</td>
<td>Yard Crane</td>
</tr>
<tr>
<td>MS</td>
<td>Millisecond</td>
</tr>
</tbody>
</table>
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIGURE S.1</td>
<td>Global illustration of three layer system with components</td>
<td>III</td>
</tr>
<tr>
<td>FIGURE 1.1</td>
<td>Illustration of the sts operations</td>
<td>1</td>
</tr>
<tr>
<td>FIGURE 2.1</td>
<td>Top view of agv system</td>
<td>4</td>
</tr>
<tr>
<td>FIGURE 2.2</td>
<td>Illustration of an industrial agv</td>
<td>5</td>
</tr>
<tr>
<td>FIGURE 2.3</td>
<td>Three layer division of operational levels in control systems</td>
<td>6</td>
</tr>
<tr>
<td>FIGURE 2.4</td>
<td>Dubins path examples, left RSL, right RSR</td>
<td>7</td>
</tr>
<tr>
<td>FIGURE 2.5</td>
<td>Top view of a scaled lab AGV with components</td>
<td>8</td>
</tr>
<tr>
<td>FIGURE 2.6</td>
<td>Bottom view of a scaled lab AGV with components</td>
<td>8</td>
</tr>
<tr>
<td>FIGURE 2.7</td>
<td>Illustration of current control architecture of the AGV lab</td>
<td>9</td>
</tr>
<tr>
<td>FIGURE 3.1</td>
<td>Illustration of centralized and decentralized network</td>
<td>12</td>
</tr>
<tr>
<td>FIGURE 3.2</td>
<td>A distributed control structure with direct communication</td>
<td>12</td>
</tr>
<tr>
<td>FIGURE 3.3</td>
<td>A distributed control structure with indirect communication</td>
<td>12</td>
</tr>
<tr>
<td>FIGURE 3.4</td>
<td>Visualization of basic components of an agent</td>
<td>13</td>
</tr>
<tr>
<td>FIGURE 3.5</td>
<td>A coordinated control structure</td>
<td>14</td>
</tr>
<tr>
<td>FIGURE 3.6</td>
<td>Control structure with multiple, hierarchical levels</td>
<td>14</td>
</tr>
<tr>
<td>FIGURE 3.7</td>
<td>Message delivery models with the push and pull principle</td>
<td>17</td>
</tr>
<tr>
<td>FIGURE 3.8</td>
<td>Computation and communication options</td>
<td>17</td>
</tr>
<tr>
<td>FIGURE 4.1</td>
<td>A visual representation of a hybrid push-pull message model</td>
<td>24</td>
</tr>
<tr>
<td>FIGURE 4.2</td>
<td>Basic setup for the subsystem database communication</td>
<td>25</td>
</tr>
<tr>
<td>FIGURE 5.1</td>
<td>Three layer control architecture with corresponding subsystems</td>
<td>26</td>
</tr>
<tr>
<td>FIGURE 5.2</td>
<td>Global setup of the designed input subsystem</td>
<td>27</td>
</tr>
<tr>
<td>FIGURE 5.3</td>
<td>Global setup of the designed scheduling subsystem</td>
<td>29</td>
</tr>
<tr>
<td>FIGURE 5.4</td>
<td>Global setup of the designed dispatching subsystem</td>
<td>31</td>
</tr>
<tr>
<td>FIGURE 5.5</td>
<td>Global setup of the designed routing subsystem</td>
<td>32</td>
</tr>
<tr>
<td>FIGURE 5.6</td>
<td>Global setup of the designed optitrack subsystem</td>
<td>33</td>
</tr>
<tr>
<td>FIGURE 5.7</td>
<td>Global setup of the designed agv subsystem</td>
<td>34</td>
</tr>
<tr>
<td>FIGURE 5.8</td>
<td>Global setup of the designed monitoring subsystem</td>
<td>35</td>
</tr>
<tr>
<td>FIGURE 5.9</td>
<td>Design of the control architecture with subsystems implemented</td>
<td>42</td>
</tr>
<tr>
<td>FIGURE 6.1</td>
<td>Diagram of input master showing its program components</td>
<td>45</td>
</tr>
<tr>
<td>FIGURE 6.2</td>
<td>Diagram of input master showing possible system states</td>
<td>46</td>
</tr>
<tr>
<td>FIGURE 6.3</td>
<td>Diagram of availability agent showing its program components</td>
<td>48</td>
</tr>
<tr>
<td>FIGURE 6.4</td>
<td>Diagram of availability agent showing possible system states</td>
<td>48</td>
</tr>
<tr>
<td>FIGURE 6.5</td>
<td>Diagram of task agent showing its program components</td>
<td>50</td>
</tr>
<tr>
<td>FIGURE 6.6</td>
<td>Diagram of task agents showing possible system states</td>
<td>50</td>
</tr>
</tbody>
</table>
FIGURE 6.7: Diagram of priority agent showing its program components .................................. 52
FIGURE 6.8: Diagram of priority agent showing possible system states........................................... 52
FIGURE 7.1: Overview of the implemented layer and its components .................................................. 54
FIGURE 7.2: Overview of the system setup in the AGV laboratory ..................................................... 55
FIGURE 7.3: Overview of the processes in the AGV laboratory ........................................................ 55
FIGURE 7.4: Example of tables in the MySQL database ..................................................................... 56
FIGURE 7.5: Example of the register_publish chart in the MySQL database ...................................... 57
FIGURE 7.6: Code example for the MATLAB® - MySQL connection ..................................................... 57
FIGURE 7.7: Code example for data import from MySQL to MATLAB® ................................................. 58
FIGURE 7.8: Code example for data insertion from MATLAB® to MySQL ............................................. 58
FIGURE 7.9: Code example for data removal in mysql from MATLAB® ................................................. 59
FIGURE 7.10: Code example for tcp/ip server and client connection .................................................... 59
FIGURE 7.11: Overview of initializer file setup .................................................................................. 60
FIGURE 7.12: Example of the human_interface_chart in the MATLAB® workspace ....................... 64
FIGURE 7.13: Example of the human_interface_chart in the MySQL database ................................ 64
FIGURE 7.14: Example output after the registering of the agent for publishing ..................................... 64
FIGURE 7.15: Example output after registering of the agent for subscribing ....................................... 64
FIGURE 7.16: Example output of the task agent after its main function .............................................. 65
FIGURE 7.17: Example output of the task agent stored in MySQL workbench ..................................... 65
FIGURE 7.18: Run and time output for the input master MATLAB® file .............................................. 68
FIGURE 8.1: Overview of the control architecture and its components ................................................. 73
FIGURE B.1: Initializer MATLAB® m-file ...................................................................................... 79
FIGURE B.2: Input master MATLAB® m-file .................................................................................... 80
FIGURE B.3: Task agent MATLAB® m-file ...................................................................................... 84
List of Tables

**TABLE 4.1:** Example of a data chart in the control architecture communication ........................................... 24
**TABLE 5.1:** Example of a load plan that can be inserted in the system ............................................................... 27
**TABLE 5.2:** Example of an availability chart showing the available times of AGVs ........................................ 37
**TABLE 5.3:** Example of a task chart showing the task to be carried out .......................................................... 37
**TABLE 5.4:** Example of a priority chart showing priority tasks ............................................................................. 38
**TABLE 5.5:** Example of a schedule chart showing the operation schedule ......................................................... 38
**TABLE 5.6:** Example of a dispatch chart showing the assigned tasks per agv .................................................... 38
**TABLE 5.7:** Example of a routing chart showing the determined routes ............................................................ 39
**TABLE 5.8:** Example of a physical action chart showing agv actions ................................................................. 39
**TABLE 5.9:** Example a task complete chart showing completed task ............................................................... 39
**TABLE 5.10:** Example of a zone occupation chart showing the occupied zones ................................................. 40
**TABLE 5.11:** Example of a predicted trajectory chart .......................................................................................... 40
**TABLE 5.12:** Example of an emergency message chart showing direct threats .................................................. 40
**TABLE 5.13:** Example of a collision chart showing future incidents ................................................................. 40
**TABLE 5.14:** Example of a human interface chart based on the load plan ......................................................... 40
**TABLE 6.1:** Example of a register_publish chart with corresponding columns .................................................. 44
**TABLE 7.1:** Input load plan for test 1 of the proof of concept .............................................................................. 61
**TABLE 7.2:** Estimation of required operational frequencies ............................................................................... 63
**TABLE 7.3:** Results of test 3 of the proof of concept ............................................................................................. 66
**TABLE 7.4:** Results of test 4 of the proof of concept ............................................................................................. 67
**TABLE 7.5:** Results of test 5 of the proof of concept ............................................................................................. 68
**TABLE 7.5:** Specifications of the computers and programs .................................................................................. 78
Table of Contents

PREFACE .......................................................................................................................... I
SUMMARY ......................................................................................................................... II
LIST OF ABBREVIATIONS .................................................................................................. V
LIST OF FIGURES ............................................................................................................... VI
LIST OF TABLES ................................................................................................................ VII

1. INTRODUCTION ............................................................................................................. 1
   1.1 GENERAL INFORMATION AND RELEVANCE ............................................................. 1
   1.2 AIM AND SCOPE OF STUDY ....................................................................................... 2
   1.3 OUTLINE .................................................................................................................... 2

2. AGV SYSTEM COMPONENTS ......................................................................................... 4
   2.1 INDUSTRIAL AGV SYSTEM ....................................................................................... 4
      2.1.1 Equipment ............................................................................................................ 5
      2.1.2 Logistic functions ................................................................................................. 6
   2.2 AGV LABORATORY .................................................................................................... 8
      2.2.1 MAGV .................................................................................................................. 8
      2.2.2 Control architecture ............................................................................................ 9
   2.3 CONCLUDING REMARKS ......................................................................................... 10

3. CONTROL ARCHITECTURE OPTIONS ......................................................................... 11
   3.1 GLOBAL CONTROL ARCHITECTURES ..................................................................... 11
      3.1.1 Classification ....................................................................................................... 11
      3.1.2 Comparison and choice principles ...................................................................... 15
   3.2 SUBSYSTEM COLLABORATION ............................................................................... 15
   3.3 COMMUNICATION ...................................................................................................... 16
      3.3.1 Push and Pull ...................................................................................................... 16
      3.3.2 Direct and message board communication .......................................................... 17
   3.4 CONCLUDING REMARKS ......................................................................................... 19

4. CONCEPT CHOICES ...................................................................................................... 21
   4.1 GLOBAL CONTROL ARCHITECTURE CHOICE ......................................................... 21
   4.2 SUBSYSTEM COLLABORATION ............................................................................... 22
   4.3 COMMUNICATION .................................................................................................... 23
   4.4 CONCLUDING REMARKS ......................................................................................... 25

5. GLOBAL CONCEPT ....................................................................................................... 26
   5.1 SUBSYSTEMS AND DATABASE ................................................................................ 26
      5.1.1 Input subsystem .................................................................................................. 27
      5.1.2 Scheduling subsystem ....................................................................................... 29
      5.1.3 Dispatch subsystem .......................................................................................... 30
      5.1.4 Routing subsystem ............................................................................................ 31
      5.1.5 OptiTrack™ subsystem ...................................................................................... 33
      5.1.6 AGV subsystem .................................................................................................. 34
      5.1.7 Monitoring subsystem ....................................................................................... 35
      5.1.8 Database ............................................................................................................ 37
   5.2 COMMUNICATION FEATURES ................................................................................ 41
   5.3 OVERALL CONTROL ARCHITECTURE .................................................................. 41
1. Introduction

1.1 General information and relevance
Over the last years the amount of worldwide container throughput in ports increased from 156 million twenty foot equivalent units (TEUs) in 1996 to 673 million TEUs in 2012 (Statista I). With a projected growth rate of 4.7% (Statista II) for the timeframe 2016-2019 this volume will keep increasing eminently. Container handling equipment is advancing with this increase in throughput volume. So automation is considered necessary to maintain steady operations. Particularly, the ship to shore (STS) operations are processes with extensive interest in automation. These STS operations can be divided in quay crane (QCs) operations, quay transport operations and stacking operations by yard cranes (YCs), illustrated by figure 1.1.

![Figure 1.1: STS operations (Petering et al., 2009)](image)

All of these operations have potential for automation, already shown in some ports worldwide. Transportation of the TEUs between the QC and YC is nowadays increasingly carried out by automated guided vehicles (AGV). These AGVs act as regular trucks, but operate driverless based on automated operating systems. Since the first studies on AGV application in container terminals (Durrant-Whyte, 1996), (Evers et al., 1996), these operations remarkably developed with increased operating efficiencies as result.

Currently, AGV laboratories have had and still have a significant share in the progressive evolution of the required operating systems that guide these AGVs. Scaled simulations in these laboratories show the effect of newly developed hard- and software. The Delft University of Technology has such a laboratory, which is a scaled and simplified representation of the setup shown in figure 1.1. The control architecture of this scaled system receives randomly generated tasks and converts those to predetermined trajectories based on the marked operation field. These predetermined trajectories are then passed on to the control system of the AGVs, which ensures that actuators direct the AGV accordingly. As the AGVs then follow that trajectory through the marked simulation area, the quay transport operations are simulated corresponding to reality. Contrary, the QC and YC operations are only simulated by instructing the AGV to execute a standstill at a predetermined location for a certain time slot. This implies that potential operational difficulties of these QCs and YCs that relate to the navigation of the AGVs are not taken into account yet.
Although the system components in the laboratory function reasonably, they do not act sufficient enough as a fully coordinated system. The collaboration between the above described operation simulations and especially between the sub components of the AGV navigation system still lack coordination. As the three logistic operations and AGV navigation interactions are carried out real-time, structured collaboration of those is of significant importance. Therefore, it is of relevance for this simulation laboratory that a structured terminal operating system (TOS) for the operations of figure 1.1 is constructed to enhance the coordination of the real-time logistics. Furthermore, a preliminary control architecture design for this simulation lab could potentially constitute to enhanced QC and YC simulations and future development of related real-life control systems for AGV operated terminals.

1.2 Aim and scope of study
The main aim of this study is to design an operating system architecture for the simulations in AGV lab of the Delft University of Technology. These simulations comprise of the quay crane, quay transport and yard crane operations as described before. The intention of this design is to bundle the subsystems in a structured and layered network to simulate the STS operations of a TOS in real world container terminals. Control systems can mainly be based on two different concepts (Evers et al., 1996); the centralized control (CC) model and the DC (DC) model. The CC model has one component that is designated as controller that is responsible for the execution of other system components. The DC model is a control system with control loops in which autonomous controllers are distributed through the total system. This research will comprise the DC model theory in the operating system design. In the remainder of this report a short literature review in chapter 3 will elaborate more on the available control structure possibilities. Besides the overall design of the operating system architecture, this research will elaborate on communication links and interaction interfaces of the subsystems. However, the mathematical principles of the subsystems themselves are not part of this study and are considered as completed and fully functional.

The primary research question is defined as; “What is a suitable architecture for a DC network that facilitates efficient functioning of the overall system and how could it be implemented in real container terminals?” A set of sub questions is established of which the answers should lead to a comprehensive and solid answer to the main question:

- What are components of an industrial AGV system and how are these present and simulated in the AGV lab?
- What are the control structure and related collaboration and communication options?
- Which of the described possibilities fit the AGV Laboratory TOS (ALTOS)?
- How can these possibilities be combined to a functional control architecture design?
- How can the concept be tested and what are the results of those tests?

With answering these sub questions a DC operating system can be set up connecting the present components of the AGV lab in an appropriate and innovative way. The setup of this research study report, based on these questions, is provided in the next section.

1.3 Outline
After the introduction the report starts in Chapter 2 with a description of the AGV system and the relation of the lab with real automated container terminals to answer the first sub question. Then the report continues with a literature survey of the potential control architecture possibilities in Chapter 3, which comprises of literature background related to sub question two. Subsequently, Chapter 4 continues with argued decisions of control architecture options to lay a fundament for the remainder of the report and answer sub question three. Then, Chapter 5 combines the components and chosen structures into a global system layout design to answer sub question four. With this global layout a general insight in the setup of the architecture of the system will be obtained. As now the overall system layout is known Chapter 6 extends the theory behind the ALTOS with a detailed description of a single subsystem and its connection to the database, which thus elaborates on the answer of sub question four. Based on this detailed design Chapter 7 continues with a system implementation and proof of concept.
with experiments and results based on predefined key performance indicators (KPIs) answering sub question five. Finally, Chapter 8 concludes the report while answering the primary research question proposed in Chapter 1 and provides further research recommendations.
2. AGV System Components

This chapter briefly outlines the components of an AGV system applied in STS operations and the relation to the AGV lab setup. Section 2.1 first describes the AGV system and its segments that are present in the real container terminals. Then section 2.2 follows with the setup of the AGV lab and how it is related to the real AGV systems. Finalizing the chapter section 2.3 consists of concluding remarks.

2.1 Industrial AGV system

An AGV is a robot that can navigate autonomously based on several navigation methods. Application of these AGVs has a wide variety, with container terminal operations as remarkable share. The implementation of those automated robots in container terminals have, as mentioned in the introduction, contributed to increased operational efficiencies. Still, an AGV is not able to carry out all the STS actions required in a container terminal. Therefore, it should interact with QCs and YCs as can be seen in figure 1.1. The system that comprises of these operations carried out by the QCs, YCs and AGV is called the AGV system. In combination with figure 1.1, that shows the three operations, figure 2.1 shows a top view of an AGV system to clearly identify the boundaries.

![Figure 2.1: Top view AGV system (Bahnes et al., 2016)](image)

The terminal area and specifically the AGV operational area can vary a lot regarding their layout influencing the AGV system. As described in the Chapter 1 the operational area of the AGVs is limited to the zone between the QCs and the YCs with a rectangular shape, displayed as AV mobility trajectory in figure 2.1. This zone is completely shielded from the other terminal zones with human presence. The AGVs are able to drive underneath the QCs and YCs to be loaded and unloaded. Depending on the abilities to lift the platform of the AGV it is also possible to unload the container on a stacking rack in front of the stacking yard instead of direct pickup by the YC.
To keep the AGV system functional multiple equipment and logistic function have to cooperate. Further insight in the system is therefore presented by an elaboration on these equipment and logistic functions. As equipment the AGV itself, QCs and YCs, navigation equipment and the control computer are discussed. For the logistic functions job scheduling, dispatching routing and navigation methods are specified.

2.1.1 Equipment

**AGV**
The AGV is the most essential part of the AGV system as it carries out the assigned transport tasks. As it is considered as a robot that follows a certain predefined trajectory human interaction is not required to operate these systems. Various types of AGVs are available that are applicable to a wide variety of industries, resulting in different constructions. Yet, in the scope of this research the focus will be on AGV application in container transport. These AGVs are specifically designed to transport containers of 20ft or 40ft. The basic components of those AGVs are the following:

- (Non-) Lifting load platform
- Wheels
- Power source
- Drive system
- Steering mechanism
- Safety barriers

Combined these components form the base of an AGV as shown in figure 2.2.

![Figure 2.2: Container AGV (DAEL)](image)

**Quay and yard cranes**
The QCs and YCs take care of the lifting operations from the ship to the AGV and from the AGV to the stacking area respectively. The QC is located at the dockside of the terminal area and reaches out over the water. To lift the container of the ship a handling tool called spreader is used. This is a device that can lift unitized containers after locking on to the corners of the unit. Localization of the spreader to pick up the container and to drop the container exactly on the AGV is a precise and tedious process. The YC is specifically constructed to positions the containers brought by the AGV in a stacking area. Most commonly used for the stacking operations are the rubber tyred gantry crane (RTG), rail mounted gantry crane (RMG) and the straddle carrier. All three of these machines are smaller than the QC but are still able to lift a loaded container over multiple stacked containers in width and height. Again these cranes use a spreader to lift and drop the container at the right location with high precision. Automation of QCs and YCs is increasingly applied and slowly becoming the state of the art.
Navigation equipment

For the navigation of the AGVs different techniques that require various equipment are installed in the terminal area. Known techniques for common AGV navigation are the following (Götting, 2000):

- Optical guidance
- Electromagnetic reference mark guidance
- Electromagnetic line guidance
- GPS route guidance

The inductive wire, electromagnetic reference mark and GPS route guidance are the most common techniques for navigation of container AGVs. Important asset of these navigation techniques are the sensors placed on the AGV and in the surroundings. These sensors can measure the required information for the computational systems to locate and direct the AGV. Depending on the AGV and navigation method chosen different sensors are used.

Control computer

In a fully automated system the control computer is the most fundamental component. This computer oversees and regulates processes like job scheduling, dispatching, routing and navigation. As processes become increasingly complex the central control tends to become more distributed over cooperative computers. Each of these computers in the system have their own assignment, while the collaboration of those results in a fully functioning system.

2.1.2 Logistic functions

The logistic functions take care of the mathematical calculations that determine the order and performance of processes in the system. Each of these functions have different time and space scales and can be designated as strategic, tactical or operational planning. In the strategic level the strategy for the total system, not just units of it, is defined. One level lower, the tactical level takes care of short range planning of operations of various parts of the system. The lowest, operational level, then takes care of coupling the strategic and tactical objectives to achieve the overall goal of the system. From the assigned tasks it can be deduced that each of these levels thus have a different operational time and scale. The strategic level comprises the input preparation of the AGV system and has lower frequency update times. The tactical level specifically focusses on the routing processes of the AGVs and has an update frequency that is higher than that of the strategic level. At last, in the operational level the focus is on the navigation of a single AGV with real-time high frequency updates. Figure 2.3 shows these three levels in a top-down pyramid structure.

![Operational levels diagram](image-url)

*Figure 2.3: Operational levels (Negenborn, 2015)*
Based on this pyramid, three level structure the following logistic functions are elaborated on in the perspective of the AGV system; job scheduling, dispatching, routing and navigation. Each of these functions is assigned to one of the three levels. As the strategic level mainly consists of processing the load plan the logistic functions will either belong to the tactical or operational level.

**Job scheduling**
The first process of the AGV system is the scheduling of the load plan, which can be categorized as being of tactical planning. The jobs consist of picking up a container at a certain location and dropping it off at a secondary one. The optimizing algorithms calculate the sequence of jobs and the estimated operational times per job. Real world scheduling problems often involve optimization of multi objectives, while computerized scheduling is mainly dominated by a less realistic single objective approach (Udhayakumar et al., 2010). Stated in the same paper is the seek for a multi-objective optimization technique to obtain a set of Pareto optimal solutions that satisfy the multiple constraints that play a role in job scheduling. Genetic and Ant Colony optimization algorithms are frequently used to carry out the process of job scheduling. Lastly, scheduling can be carried out in a static or dynamic manner. With static scheduling the complete schedule is calculated before operation based on estimations and assumptions. With dynamic scheduling the schedule is created and updated based on actual performance of the system and jobs are scheduled depending on previous completed ones.

**Dispatching**
Dispatching continues on job scheduling, also in the tactical level, by assigning the scheduled jobs to the available AGVs. Each of the AGVs receives a sequence of jobs it has to carry out in the predetermined sequence. From the initiations of the system the AGVs processes these jobs till the full schedule is completed. Again a distinction can be made between the static and dynamic processing. With static scheduling the dispatching is also done statically before the real-time operation starts. With dynamic scheduling the dispatching is done based on the real-time performance where an AGV is assigned to a new job as soon as it asks for a new job after completion of a previous one.

**Routing**
The follow-up process after dispatching is routing, which is part of the tactical planning layer too. The routing algorithm selects the route that the AGV should follow in order to complete the previous assigned task. Many different methods to carry out this routing function are known in literature. For a brief overview of some of the methods Routing automated guided vehicles in container terminals through the Q-learning technique (Jeon et al., 2011) can be addressed.

Yet, one routing algorithm of particular interest with respect to usage in the AGV lab is the Dubins path method. The Dubins path indicates the shortest curve that connects two points in the x-y plane with a constraint on the arc and with prescribed initial and terminal tangents to the path while assuming only forward travelling (Dubins, 1957). It basically suggests that a shortest path between two points can be expressed as a combination of three motion primitives; “left turn (L)”, “right turn (R)” or moving “straight (S)”. This results in the fact that the optimal path will always be one of the following six combinations; LSR, LSL, RSR, RSL, RLR or LRL. A visual representation of the RSL and RSR Dubins path is provided in **figure 2.4**.

![Figure 2.4: Dubins path examples, left RSL, right RSR (Hvamb, 2015)](image)
Navigation
As now the jobs are assigned and the routes calculated the last logistical algorithm concerns the navigation of the AGV, classified as part of the operational level. Depending on the previously described methods in subsection 2.1.1 different algorithms apply. Each of these algorithms calculate the required real-time actions that the AGV should carry out based on the information provided by the equipment and higher layer functions to follow the predetermined route. Intensive collaboration between the routing and navigation processes is of essence to ensure precise navigation, collision avoidance and allowance for route updates along the movement process of the AGV.

2.2 AGV Laboratory
The AGV lab of the Delft University of Technology is used to allow students to develop existing or new systems related to the container AGV operations. The main purpose is to experiment with scaled AGVs and AGV-systems and to carry out research regarding complex transport systems with intelligent AGVs. Currently, only one of the three in the introduction described operations is simulated with related hardware. The QC and YC processes are modelled as waiting times at specific locations. Developments to extend these processes in order to get closer to reality are under construction. Contrary to these time simulated operations, the process of container transport is carried out by 25 mini-AGVs (MAGV), scaled 1:25 according to AGVs for container transport. Still, material handling, in the sense of scaled containers, is also not yet part of the simulation.

2.2.1 MAGV
The MAGV models are equipped with two servos for driving and steering of both axes. Each of these servos is able to accomplish angles between -45° and 45°. Besides that, sensors for odometry and a processor based computer, the Arduino®, with wireless network are present. The position of the MAGVs is determined by the separate camera system called OptiTrack™. This system is able to track all MAGVs based on the OptiTrack™ markers place on top of the model. The OptiTrack™ can thus be seen as the hardware for the navigation of real container AGVs. A top and bottom view of the MAGV is provided in figure 2.5 and figure 2.6 with numbered components. In essence it can thus be seen that the setup of the MAGVs, although simplified, is similar to that of a real AGV.

Figure 2.5: Top view; 1. Battery, 2. Arduino board, 3. OptiTrack™ markers

Figure 2.6: Bottom view; 1. Engines, 2. Servos
2.2.2 Control architecture

The control architecture of the AGV lab can partly be seen as loosely configured programs as most of the subsystems have separately been designed by students. For now, the architecture is divided in the three different levels as described in subsection 2.1.2. The first level, the tactical layer, consists of the function transport planning system with job scheduling and dispatching and the function routing based on Dubins trajectories or free range routes. It combines these algorithms with time based zone claiming to prevent collisions or deadlocks. These functions are similar to the ones used in reality, but with less complex objectives and restrictions. The second layer, the operational layer, consists of an observer with Kalman filtering and sensor fusion and a controller based on open or closed loop. The observer receives data from the OptiTrack™ system, Arduino® board and controller which it combines into a state estimation that is passed on to the controller. At this moment a PID-controller is used to compare the reference state provided by the tactical layer with the provided actual state estimation of the AGV. Based on this comparison the controller determines the appropriate actions that are then send to the lowest layer. This third level, the physical layer, consists of the Motive Tracker® and the Arduino® board. The OptiTrack™ Motive Tracker® combines the data provided by the OptiTrack™ hardware and sends it to the observer in the operational layer. The Arduino® board controls the actuators on the MAGV and sends sensor data back to the observer in the operational layer. This Arduino® board can be considered as a simplified version of the on-board computer of real AGVs. The discussed architecture is shown in figure 2.7 to obtain an even more excelling understanding of the system layout.

![Diagram of AGV system control architecture](image-url)

**Figure 2.7: AGV system control architecture**
2.3 Concluding remarks

The AGV system components are elaborated on and a relation is shown between the AGV laboratory and a real container terminal. For the industrial AGV systems a distinction was made between the required equipment and logistic functions. The equipment mainly consists of AGVs, QC's and YCs and a control computer. The logistic functions, taking care of the mathematical calculations that determine the order and performance of the system, were divided over the three operational levels; the strategic, tactical and operational level. The job scheduling function, which is allocated to the tactical level, takes care of the scheduling of the load plan. The dispatching function, also allocated to the tactical level, takes care of the dispatching of the available AGVs. The routing function, again allocated to the tactical level, ensures that every dispatched AGVs receives the right trajectory data. With these functions carried out the navigation function, allocated to the operational level, makes sure that the AGVs receive the appropriate physical instructions to obey the instructions from the higher levels.

With the real industrial global setup discussed a similar distinction was made for the AGV laboratory. A short insight in the scaled AGVs, MAGVs, was provided showing the present components with a top and bottom view. Although simplified, the main components of these MAGVs were similar to that of real AGVs. However, QC's and YC's are not yet available as scaled components. Contrary to the relatively on industry based MAGVs, the control architecture of the AGV lab thus still lacks in its similarity to real life application. The current system can partly be seen as loosely coupled programs that can only work in sequential order. Still, the system is based on the three-layer setup but with different functions in each layer as can be seen in figure 2.5.

Overall it can be stated that specifically the quay transport operations in the lab, with the use of scaled AGVs and pickup and drop-off locations, are simulated according to the real world. However, the operations of the QC's and YC's do significantly influence the overall performance of the system and are not simulated in the AGV lab. Considering this shortcoming, future goals could be to extend the simulation with a fully integrated system consisting of all three processes. To permit such operations it is of relevance that the current control architecture is adjusted to one with a DC perspective with functions allocated to one of the three layers as is done in real life application. The remainder of this report is dedicated to that purpose.
3. Control Architecture Options

This chapter elaborates on the possibilities that are available to set up a control architecture for an AGV system. Not only the basic structures are discussed, but also the level of collaboration of the components and the way in which these communicate. Section 3.1 describes the possible global control architectures for the system. Afterwards section 3.2 and section 3.3 continue with subsystem collaboration and ways of communication. At the end of the chapter an insight is obtained in potential options for the control architecture of the AGV lab. Section 3.4 then concludes the chapter.

3.1 Global control architectures

Regarding global structures this section is based on Coordination Control of Distributed Systems (van Schuppen et al., 2014) with some small additions to extend the provided information. It is stated that a control architecture is a description of the various controllers of a control systems and the ways in which these controllers are functioning. The choices made in designing a control architecture have to be based on full understanding of the available concepts. Therefore, these concepts are elaborated on in the continuation of this section.

3.1.1 Classification

In control theory still no standard is available for the classification of control architectures. As it is useful to have one, the guidelines provided by van Schuppen (van Schuppen et al., 2014) will be taken as main thread. These guidelines are preliminary and are subjected to changes over the years. The following system structures have been declared as possible classifications:

- (De)centralized control
- Distributed Control (with direct communications between controllers)
- Coordinated control
- Multilevel control

(De)centralized control

CC is characterized by one component that is designated as controller and is responsible for managing completion of tasks by other components. Main advantage of such a centralized system is its simplicity. The disadvantage is mainly the complexity of the control synthesis, the communication efforts required and the lack of robustness in case of failure. Mainly two different classes can be distinguished in CC, the sequential or parallel class. With sequential execution the system is a top-down subroutine model with the control starting at the top of the hierarchy and then works down to the lower levels. With parallel execution the one designated control component manages other components parallel to each other. Important implementation of this parallel class is in systems where components depend on values of other variables. As counterpart of the centralized concept the decentralized concept exists. In the decentralized concept lower level components operate on local information to accomplish the global goals and not on instructions or commands from higher levels. Special characteristic of the decentralized system is that the control input of individual systems can be computed without the knowledge of other agents in the system. As result of this configuration an advantage of such a decentralized system is fault tolerance. The decentralized systems are less likely to have accidental failure as they rely on separated components. Figure 3.1 visually shows the differences between these two concepts.
Distributed control (with direct communication between controllers)

DC architectures are characterized by having multiple controller loops, in which autonomous controllers are distributed over the system. Advantages of this distributed system are higher reliability, scalability and reduced installation cost as control functions can be localized. Disadvantages are known to be the more complex structures that are harder to synchronize, increased processing overhead due to more exchange of information and additional computations and lastly higher costs due to the implementation of more advanced distributed systems. An important choice in the DC architecture is whether to allow direct communications between controller or not. There is always indirect communication between controllers via the control system, but for direct communication the information goes directly from one controller to the other. Drawback of the direct communication is an even more difficult control synthesis. Protocols have to be set up to determine when extra direct observations have to be requested or sent and how this extra information can be implemented. Figure 3.2 and figure 3.3 show a DC architecture with indirect and direct communication respectively.

**Figure 3.1:** Centralized network (Left) and decentralized network (Right) (Baran, 1962)

**Figure 3.2:** DC with indirect communication. $S_k$ are subsystems and $C_k$ are the corresponding controllers (Van Schuppen et al., 2014)

**Figure 3.3:** DC with direct communication. $S_k$ are subsystems and $C_k$ are the corresponding controllers (Van Schuppen et al., 2014)
A sub class that is often connected to DC with direct or indirect communication is the multi-agent system (MAS). MAS consist of autonomous, interacting, more or the less intelligent agents (Braubach et al., 2004). In Artificial Intelligence, a Modern Approach (Russell et al., 1995) it stated that an agent’s choice of action at any given instant can depend on the entire percept sequence observed to date, but not on anything it hasn’t perceived. In the perspective of application of agents in AGV control, the definition of the agent can be considered as software that processes data from connected agents to construct data output that is necessary for functioning of other agents in the system. In principle an agent is thus a small data processing programs in the overall control structure. The data for this agents is particularly formatted in charts with rows and columns. The structure of these charts is later discussed in section 4.3. For more detailed information about agents and their different classes (Russell et al., 1995) can be consulted. Yet, for general visualisation figure 3.4 shows the structure of an agent as described above that is able to maintain a certain instructed state.

![Figure 3.4: Agent that maintains state (Wooldridge, 1999)](image)

In a MAS these agents, which are thus in essence small data processing programs, process at the same time while sharing common resources network. Besides that, based on required information the agents can communicate with every other agent in the system via direct communication. Flores-Mendez (Flores-Mendez, 1999) formulates MAS as a loosely coupled network of problem-solver entities that work together to find the answer to problems that are beyond the individual capacities or knowledge of each program. This thus means that complex problems can be divided in less demanding problems, which require a less complicated processing program as agent. Key issue of this divided autonomous problem solving is to have coordination, responsibility and behaviour instructions formalized flawless. This implicitly indicates the disadvantage that individual agents can easily provide inadequate solutions for the global problem if one of the interactions between other agents or databases lack.

Regarding the above described hierarchy and MAS architecture, a combination of both is also possible resulting in hybrid concept. In this concept the lower level-level agents can negotiate with each other within boundaries established by higher levels. This hybrid concept provides developers and operators with a clearer overview of defined agent responsibilities and communications resulting in a potentially better perception and functioning of the total system.

**Coordinated control**

A coordinated system consists of a coordinator and two or more subsystems such that conditioned on the coordinator the subsystems are independent (Van Schuppen et al., 2014). In this system a controller is present for this coordinator as well as for the subsystems. The coordinator influences the subsystems while the subsystem do not influence the coordinator on their turn. The responsibility of the coordinator is to regulate the actions of the subsystems. Van Schuppen states that coordination control becomes of interest when the control objective cannot be met by DC with or without direct communication. A certain degree of coordinated control has to be imposed in that case. Figure 3.5 shows the architecture for a
coordination control architecture. Main difference with the concepts of figure 3.2 and figure 3.3 is the addition of a coordinator, displayed as $S_k$.

![Figure 3.5: Coordinated control. $S_k$ is the coordinator, $S_1$ and $S_2$ the subsystems and $C_k$ are the corresponding controllers (Van Schuppen et al., 2014)](image)

**Multilevel control**

A multilevel system consists of multiple levels in which each subsystem of a particular level is connected to the level below and above it. The multilevel system is often referred to as a hierarchical control system. Most of these hierarchical systems are frequently combined with the previous described DC. In *Distributed Hierarchical Control for Parallel Processing* (Feitelson et al., 1990) the combination of distributed and hierarchical control is proposed as possibility for parallel operating systems. In hierarchical control the complex control problem is decomposed in smaller sub problems while assembling the solutions of those subsystems into a “functioning” hierarchical structure (Max Planck Institute). Main characteristic of the hierarchical structure is the presence of multiple controllers divided over different levels, each obeying instructions from the controller one level above their own. The higher level controllers in this hierarchical structure do not keep track of all individual details of the lower level systems, but only use data supplied by the controller on the level below. Furthermore, it is stated by *Feitelson and Rudolph* (Feitelson et al., 1990) that controllers in the same level of hierarchy disjoint sets of controllers in the level below, providing a spatial portioning of the system. This results in local control when possible and global coordination when needed based on the three planning levels as described in subsection 2.1.2. A visualisation of a multilevel control architecture is provided in figure 3.6. The DC aspect can be implemented while using the same structure.

![Figure 3.6: Multilevel (hierarchical) control with boxes representing a closed-loop system consisting of a subsystem and controller and links as two-directional communication](image)
3.1.2 Comparison and choice principles
Van Schuppen (Van Schuppen et al., 2014) provided preliminary guidelines that can be followed in order to compare the above described sub classes of control architectures. As additions are included some deviations can be present regarding these guidelines based on the supplementary literature.

1. If a particular system is not of high complexity application of central control is the easiest implementable control architecture. When a system needs more flexibility and fault tolerance is an important aspect the decentralized concept could be a better option.
2. If the control input of the subsystems cannot be computed without the knowledge of the states of other subsystems and a system becomes to complex, a DC architecture could fit better.
3. If the closed-loop system with DC architecture cannot meet the control objectives, a DC architecture with direct communication instead of indirect communication could be considered.
4. If the performance with respect to the control objective is still unsatisfactory and more intense coordination of subsystems is required, the coordinated control architecture could be used.
5. If the complexity of the DC system is very large a multilevel system, in the sense of a hierarchical structure seems the best.
6. Lastly, it is important to notice that often a hybrid control architecture could be an even more satisfying concept as it can accomplish different system objectives on multiple levels in the system. Drawback of this hybrid implementation is the often hard control synthesis of the overall system.

3.2 Subsystem collaboration
Based on the interactions between subsystems and the information being used the control method can be classified as coupled or decoupled. Using the coupled method of subsystems, the overall system is considered as a composite system to which classical single-vehicle motion planning algorithms are applied (Draganjac et al., 2016). The coupled method is known as highly demanding in terms of computational resources with exponentially increasing computational complexity as, in the case of AGV control, the number of AGVs increases. To restrict this computational complexity, the decoupled method is an option. In this case some subsystems are completely decoupled from each other so that they carry out their assignment without considering information from the other systems. This reduces computational complexity but comes at the cost of losing completeness and being suboptimal (Draganjac et al., 2016). As examples of decoupled methods zone control, time windows and multi-agent systems are mentioned.

A second important aspect of subsystem collaboration is whether they are cooperating or non-cooperating. Based on the game theory cooperative control is now as control with components that do not function with only a self-enforcing perspective. The decisions made by the system components do take into account states of other components in the system to comply with a common goal. Main difficulties of this cooperative approach are high volume communication streams, extensive computational algorithms and required exact synchronization. Briefly described a cooperative has four basic elements: group objective, agents/subsystems, information topology and control algorithms governing the motion of the agents/subsystems (Bai et al., 2011). Non cooperative control is the opposite in terms of collaboration of system components. In this non cooperative structure the components only do their job with the underlying perspective of maximizing their own objective. In that case the states and outcomes of other components resulting of the decisions made by that one particular component do not matter. The non-cooperative approach is more general and requires less interaction between components. However, this goes often at the expense of a less optimal overall system outcome. Possible modification to the non-cooperative control is a control architecture with coalitions. Within these coalitions the components work according the cooperative theory, while the coalitions themselves apply the non-cooperative control between each other.
3.3 Communication
Regarding communications between system components two aspects will be discussed in this section. First of all, the push and pull principle will be shortly reviewed followed by an elaboration on direct or indirect communication, which is referred to as message board communication.

3.3.1 Push and Pull
The push-pull mechanism, commonly known from inventory control, does also apply to communication techniques. In *Push vs. Pull: Implications of Protocol Design on Controlling Unwanted Traffic* the “Sender-Push” model and the “Receiver-Pull” model are elaborately discussed (Duan et al., 2005). With the importance of this principles in mind, the following section described the basic structure of each of the possible options based on this paper.

In the sender-push model, the sender knows the identity of a receiver in advance and pushes the message in an asynchronous manner to the receiver. The receiver can then accept the complete message or review it first before accepting or discarding it. Important for proper functioning of this model is that the entire message is received before any receiver-side processing is performed. Primary advantages of the sender-push model are that its asynchronous message delivery structure is simple, easily applicable for most systems and that there is no significant storage required on the sender side. The disadvantage of this mechanism is that the sender completely controls what message is when delivered without the receiver knowing anything about it. This means that the receiver has to receive or discard, process and store the message even if it is not of interest. To partly deal with these drawbacks the receiver-intent-based sender-push (RISP), variant of the sender-push model, can be implemented. In this model users subscribe to a service, which subsequently push the data to the receiver. Besides that, the receiver can provide control feedback about the messages being sent. This looping concept could provide a basis for communication of only the minimum required amount of information. Summarized the sender-push concept is a simple and convenient communication model, but the total control by the sender can have significant negative consequences.

The other option is the receiver-pull model with the receiver commencing the information transfer by approaching the sender. In this case the sender passively waits for a call of the receiver and delivers the entire message upon receiving that call. This means that the receiver has explicit greater control over the transfer and implicit greater trust in the received content then with the sender-push model. Similar as with the sender-push model a variation on this model is available called the sender-intent-based receiver-pull (SIRP). In this concept the sender first expresses an intent to send content to the receiver via an intention message. If the receiver is interested it contacts the sender and requests the information. Main feature of this concept is that the receiver pulls the content while only a short intent is pushed by the sender. First advantage of the receiver-pull model is that the receiver can determine its own level of interest in the available contents before requesting information. Secondly, it becomes the responsibility of the sender to store and manage information till the receiver asks for it. Lastly, there is a large window of time in which the receiver is free to verify a sender’s identity and information. On the other side the receiver-pull model results in more content management for the sender. The sender now needs to control and maintain the information till the receiver is willing to retrieve it. Important is that specific protocols are available regarding information removal if it is not retrieved at all. If data is stored for too long the queues arise resulting in overload of the system and moreover use of outdated information by the programs that consult this data.

The four above described models, sender-push, receiver-pull, RISP and SIRP are shown in figure 3.7.
3.3.2 Direct and message board communication

Another model feature to consider is whether to allow communication directly from component to component or via a message board in the sense of a database. However, before describing the direct and indirect message board communication it should be stated that each of the subsystems can operate with different communication frequencies. Regardless of the way in which the subsystems communicate the communication frequency depends on the system levels, discussed in subsection 2.1.2 and shown in figure 2.3. As mentioned, the strategic level commonly has lower communication frequencies than the tactical and operational level and the tactical in its turn lower than the operational level. These differences in frequencies are later defined in the design of the control architecture, while the remainder of this section continues with a division of direct and message board communication.

Direct communication

For the direct communication between components six schemes, mentioned in the lecture slides of the course Coordination of Real-time Logistics by Mr. Negenborn at the Delft University of Technology (Negenborn, 2015), come into play; asynchronous, synchronous, parallel, serial, single iteration and multiple iteration communication. Figure 3.8 shows these aspects in a single figure.
The first choice concerns asynchronous or synchronous communication. For the asynchronous approach signal timing of agents is not required. The information is send in an arranged structure and if both agents agree the communication can take place. It is important that each stream of data contains a start and stop bit to clarify when a data block starts or stops. Advantages of asynchronous messaging are system flexibility, higher availability and that the system does not completely fails if one of both ends does not work properly. If one side of the communication stream is down, data can still be send from the other side. Drawback of asynchronous messaging is the relative slow transmission due to the amount of bits and gaps and lack of immediate interaction. As long as one system is sending information without instant reply of the other system the overall communication remains one-sided without interaction. Contrary, for synchronous transmission signal timing is required as data blocks are send according to a specified pattern. Specific for this way of transmission is that there are no communication gaps as each bit is send one after the other. The receiver should count the bits and accurately reconstruct the bytes. In this way the synchronized transmission is faster than the asynchronized as, without the need for control bits, fewer bits have to be transmitted.

Furthermore, a choice between parallel or serial transmission should be made. As can be deduced from figure 3.8 parallel transmission means that communication can take place in both ways at the same time. For serial transmission data is only transmitted in one way at a certain time. With respect to serial transmission the parallel communication is significantly faster. But the parallel communication does also mean that more complex processing systems are required to keep up with the high information flow. Concerning the subsystems of an AGV control architecture the parallel scheme would mean that all systems or agents simultaneously perform a local step, exchange information and then solve the next step (Negenborn et al., 2008). For serial schemes it is stated that only one agent at a time performs a local step, sends information to the next agent, after which this next agent performs a local computation step and sends information to the next agent. Only after all agents have made a local step, the next round of local steps is started.

Lastly, the choice for single iteration or multiple iterations should be made. This choice concerns the computation information send within the data blocks. With single iteration the subsystem or agent only transmits information based on calculations once. For a multiple iterations scheme the same communication within a data block is repeated several times with possible updates and adaptions. As result application of multiple iterations is more complex and computationally more demanding, but on the other hand more accurate and complete with respect to provided system state information.

Message board
Another option with respect to the communication within the system is using a message board in the sense of a single or multiple databases. A competent example of this communication structure is the communication concept of the recently developed Robot Operating System (ROS). Each agent in a subsystem, or in ROS terms node, communicates via an anonymous publish and subscribe mechanism to the so-called topics in the database. This communication is regulated by a master that overarches the agents in that subsystem. The master is thus mainly a software structure that coordinates the data streams requested and send by the agents. As soon as a request from an agent is send to the master it requests the required topics in the database. Subsequently, it passes on the data to the related agent. These database topics are defined as sections in the database where related data is stored. Thus whenever data becomes available the master of the subsystem sends it to the database under such a specified topic. Agents that require the stored information subscribe to that same topic via their particular subsystem master so that the information can be retrieved via that master at any moment in time. Within the overall system clusters of subsystems can be created with multiple databases that are only available for these subsystems if the system would become so extensive for just one database.

With the use of these message boards asynchronous communication is made possible without complex communication protocols and processing algorithms. This reduces development effort and promotes flexibility and modularity (ROS). Besides that, different data broadcasting frequencies do not matter as each component can publish data according to its own recurrence while the receiving end only uses the required part of the data. However, this concept does require databases that allow very fast data changes.
and exactly defined data topics to accommodate extensive control structures. Furthermore, redundant information flows can be present in the system as the sender can still store information that is not used or required by the receiver.

3.4 Concluding remarks

This chapter basically elaborated on three subjects with respect to design possibilities for the control architecture; global control architectures, subsystem collaboration and communication. As potential system structures the following ones have been discussed:

- (De)centralized control
- Distributed control (with direct communication between controllers)
- Coordinated control
- Multilevel control

The CC was characterized by one component that is designated as controller and is responsible for managing all other components while in decentralized control the lower level components operate on local data, thus not on instructions from higher levels, to accomplish a global goal. Distributed control was characterized by having multiple controller loops, in which autonomous controllers are distributed over the system. Within the DC a distinction can be made between direct and indirect communication, of which an illustration is given in figure 3.2 and figure 3.3. Furthermore, it was stated that a MAS is a sub class of DC consisting of autonomous, interacting and more or the less intelligent data processing programs called agents. The third system structure was the coordinated control characterized by a coordinator with conditioned but independent subsystems. The coordinator and subsystems each have controllers, but only the coordinator can influence the subsystems and not the other way around. Lastly, the multilevel control came to discussion being characterized by having multiple levels in which each subsystem is connected to the level below and above it. Visualizations of these control structures and a short comparison with choice principles can be found in subsection 3.1.1.

With these global control architectures in mind the subsystem collaboration was discussed. This collaboration could be classified as coupled or decoupled. The coupled method implied that the overall system is considered as a composite system to which classical single-vehicle motion planning algorithms are applied. Contrary, the decoupled method meant that the subsystems carry out their assignments without considering data from the other presents subsystems. As second aspect in the collaboration a cooperative and non-cooperative system was considered. In the cooperative manner the subsystems do not function with only a self-enforcing perspective, but also take into account the states of the other subsystems. Thus, as opposite, the non cooperative method means that the subsystems only function with maximizing their own objective without taking into account the states of the other subsystems.

The last subject was direct or indirect communication within the system, which basically came down to the push and pull principle and direct or message board communication. The push principle indicated the sender pushed data towards a known receiver. The receiver can then accept or discard that data. The pull principle indicated a receiving component commencing data transfer by approaching the sender with a request. Besides these principles, a distinction was made between direct and message board communication too. For direct communication data is exchanged between components without an intermediate physical platform that facilitates this messaging. Within this direct messaging six schemes came into play. The six options were defined as asynchronous, synchronous, parallel, serial and single or multiple iteration communication. Visualization of these concepts can be seen in figure 3.8. Instead of direct messaging a message board was also suggested as option. In this case an intermediate physical platform is present to support exchange of data via this platform. Important advantage of this message board principle is the relatively simple allowance for asynchronous communication. Stated by ROS (ROS) this could reduce development effort and promotes flexibility and modularity.
Thus, it can be stated that there are a lot of options to construct an appropriate control architecture for the ALTOS. A distinction is made between the basic overall control structures, collaboration between subsystems and possible communication configurations. Regarding the overall control structures the centralized, decentralized, distributed, coordinated and multilevel control architectures were discussed. Afterwards, the coupled, decoupled, cooperative and non cooperative collaboration methods were reviewed. At last, the push and push model and direct and message board communication were analysed. With a clear overview the report now continues with concept choices in Chapter 4 to establish a substantiated basis for the final control architecture design.
4. Concept Choices

In this chapter choices are made on the previously described control architecture options with for each choice additional and more extensive information. The chapter maintains the same structure as Chapter 3 starts with a choice of global control architecture in section 4.1. Subsequently, section 4.2 elaborates on the preferred subsystem collaboration and section 4.3 on the communication protocols of the system. Each choice is supported by arguments to have a clear thread in the line of development. Finally, in section 4.4 consists of the concluding remarks.

4.1 Global control architecture choice
The current centralized system becomes computationally too complex to allow for future improvement that enhance the system performance. The action of the AGVs are still determined and carried out via a chain of actions that significantly slow down the process. Furthermore, these interconnected series of actions resulted in a system that is relatively inflexible and not robust as unexpected actions will lead to slow recovery. As each operating MAGV in the system has different assignments with potentially varying speed, directions and trajectory length it is of importance that their control can be performed in parallel. As previously mentioned in subsection 3.1.1 parallel control is possible in CC. However, in this concept it is still required that the single supervisory controller instructs every parallel functioning subsystem while keeping track of the data stream of all these subsystems. This could result in a processing overload for that particular controller. The DC architecture provides an appropriate solution for this dilemma as the computational effort will be divided over multiple subsystems. Besides that, the distributed control has the characteristic of being easily expandable and scalable, which would allow for potential future extensions. Furthermore, the modularity of the distributed architecture results in a better robustness of the system. In Comparative Analysis of Robustness of Centralized and Distributed Network Route Control Systems in Incident Situations (Hawas et al., 2012) it is experimentally shown that the robustness of a distributed control architecture is more excelling than that of a centralized system, which supports this statement.

To further meet up with future implementations on autonomous and intelligent level it is also chosen to use the MAS approach, described in subsection 3.1.1, with multiple levels. A MAS structure can even further improve scalability, flexibility and also enhance self-configuration, fault tolerance, emergent behaviour of a distributed control architecture. Moreover, it allows massive parallelism and reduces costs (Monostori et al., 2014). These last two advantages could be of particular interest with regards to implementation in real industries. Previous mentioned drawback is the complex theory behind the application of MAS. Yet, several modelling frameworks and semi-formal languages are nowadays available to design MAS based systems, which are summarized in Cooperative Control in Production and Logistics (Monostori et al., 2014).

Although it does not belong to the current possibilities, self-learning algorithms could later on be implemented in these system agents. The combination of multi-agent and distributed system provides the opportunity to use these future improvements at the moment they become available for the applied techniques in the AGV lab. These learning agents could improve the system starting with a general setup that could be applied to any basic environment. In that way it becomes possible to create a general control architecture that could be used under any circumstances as long as the system is provided with the right amount of time to adapt. At the moment of writing this report this is still a positive prospect, but with the use of MAS in the current architecture the foundation has been laid to extend the system with this innovative aspect as soon as it becomes available.

As last implementation in the control architecture a choice is made to partly use the coordinated control theory and apply it to a combination of the scheduling, dispatching and routing subsystems. These subsystems basically run on a combination of data of all of the system components. This coordination
thus restricts the actions and behaviour of the agents in lower levels so that the overall objective of the system will be met. To prevent over processing of the coordinator controllers it is of importance that the data stream in and out of this coordinator is limited to the absolute minimum. For example, the information required by the MAGV to operate according to its predetermined trajectory is not necessary for the coordinator to know. On the other hand, it is of importance that it knows the exact location of the MAGV and its future states. Therefore, this information exchange structure should be precisely defined in the design of the control architecture in the follow-up of this research.

4.2 Subsystem collaboration

The next choice to be made is whether to implement a coupled or (partly) decoupled subsystem network. As the navigation of AGVs is highly dependent on the state of every single AGV the system can only function if it works integrated. In that sense it is chosen to use a coupled subsystem architecture where the agents are able to exchange data with each other. The problem of over processing described in the previous section reappears in the consideration to fully couple the network or only use coupled segments. In conclusion of the typical decoupled coordination approaches mentioned in Decentralized Control of Multi-AGV Systems in Autonomous Warehousing Application (Draganjac et al., 2016) a similar structure is proposed for the AGV lab control architecture. This implies that parts of the system, in particular the coordinator and real time navigation action system of the MAGV, are only loosely coupled. The required operational actions to navigate the MAGVs can be carried out without direct real-time communication with the combined coordinator systems routing and transport planning. Communication between these two systems and the operational controllers of the MAGVs should be sufficient when only the predetermined trajectories and necessary updates are broadcasted.

The next deliberation concerns the cooperative or non cooperative control approach. Using a non-cooperative structure would result in a less demanding communication and computation scheme. Moreover, it would mean that every MAGV would operate according to the best possible solution for that particular one at that specific time. However, with this individual based principle applied it is guaranteed that the MAGVs have crossing trajectories resulting in deadlocks or collisions. Therefore, the system is only fully functional if the subsystems work on a cooperative basis with not only their own objective in mind but also that of the other MAGVs in the system. In Cooperative Control of Dynamical Systems (Qu, 2009) it is stated that cooperative and pliable behaviours are desirable for engineering systems such as autonomous vehicles, but to apply both robustness is also essential. As continuation on that statement it is declared that to obtain a cooperative, pliable and robust system the subsystems should have the following aspects:

- Trajectories of all subsystems in a cooperative system move collaboratively toward achieving a common objective, which can be referred to as equilibrium points. Yet, for this autonomous system this equilibrium point may not be selected in advance. Rather, they will depend on initial conditions, changes of system dynamics and influences from the environment.
- Associated with a cooperative system, there should be a network of either sensors or communication links or a mix of both. The sensing/communication network provides the means of information exchange among the subsystems and, unless sensor control or a communication protocol is explicitly considered, its changes can be modelled over time as binary variables of either on or off. Again feedback patterns may not be known in advance, which means that not individual subsystems must satisfy controllability but the network systems must require cooperative controllability too.
- The cooperative approach generally evolves in a dynamically changing environment, which has direct influence on the status of sensing and communicating. Another major impact of the environment is the geometrical constraints on the manoeuvrable sub-space. In the process of achieving a common goal, all subsystems must be robust in the sense that their trajectories comply with the changes in environment.
• Motion by each of the subsystems may also be subject to constraints in kinematics or dynamics or both. As result, control design must be pliable in the sense that the resulting motion trajectories comply with these constraints.

So, it is essential that these factors are taken into account in when using a cooperative scheme in the design of the control architecture for the AGV lab. Therefore, besides the cooperative setup, pliable and robust subsystems are required to grant proper functioning. A system is considered pliable if all the constraints are met and if all the network-based controls conform with the information flow in the network. Robustness is achieved if the same system also complies to changes in its physical environment. In the case of AGVs this mainly means that each of those can avoid static and dynamic obstacles including each other.

In the continuation of Cooperative Control of Dynamical Systems (Qu, 2009), it is also stated that the key to these above described requirements is multilevel control and autonomy. The choice for autonomy is already made with implementing the MAS setup, but the multilevel approach should be added to the chosen distributed coordinated system of the previous section. With regards to the control of heterogeneous vehicles the following three levels are proposed as key in this multilevel structure:

• Vehicle-level autonomy: Navigation and control algorithms designed and implemented such that each of the vehicles is pliable to its constraints, is robust to environmental changes and is capable of best following any given command signal.
• Team-level autonomy: Cooperative control algorithms designed and implemented to account for an intermittent sensing and communicating network to achieve cooperative behaviours. Through the network and by means of a virtual vehicle, an operator can adjust the status of cooperative behaviour.
• Mission-level autonomy and intelligence: High-level tactical decisions are enabled through human-machine interaction and automated by a multi-objective decision-making model with learning capabilities.

These levels will be taken into account in the design consideration of the distributed, coordinated and multilevel control architecture with cooperative subsystem behaviour.

4.3 Communications

The first choice with respect to the communication structure for the control architecture is related to the push and pull principle. Currently, it is noticed that mainly the push principle is applied in the exchange of information. Each of the subsystems sends their acquired information straight to the components connected to it. Due to different processing frequencies of the subsystems this results in overflows of information with the consequence of processing delays or backlog of data. As most of the systems process the data in sequence of arrival, updated information will be placed on hold if previous data is not yet processed. By the time this new data will be taken into account it will be outdated too, which implies that the system will be based on less real-time data eventually. Therefore, it would be auspicious to implement the data pull model in the design of the control architecture. Still, as the principle of pull works with a request of data before it is send it also has an implicit delay in transfer. Besides that, almost similar to the possible data overflow in the push model, the pull model could result in an overflow of requests at the sender side.

Solution to the above disadvantages can be found in a hybrid model as described in A Model of Computation with Push and Pull Processing (Zhao, 2003). In this model an event channel is introduced with which the the sender and receiver interact. The sender pushes the data to this event channel while the receivers pulls it from the same channel. It is also stated that if the data supplier and consumer operate at different rates, the channel may need an unbounded queue to store the events. Other option if that is not practical, is implementing some dropping mechanism or blocking when the queue is full. A visualization of this model is shown in figure 4.1.
With this event channel the choice for direct or message board communication for inter system interaction is implicitly made. This event channel should function as message board for the subsystems of the AGV control architecture. Within a subsystem the communication can still be carried out by direct messaging between the components, which are in this architecture the agents and masters.

An important aspect to be taken into account is the required two-way data exchange for most of the subsystems. A communication structure as developed by the ROS foundation (ROS) is considered as the most suitable option in that case. The publish and subscription principle with topics, as described in subsection 3.3.2, allow for asynchronous communication in two ways. Each of the subsystems can publish data according to their own processing speed while subscribing to data from other subsystems at the same time via the subsystem master. The data that is pushed and pulled from and to these topics is formatted into charts with rows and columns that comprise the necessary data. Fundamentally, it thus comes down to a push of data from the agent to the master, a push of data from the master to the database, a pull of data from the database to the master and a push of data from the master to the agent.

This pushed and pulled data is formatted into different charts that are required by the agents for their data processing. With these charts delivered to the agent via the subsystem master that agent can process the data into its own output chart. This output chart is then stored in the database so that it can be used by other agents in the sub- or overall system. Table 4.1 shows such a chart, with for each column the column title and an example of data that is required for the agent that requests this chart.

Table 4.1: Data chart example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>23</td>
<td>10</td>
<td>1</td>
<td>360</td>
<td>HAZ</td>
<td>1</td>
</tr>
</tbody>
</table>

Remaining is the data delay complication of this structure. Therefore, it is of excessive importance that precise and clear communication protocols are set up. The communication model of ROS is based on the message interface description language (IDL) is a good instance to base the MATLAB® codes on. As alternation on the model in figure 4.1 the model with the ROS structure and database as message board implemented is shown in figure 4.2.
4.4 Concluding remarks

With the basic concepts regarding possible control architecture structures mentioned and choices made on which to use, the fundament is laid for a proper design. The first major choice resulted in a preference for a distributed system instead of a (de)centralized one. To be able to efficiently and autonomously operate this distributed system the concept of MAS will be implemented with future intelligent possibilities in mind. The subsystem in the strategic level functions as coordinator for the tactical level and the subsystems in the tactical level as coordinator for the operational level to restrict the calculations and actions.

Furthermore, the option to partly couple the total network is made. These coupled subsystems also collaborate on a cooperative basis. As mentioned in section 4.2, research and literature study showed that cooperative behaviour is desired for real-time navigation of AGVs. Result of implementing this cooperative behaviour is that the multilevel structure, that was already chosen, is now also necessary for proper segregation of components. Thus a multilevel structure is used in the design as well.

Lastly, the described hybrid push and pull mechanism is selected to meet up with the communicational demands of such a system. The best suited option for this hybrid structure is determined to be a message board with a publish and subscription mechanism as described by the ROS Foundation (ROS) that is used for inter subsystem communication. With a well defined IDL the communication network of the system can process in an asynchronous and close to real-time manner. The communication within subsystems, thus between the agents and masters, is done with direct interaction is the sense of short messages and data exchange.

The remainder of this report elaborates on the design of a distributed, multi-agent, coordinated, multilevel system with coupled and cooperative subsystems that communicate via a message board.
5. Global Concept

Now that choices regarding the architecture have been made a global setup can be designed with main subsystems and their interconnections. Therefore, section 5.1 first elaborates on the subsystems and their required information that are present in the architecture. Section 5.2 follows with the communication aspects of the concept. As combination of section 5.1 and section 5.2, section 5.3 shows the fully integrated architecture with brief summary. Finally, section 5.4 concludes the chapter so that the detailed design with communication protocols and time spans can be discussed in Chapter 6.

5.1 Subsystems and database

The required subsystems of the control architecture design are discussed in this section. Each of the subsystems fit within one of the three levels as described in subsection 2.1.2 and can be linked to one of the functions that are present in industrial application of AGVs and the AGV laboratory. Furthermore, each subsystem consists of one or more agents, which have been described in subsection 3.1.1, that carry out the calculations for that part of the system. Based on the ROS setup, described in more detail in section 3.3, the present agent or agents in a subsystem are assigned to an central master. This master ensures that structured communication streams leave or enter the subsystem to and from the message board. This communication streams consist of the formatted charts as described in section 4.3. For each of the charts mentioned in this section an example is provided in subsection 5.1.8. Figure 5.1 shows the present subsystems in the control architecture, that is proposed in this chapter, integrated in the three level pyramid with the strategic, tactical and operational level.

![Control architecture subsystems structured in pyramid layers](image)

**Figure 5.1: Control architecture subsystems structured in pyramid layers**

The input subsystem takes care of the processing of the load plan that is submitted via the human interface. If the system is extended to a dynamic instead of a static one, where the scheduling, dispatching and routing is based on actual operation performance, the input subsystem also processes data from the operational level to prepare data input for the tactical level. In the tactical layer the scheduling, dispatching and routing subsystems take care of the time scheduling of the load plan, allocation of AGVs to scheduled jobs and determination of trajectories for dispatched AGVs respectively. In the operational level the real time operations are realized by the OptiTrack™, AGV and monitoring subsystems. The OptiTrack™ subsystem processes data from the cameras, the AGV subsystems directs the physical AGV systems and the monitoring system tracks the operations to allow for application of a dynamic system in future application.
With each of the subsystems is briefly described and allocated to one of the three levels of the overall system, the remainder of this section elaborates on each of the subsystems and the agents that are present within that subsystem. The basic processes and required data of these subsystems are discussed. To complete the control architecture, the database setup, functioning as message board, is elaborated on after all the subsystems have been clarified. Data chart structures mentioned in the subsystem descriptions can be found later on as tables in the subsection 5.1.8.

5.1.1 Input subsystem

The input subsystem functions in the strategic level of the control architecture. In essence the strategy of the system is determined in this layer with preparing the data for the scheduling, dispatching and routing subsystems in the tactical level. The basic input of the input subsystem is a load plan that is prepared by the operator of the TOS. An example of such a possible load plan is provided in table 5.1, but different operators can provide different load plans with varying information. In this example an end time of 13:00 is provided for each of the containers. As this end time is often similar for the complete load plan it could also be coupled once to the load plan file instead of a specific column in the table. Furthermore, it is assumed that the load plan has the load instructions structured in the order that the tasks should be carried out. If this is not the case a column should be added to display the order of each operation.

<table>
<thead>
<tr>
<th>Container ID</th>
<th>Pickup Location</th>
<th>Drop off Location</th>
<th>End time</th>
<th>Cargo Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>YC_3</td>
<td>QC_1</td>
<td>13:00</td>
<td>HAZ</td>
</tr>
<tr>
<td>2</td>
<td>YC_2</td>
<td>QC_2</td>
<td>13:00</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>YC_4</td>
<td>QC_3</td>
<td>13:00</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>YC_3</td>
<td>QC_4</td>
<td>13:00</td>
<td>HAZ</td>
</tr>
</tbody>
</table>

For a completely statically orientated system the load plan can be processed by one agent, specified as the task agent, that sends the input to the database so that it becomes available for the subsystems of the tactical level. However, to allow for dynamic operations in the future the planning subsystem is extended with two more agents that process data from the monitoring subsystem in the operational level. These two agents are specified as the AGV availability agent and the priority agent. In total the subsystem then consists of three agents that process data. These three agents are directed by the central input master that regulates the data stream in and out of the subsystem. The general setup of the input subsystem is shown in figure 5.2.

![Input Subsystem – Strategic Level](image)

Figure 5.2: Input subsystem
**Availability Agent**

The availability agent is a data processing program, as described in subsection 3.1.1, that is used when the system is advanced from static to dynamic methods. It has the task to provide an AGV availability chart for the tactical level subsystems. Therefore, it should keep track of the available and occupied times of the AGVs in the system. It starts with an initial chart where all AGVs are available and proceeds to update that chart based on data provided by the dispatch and monitoring subsystem. In that way an estimation of available times for each AGV can be made allowing for a more precise scheduling and dispatching. The required information for this agent, I in figure 5.2, is:

- Dispatch chart, shown in table 5.6
- AGV actual location chart provided by the OptiTrack™ system
- AGV predicted trajectory chart, shown in table 5.11

As result the outgoing data of this agent is an availability chart, shown in table 5.2, with estimations for the end time and location of each AGV per task that is scheduled and dispatched. In that way the chart can be used to provide more accurate dynamic scheduling and dispatching.

**Task Agent**

The task agent processes the input of the human interface into a load plan with each task separated. With the data from the human interface stored in a structured load plan the task agent can more convenient process the data. With only the static method the input of the task agent is often similar to the output. But as different operators often work with different load plan inputs, the agent is still used to create a similar output regardless of the input structure. In that way it is not required to adapt the system in lower layers for different load plans. Yet, as dynamic methods are upcoming in the AGV lab it is chosen to implement a function that keeps track of the completed tasks. In that way it is possible to allow for dynamic methods in the tactical level as it is then known which task is processed, which is being processed and which is still to be processed. To be able to keep track of the progress of the tasks this agent also requires data from the monitoring subsystem in the operational level. Furthermore, the operational frequency of this agent depends on the method used. In the static case the operation of the agent can be initialized by a load plan input, while for the dynamic method the agent should have a regular operational frequency. The required information for this agent, II in figure 5.2, is:

- Human interface input, which is an excel load plan
- Task updates from AGV subsystem, shown in table 5.9

The output data of this agent is a task chart, shown in table 5.3. In this task chart the data from the human interface is structured into identified tasks with the container ID, container pick up location, container destination location, completion time and as potential extension cargo description. As addition for the dynamical methods a column is added to show whether a task is processed, being processed with estimated completion time or to be processed.

**Priority Agent**

The priority agent is, as the availability agent, a processing program that is used if dynamic methods are applied in the system. Based on delayed processes it can prioritize the tasks that become obstructively behind schedule. Besides that, it can implement a function to recognize tasks that need priority based on their transported cargo or human interface calls. This function is part of industrial application of AGV transportation, but not yet of the system in the AGV lab. As result of this addition it becomes possible to process transportation tasks that concern hazardous or degradable material before less critical cargo. Furthermore, it would become possible to process priority calls that are placed via the human interface. The priority agent is set up separately from the task agent to be able to have separated software programs when the system becomes more complex in future applications. The required information for this agent, III in figure 5.2,

- Task chart from task agent with dynamic task updates, shown in table 5.3
• Priority calls from the human interface

The output data of this agent is then a priority chart with task that have been assigned priority, shown in table 5.4. Contrary to the output chart of the task agent, this output chart only contains the task ID and reason for priority, which can be delay (DEL), hazardous cargo (HAZ) or human interface calls (HUM). This data can then be interpreted by the scheduling agent in the tactical level so that it can adapt its scheduling.

5.1.2 Scheduling subsystem
The scheduling subsystem is the first of the tactical control layer. In this subsystem the scheduling of the tasks from the task chart takes place. Previously the use static and dynamic method were announced, which in particular concerns this subsystem. As stated in Multi AGV Scheduling Problem in Automated Container Terminal (Jin et al., 2016) scheduling of AGVs can be divided in two groups; static scheduling and dynamic scheduling. Static scheduling assumes that operational times are definite and can be estimated accurately allowing for in advance calculations. Dynamic scheduling assumes that the operational times cannot be predicted in advance and change dynamically during operation. In the same paper it is stated that static scheduling is less convenient for large discrete systems with multiple uncertainties, but that dynamic scheduling is more complex and requires extensive computational effort. A combination of static and dynamic scheduling could offer a solution to both complications. The system in its most basic format schedules the task according to the static scheduling method with estimation based times. With this statically produced schedule the other subsystems in the control architecture can perform their tasks to ensure operation of the AGV system. However, as extension for the future system the dynamic scheduling method can be applied parallel to the static one. After the static calculation, the output can be updated with a dynamic calculation based on the data provided by the monitoring subsystem in the operational level. In that way the computational complexity of the dynamic method is slightly reduced while the system operation is based on the actual performance and not on solely estimations and assumptions. To allow for both the static as the dynamic scheduling the scheduling subsystem consists of a static scheduling and dynamic scheduling agent, both overarched by the scheduling master to direct the data streams. The general setup of this scheduling subsystem is shown in figure 5.3.

![Scheduling Subsystem – Tactical Level](image)

**Figure 5.3: Scheduling subsystem**

**Static scheduling agent**
The static planning agent receives the task chart via the database from the input subsystem in the strategic level. With this task chart the agent determines the most convenient and suitable schedule based on time estimations and system assumptions. In this way each task from the load plan is sequentially scheduled for the dispatch subsystem. Previous researches have proposed scheduling models that suite this static scheduling, which could be consulted for further information (Dkhil et al., 2013), (Rashidi et al., 2011), (Kim et al., 2004), (Zhang et al., 2005), (Lee et al., 2010), (Angeloudis et al., 2010), (Moussi et al., 2012), (Le et al., 2012), (Nguyen et al., 2009). The required information for this static planning
agent comes mostly from the fixed data chart that needs to be specified based on the used static or dynamic methods, I in figure 5.1, is:

- Task chart, shown in table 5.3
- Terminal layout from the fixed data chart
- Number of AGV from the fixed data chart
- Number of QCs and YCs from the fixed data chart
- QC and YCs operation time estimations from the fixed data chart
- Travel time estimations between any container pickup and drop location from the fixed data chart
- Buffer estimation times from the fixed data chart

- As future implementation the priority chart part that concerns cargo description, shown in table 5.4

With this data a static schedule can be produced as output by this agent, which is used in the dispatch subsystem to allocate the AGVs in the system to the scheduled tasks. It consists of six defined times: the start and end time for the YC, AGV and QC.

**Dynamic scheduling agent**

As mentioned above the dynamic agent encompass the dynamic scheduling that could be implemented in the near future. It in essence carries out the same actions as the static scheduling agent, but now with actual system performance data implemented. The dynamic scheduling agent receives the static schedule and updates these calculations if the process deviates from this static schedule. As the dynamic planning agent has calculations based on the actual, and not predicted, performance it should receive additional information about the ongoing processes from the strategic level as described in subsection 5.1.1. Previous researches have proposed scheduling models for this dynamic scheduling, which could be consulted for further information (Jing et al., 2006), (Jing, 2010), (Cheng et al., 2008), (Zheng et al., 2006), (Briskorn et al., 2006). (Briskorn and Hartmann, 2006), (Grunow et al., 2005). (Grunow et al., 2006), (Koster et al., 2004). The required information for this dynamic planning agent, II in figure 5.1, is:

- Schedule chart, shown in table 5.5
- Task chart with task progress updates, shown in table 5.3
- Priority chart with cargo descriptions and delayed process and human call updates, shown in table 5.4

The output of this agent is a schedule chart based on actual performance of the system in which deviation are accounted for. The dispatch subsystem can in this case more accurate dispatch the vehicles.

**5.1.3 Dispatch subsystem**

The dispatch subsystem concerns the task of allocating the AGVs to scheduled tasks. Again a distinction can be made between static and dynamic dispatching. Similar to the scheduling method the static dispatching allocates the complete task schedule before the actual operation starts. It does that based on assumptions and estimations. However, in continuation to the dynamic scheduling the dispatching could also be done dynamically to prevent less optimal AGV allocation. Thus it then bases the AGV allocation on the data provided by the monitoring subsystem to allow for more convenient dispatch. As mentioned before this is currently considered as a future implementation, but so it is a present agent in this subsystem. An even more convenient option that would require future research is to combine the scheduling and dispatching to obtain a more convenient time estimation in the schedule. With the current limitations in the AGV laboratory this combination is not further treated in the control architecture design of this research. So, the subsystem contains a static and dynamic dispatch agent with the central dispatch master to control the in and outflow of data. Figure 5.4 shows the setup of this dispatch subsystem.
The static dispatch agent continues with the schedule chart of the static scheduling agent. Based on the number of AGVs in the system allocates each scheduled task sequentially to one of the AGVs. As result an ordered task list with scheduled start and end times is provided for each AGV. Again as it is a static calculation, this is done for the entire schedule before the operation starts. Deviations of AGV operations are therefore not taken into account, which could result in a less optimal AGV task allocation. The required information for this agent, I in figure 5.4, is:

- Task schedule of the static agent, shown in table 5.3
- Number of AGVs in the system from the fixed data chart

The outgoing data is then a dispatch chart in which each AGV has a sequential order of tasks to perform. The start and end time of each of these tasks is provided in combination with the start and end locations. In that way the AGV exactly knows where to be in the system at what specific time.

The dynamic dispatch agent is thus a continuation on the output of the dynamic scheduling agent. As the dynamic scheduling agent schedules task based on real time operation the dynamic dispatch agent can combine this schedule with the actual availability times of the AGVs. With this principle it is possible to determine which AGV is the closed to a task start location so that the transport time can be further reduced. For now, this remains a future implementation that needs extensive research. Yet, if applied the required information for this agent, II in figure 5.5, is:

- Task schedule of dynamic agent, shown in table 5.3
- Availability chart from the availability agent in the strategic level, shown in table 5.2

As output this agent produces a close to real-time dynamic dispatch chart with each dynamically scheduled task assigned to an operational AGV. In this way each AGV receives a new task as soon as the previous one is finished in order to avoid unnecessary delays as result of deviation from the predetermined schedule and dispatch data.

The third and final subsystem on the tactical level is the routing subsystem. This system follows up the scheduling and dispatching agent by determining the trajectory for each task for each AGV. It calculates a set of coordinates that have to be followed by the AGV in the operational level. The multiple aspects that play a role in both physical as logistic routing of AGVs in container terminals can be divided according to the three level pyramid structure as follows:
• Strategic measures and decisions
• Tactical measures and decisions
• Operational measures and decisions

For consultation of the detailed aspects per level the literature study of Bentvelsen (Bentvelsen, 2015) can be consulted. What mainly comes to mind in this literature study is the essentiality of adaptable routing mechanisms in the shape of a dynamic subsystem. Another proclamation regarding promising routing concepts can be found in Dispatching, routing and scheduling of two automated guided vehicles in a flexible manufacturing system (Langevin et al., 1996). In this paper the authors integrate and solve, as first ones to their own knowledge, the scheduling, dispatching and routing functions. With regards to the conformity with the real-time and industrial application this options would result in the most suitable and precise overall system planning. In this way the scheduling can be based on actual start and end locations and trajectories of the AGVs. The operational time estimations would in that case be more accurate resulting is less system deviation. Yet, for simplicity and implementing possibilities it is chosen to initiate with only static routing in the control architecture fundament in which all possible trajectory options are predetermined. So as soon as the static schedule is produced and the AGV dispatch chart is available the routing subsystem can immediately add trajectory instructions for the operational level. But again with future advancements in mind it is chosen to also have a dynamic part of the subsystem. Therefore, the routing subsystem at this moment consists of a static and dynamic routing agent. Figure 5.5 shows the setup of this routing subsystem.

Routing Subsystem — Tactical Level

Static routing agent
The static agent is the calculation program that comprises of the last task of the tactical level. With the schedule and dispatch chart calculated it determines the trajectory specifications for the AGVs in the operational level. All of the available trajectories between the possible pick up and drop off point are predetermined and stored in the database of the system. This means that as soon as the dispatch subsystem is finished the trajectories can immediately be added to the information for the AGV agents, later explained in subsection 5.1.6. The required information for this agent, I in figure 5.5, is:

• Schedule chart, shown in table 5.5
• Dispatch chart, shown in table 5.6
• Task chart, shown in table 5.3
• Trajectories for each possible start and end location from the fixed data chart

The output data of this agent is a routing chart, shown in table 5.7, that can be used by the AGV agents to navigate the physical AGVs through the simulated terminal area. The trajectory chart in this setup contains coordinates that configure vectors used in the AGV agent to calculate the required navigation actions.
**Dynamic routing agent**

The dynamic routing agent cooperates with the dynamic schedule and dispatch agents to calculate the prescribed trajectories based on actual performance data. The trajectories in this case not predetermined and each trajectory is calculated by the agent based on the scheduled and dispatched AGV at that particular moment. The actual AGV situation in the system then influences the function that determines the trajectories. In this way a more precise and suitable trajectory is chosen per dispatched AGV. In the end this could result in more beneficial functioning of the overall system. Drawback of this dynamic extension is the remarkable computational complexity it entails. Thus if the dynamic routing agent is applied in the future the calculation abilities of the system should be extended significantly. The required information for this agent, II in figure 5.5, is:

- Schedule chart, shown in table 5.5
- Dispatch chart, shown in table 5.6
- Task chart, shown in table 5.3
- Predicted trajectory and zone data from the monitoring subsystem, shown in table 5.10 and table 5.11
- Non immediate collision chart, shown in table 5.13

The output data of this agent is a real-time updated routing chart, shown in table 5.7, based on the actual performance of the system. With these updated trajectories the AGV can precisely follow the prescribed vectors that fit the actual situation on the terminal floor.

### 5.1.5 OptiTrack™ subsystem

The first component of the tactical level of the control architecture is the OptiTrack™ camera system. Contrary to the previous components, this is an external subsystem that only provides measured data to the tactical database. The open area in the AGV lab is completely monitored with a set of infrared cameras from OptiTrack™. The infrared beams from the cameras are reflected by four corner reflectors and one randomly placed. With different patterns for each operating AGV the software can already identify the AGVs in the system. Furthermore, the position and orientation of the AGVs can be determined on the basis of the asymmetric placement of the reflectors. Currently, the OptiTrack™ system in the AGV lab consists of 13 cameras to cover each part of the operational area with at least three cameras as that is required to triangulate the exact location of the AGV. As the OptiTrack™ system only acquires data, processes it with use of an internal database and then sends it to the monitoring agent in the same operational level, the elements of this subsystem are entitled as components and not as agents. Yet, according to the principles of the ROS the subsystem still needs a master to regulate the information stream to the related subsystems. Therefore, the camera master is implemented to accommodate communication of the data provided by the OptiTrack™ to the database and thus other related subsystems. Figure 5.6 shows this external subsystem of the total control architecture.

![OptiTrack™ Subsystem – Operational Level](image)
As future alternative a visualization tool from ROS, the ROS Visualization (RVIZ), could also be used for the tracking of the AGVs. It provides 3D visualization of many sensors types and any Unified Robot Description Format (URDF) described robot. With the exact and fast 3D-visualization of RVIZ problems as sensor misalignment and model inaccuracies can be identified (ROS). For now, the advantage, use and implementation of this system in the AGV laboratory is left for further research.

5.1.6 AGV subsystem
The AGV subsystem consists of the separated AGV control agents. With each AGV having its own assigned AGV agent removal or addition of AGVs can be carried out immediately without changing the logistic functions. The AGV master manages the data requests by pulling the information from the database that is produced by the tactical level subsystems. The AGV master should be able to split the data blocks into specified topics for each singular AGV. The setup of this AGV subsystem is shown in figure 5.7.

**AGV Subsystem – Operational Level**

![AGV Subsystem](image)

*Figure 5.7: AGV subsystem*

**AGV agent**
The AGV agent is a program that processes the data provided by the subsystems in the tactical level. Based on the prescribed trajectories, carried out actions and data from the OptiTrack™ it calculates the required actions for the physical components of the AGVs. The actions determined by this AGV agent are communicated to the Arduino board, which send out the electrical pulses to adapt or maintain the state of the engines and servos. Main function in these AGV agents is the state estimation. With its own historic action data, the prescribed tactical level data and the data provided by the OptiTrack™ system, the AGV agent can produce a state estimation of its own actual trajectory. With this state estimation the internal controller of the AGV can determine the above mentioned physical actions. A more elaborate description of agents, with application in multi AGV terminals, can be found in *A multi-agent architecture for control of AGV systems* (Farahvash et al., 2004). The required information for this agent, shown in figure 5.7, is:

- Schedule chart, shown in table 5.5
- Dispatch chart, shown in table 5.6
- Routing chart, shown in table 5.7
- OptiTrack™ data
- Physical Actions carried out, which is acquired via an internal loop
As mentioned above, the output data is a chart of physical actions, shown in Table 5.8, that are sent to the Arduino board and subsequently the engines and servos of the AGVs as shown in Figure 2.5 and Figure 2.6. Besides this, as internal output, the determined actions are fed back into the state estimator of the AGV agent. As part of the dynamic version of the system, it also produces a task completion chart. A future output extension of the AGV agents could be a direct and integrated communication structure to communicate potential disturbances or collisions without the intervention of a database, a collision detection system, or monitoring systems. With the fundaments of distributed control in mind, this would be a considerable advantage in the real-time performance of the multi-AGV system.

5.1.7 Monitoring subsystem

The last subsystem, part of the operational level of the control architecture, is the monitoring subsystem. As the name suggests, this subsystem completely focuses on processing actual performance data. This data is then used by the strategic and tactical level to allow for dynamic scheduling, dispatching, and routing. For this subsystem, it is chosen to have three operable agents under the supervision of the monitoring master. The first agent is the zone agent, monitoring the zone occupation in the simulated terminal area. The second agent is the trajectory agent, which monitors the actual and prescribed trajectories to provide a prediction of the actual followed trajectory in the future. The third agent is the collision agent, which particularly focuses on preventing collisions of the operating AGVs. Within the present AGV laboratory system, this subsystem would not be required to operate, but when applying the proposed dynamic methods, it would become necessary. A first implementation could be done with only showing results of the monitoring subsystem to verify its functioning. If the subsystem is extended in such a way that the data is real-time and complete a coupling can be made with the strategic level via the database of the system. The monitoring subsystem has a setup as shown in Figure 5.8.

Monitoring Subsystem – Operational Level

![Monitoring Subsystem Diagram](image)

**Figure 5.8: Monitoring subsystem**

Zone agent

Essential for safety is the occupation of zones in the terminal area. A current development in the AGV laboratory is a controlling method that uses zone claiming. This concept basically means that the terminal layout, with specifically the driveways for the AGVs, is divided into bounded zones. Per zone, one AGV is allowed to be present, and the next one can only enter as soon as the previous one completely left the zone. Currently, the basic system is based on the principle that the AGV requests at the zone software whether a zone is free to enter or not. If the zone is free at that time, the AGV continues its path, but if the zone is occupied by another AGV, the entering AGV stops and waits till the zone is free. With this basic zoning structure, a safety margin is built into the system to prevent the AGVs from
getting to close to or even collide with each other. This setup could be maintained but with future dynamic methods in mind an advancement of the system would be desirable.

For dynamic routing this advancement would also be of particular interest as new and updated trajectories should not interfere with claimed zones. As future extension of the already existing system, as described above, the zone agent should therefore be able to produce a data chart with the time related occupied zones. Based on the prescribed trajectories and actual performance it should make an estimation of each occupation time per zone for a certain operational time horizon. The required information for this agent, I in figure 5.8, is:

- **OptiTrack™ AGV location and orientation data**
- Prescribed trajectory from routing agent, shown in table 5.7
- Predicted trajectory from trajectory agent, shown in table 5.11
- Schedule chart, shown in table 5.5

With this data the zone agent should be able to provide a chart with the predicted occupation times per zone, shown in table 5.10. Based on the zone division of the laboratory terminal area the process frequency of the zone agent should be adjusted. As for now this is again a future implementation it would require comprehensive research.

**Trajectory agent**
The trajectory agent has as task to keep track of the actual and prescribed trajectories to provide a prediction of the followed trajectories for the successive time steps. It communicates this data with the routing subsystem on the tactical level and the input subsystem on the strategic level. Deviations that are not negligible, based on parametric restrictions, and are not completely corrected within subsequent time steps would in this case be instantly noticed. With the dynamic methods in place in the tactical level a recalculation of the trajectories could then take place to enhance operational efficiency and prevent possible collisions later on in the system. The required information for this agent, II in figure 5.8, is:

- Physical action chart from AGV agents, shown in table 5.8
- **OptiTrack™ AGV location and orientation data**
- Routing chart from the dynamic routing agent, shown in table 5.7

The output data of this agent is a trajectory prediction chart, shown in table 5.11, for each of the operational AGVs. Similar to the trajectory chart of the routing agent this chart contains the vector coordinates, but now for the predicted trajectory based on the actual performance.

**Collision agent**
The third agent of this system is basically a safe line for the overall system with the AGVs in particular. As the control architecture is designed to let the system operate without interferences or collision probabilities this subsystem is an addition to ensure complete protection from system or AGV failure. With the data from the **OptiTrack™** system and the instructed actions of the AGV agents the collision agent is able to determine possible collisions. Based on the time horizon of these potential collisions the agent should send different messages to different subsystems in the control architecture. If an immediate threat is observed an emergency message has to be pushed to the AGV agents consisting of instructions for a partly or total shutdown of the system. In the case of a less immediate threat, but only a noticed interference later in the operation instructions should be send to the dynamic routing agent to adapt the trajectories of the AGVs. Without the dynamic methods in the system the non immediate collision threat could only be resolved by sending direct instructions to the AGV agents to adapt the AGV course or speed temporarily or permanent. Currently, basic research was done in the AGV laboratory to implement and enhance such a system structure. In the research report of Thijs Hogenboom (Hogenboom, 2017) more elaborate information can be found regarding obstacle avoidance including collision prevention. The required information in this system for this agent, III in figure 5.8, is:
• Zone occupation chart, shown in table 5.10
• Physical action chart from AGV agents, shown in table 5.8
• OptiTrack\textsuperscript{TM} AGV location and orientation data
• Predicted trajectories chart, shown in table 5.11

With this data the collision agent can produce the collision prevention instructions as described above. The output for an immediate threat is an emergency message chart for the AGV agents, shown in table 5.12. If it is not an immediate threat the collision agent produces a collision chart with the involved AGV IDs, time slot of the collision and both AGV locations or the specific zone of the collision, shown in table 5.13. This chart is than published for use by the dynamic routing agent.

5.1.8 Database

The database of the control architecture functions as message board, which originates from the ROS structure. With this message board it is thus possible to operate the subsystems at different frequencies allowing them to publish and request data in their own required rate. Based on the produced data by the agents of the subsystems a certain amount of charts, described in section 4.3, is demanded in the database. These charts are the topics to which the agents can publish or subscribe via the master of its subsystem. It is important data coding is established to ensure consistency in the data. Data columns of the tables that that carry the same data, such as the multiple use of the AGV ID, should have the same column title to arrange the interconnections. Due to this relational database setup different charts can be linked with each other by using a primary key. In this case the primary key is the Task_ID that is assigned by the task agent. With this Task_ID it is not necessary to store similar data in different charts, which means no redundant data is present. Thus for example, the task chart, schedule chart and dispatch chart can be linked with the Task_ID to have the full range of data available. Based on the the output data of the above described agents the database need to store 16 charts with agent data. Each of these charts is described below with its column titles and column data format, which can be integer (INT) or text (TXT).

Availability chart
This chart constructed in the strategic level by the availability agent can be requested or stored as the topic Availability_Chart. The structure of the chart is shown in table 5.2. Based on the dispatch method that is used in the system this chart can be used to statically, with estimates, or dynamically, with real-time operational data, dispatch the AGVs.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>86</td>
<td>10</td>
<td>Executed</td>
</tr>
</tbody>
</table>

Table 5.2: Availability chart

Task chart
The task chart is constructed in the strategic level by the task agent and can be requested or stored as the topic Task_Chart. The structure of the chart is shown in table 5.3. The T_End correspond to the time left till the end time of the load plan, see table 5.1, but now in the system time unit of minutes or seconds. The status can be pending, operation or complete.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>23</td>
<td>10</td>
<td>360</td>
<td>HAZ</td>
<td>Pending</td>
</tr>
</tbody>
</table>

Table 5.3: Task chart
Priority chart
The priority chart is constructed in the strategic level by the priority agent and can be requested or stored as the topic Priority_Chart. The structure of the chart is shown in table 5.4.

Table 5.4: Priority chart

<table>
<thead>
<tr>
<th>TASK_ID [INT]</th>
<th>Priority_Desc [TXT]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HAZ</td>
</tr>
</tbody>
</table>

Schedule chart
The schedule chart is constructed in the tactical level by the static scheduling agent and can be requested or stored as the topic Schedule_Chart. The six defined times are the start and end times of the YC, AGV and QC with T_3 and T_4 as specific start and end times for the AGV and T_6 always equal or less than the T_End from the Task_Chart. The structure of the chart is shown in table 5.5.

Table 5.5: Schedule chart

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>22</td>
<td>24</td>
<td>106</td>
<td>107</td>
<td>117</td>
</tr>
</tbody>
</table>

For dynamic operation the dynamic schedule chart is constructed in the tactical level by the dynamic scheduling agent and can be requested or stored as the topic Schedule_Chart. The structure of the chart is similar to that of the static schedule chart, shown in table 5.5. The only difference is that this chart is updated based on the actual performance of the system.

Dispatch chart
The static dispatch chart is constructed in the tactical level by the static dispatch agent and can be requested or stored as the topic Static_Dispatch_Chart. The structure of the chart is shown in table 5.6. Data about the container ID, pick up and drop locations and T_3 and T_4 can be obtained from the schedule and task chart.

Table 5.6: Dispatch chart

<table>
<thead>
<tr>
<th>Task_ID [INT]</th>
<th>AGV_ID [INT]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

For dynamic operation the dispatch chart is constructed in the tactical level by the dynamic dispatch agent and can also be requested or stored as the topic Dispatch_Chart. The structure of the chart is similar to that of the static dispatch chart, shown in table 5.6. Again the only difference is that this chart is updated according to the actual performance.

Routing chart
The routing chart for static operation is constructed in the tactical level by the static routing agent and can be requested or stored as the topic Routing_Chart. Based on the implemented method the coordinates for the trajectory, with X and Y, could be extended with the speed as coordinate Z. The structure of the chart is shown in table 5.7. In the static case the AGV follows the points defined by the X and Y coordinates for each time step with an average speed. Using the dynamic method, the speed can be adapted per time step resulting in a more dynamic trajectory. Important remark for this routing chart is that the orientation of the container is not yet taken into account. Extended research in this routing data and the most suitable setup is required to amongst others implement the orientation too.
For the dynamic method the routing chart is constructed in the tactical level by the dynamic routing agent and can be requested or stored as the topic Dynamic_Routing_Chart. The structure of the chart is similar to the one showed in table 5.7, but this one is also updated based on the actual performance of the system.

**OptiTrack** chart

The OptiTrack chart is constructed in the operational level by the OptiTrack agents and can be requested or stored as the topic Optitrack_Chart. The structure of this chart depends on the output data of this external system. The proper storage of data from this subsystem should be determined based on information provided by the producers of the system. More elaborate research is required on this subject.

**Physical action chart**

The physical action chart is constructed in the operational level by the AGV agents and can be requested or stored as the topic Physical_Action_Chart. For this chart the AGV master plays an important role in combining the data from each of the operational AGVs. Of particular interest is the number of time steps that is required to be present in the chart per AGV. For the other agents a set of the last 5 action instructions of the AGV agent could be enough to perform their calculations accurately. However, as this is a parameter that thoroughly has to be investigated and tested is it stated as a parametric boundary for now. The AGV master should combine each of this five action instruction charts to one chart with each AGV in sequence structured below the other one. The structure of the chart, inclusive example data for a combined output, is shown in table 5.8.

Table 5.7: Routing chart

<table>
<thead>
<tr>
<th>Time_Step</th>
<th>AGV_ID [INT]</th>
<th>AGV_ID [INT]</th>
<th>AGV_ID [INT]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[INT]</td>
<td>X;Y;Z [INT]</td>
<td>X;Y;Z [INT]</td>
<td>X;Y;Z [INT]</td>
</tr>
</tbody>
</table>

Table 5.8: Physical action chart

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>25</td>
<td>25</td>
</tr>
</tbody>
</table>

**AGV task chart**

This chart shows the finished tasks of the AGVs. Each time that an AGV finishes a task, it sends a chart with the task ID via the master to the topic. The chart is stored as the topic Complete_Chart, shown in table 5.9.

Table 5.9: Task complete chart

<table>
<thead>
<tr>
<th>Task_ID [INT]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

**Zone occupation chart**

The zone occupation chart is constructed in the operational level by the zone agent and can be requested or stored as the topic Zone_Chart. The structure of this of this chart is shown in table 5.10.
Predicted trajectory chart
The predicted trajectory chart is constructed in the operational level by the trajectory agent and can be requested or stored as the topic Predicted_Trajectory_Chart. The structure of this chart is basically the same as the trajectory chart of the routing agent but without the speed as can be seen in table 5.11. Yet, this chart is updated on regular basis based on the performance and deviations of the system.

<table>
<thead>
<tr>
<th>Zone_ID [INT]</th>
<th>Zone_Start [INT]</th>
<th>Zone_End [INT]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>60</td>
<td>63</td>
</tr>
</tbody>
</table>

Collision emergency message
The collision emergency message is structured in the operational level by the collision agent and consists of the AGV ID and the forced stop message. This message is not requestable but is pushed immediately via the monitoring master towards the AGV master via the database topic Emergency_Message. It is structures as shown in table 5.12.

<table>
<thead>
<tr>
<th>AGV_ID [INT]</th>
<th>Emergency_Message [TXT]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>STOP</td>
</tr>
</tbody>
</table>

The second potential output of the collision agent is a chart with possible interferences or collisions in the future operation of the system. This chart can be requested or stored as Collision_Chart. The structure of this chart is shown in table 5.13.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>78</td>
<td>2</td>
<td>4</td>
<td>16</td>
</tr>
</tbody>
</table>

Human interface chart
The human interface chart consists of the load plan. This is stored under the topic Human_Interface_Chart. With regards to industrial application, the specific format of this chart depends on the applied system by the terminal operator. For convenience in the laboratory this human interface chart can also be created in the format of the task chart, which could spare out a data preparation function. An example of this chart is provided in table 5.14.

<table>
<thead>
<tr>
<th>Container ID</th>
<th>Pickup Location</th>
<th>Drop off location</th>
<th>End time</th>
<th>Cargo Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>YC_1</td>
<td>QC_1</td>
<td>13:00</td>
<td>HAZ</td>
</tr>
<tr>
<td>2</td>
<td>YC_2</td>
<td>QC_3</td>
<td>13:00</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>YC_3</td>
<td>QC_2</td>
<td>13:00</td>
<td>-</td>
</tr>
</tbody>
</table>
Fixed data chart
Especially when the static method is used in the system estimations have to be made based on predefined data. Therefore, the fixed data regarding the AGV terminal area is stored in the Fixed_Data_Chart in the database. The structure and layout can be determined later based on the applied methods. The following data should be present in this chart:

- Number of AGVs in the system
- Average operational times of the QCs and YCs
- Number of QCs
- Number of YCs
- Average buffer times
- Travel times between location combinations
- Fixed route coordinates if this method is used

Register chart
As the agents need to register and unregister via the master of the subsystem to publish or subscribe to a topic, charts for these actions are also required. With a register and unregister function for publishing and well as subscribing this means that two charts, Register_Publish and Register_Subscribe, are needed that contain the data that is necessary for these registrations. With an unregister request the data from the charts is removed. This exact data and the layout of these charts is discussed in section 6.1, the detailed design of the input master.

With all these topics described the global database setup is now acquainted. A total of 16 topics is present in the database from which data can be requested or data can be stored to. Each of the tables has its own columns and rows, which are title with data strings. The masters of each of the subsystems make sure that the data stream reach the agents or database in a structured way. In this way it should be possible to have parallel data streams without interference.

5.2 Communication features
As already shown in the previous section a message board in the form of a database is used to communicate information between the subsystems via the masters. Still, the push and pull principles have to be applied to each of the agents and masters. Therefore, the following section will briefly elaborate on the applied push and pull communication in the control architecture. Each of the agents can push their produced information towards the databases at their own processing frequency while pulling the information that is required from other subsystems in the control architecture. Important is that each of the pull frequencies never exceeds the frequency of the pushed information that is required from other agents. These frequencies are set as parameters for now and have to be determined based on more elaborate research and testing per subsystem. Besides that, it is of importance that communication protocols are set up regarding the dropping of queued information. If the pushed information flow exceeds the pulled information flow of the same data queues will be build resulting in usage of the dated data. Therefore, an information drop protocol is required per database topic that ensures that old, no longer needed information is deleted from the topics. Furthermore, one exception to the pull strategy is present in the system. The emergency message, required to prevent instant collision, is pushed from the collision agent to the AGV agent via the database. The masters of these two subsystems should be able to recognize and process such a pushed emergency message.

5.3 Overall control architecture
Now that the subsystems, database and the external OptiTrack™ system have been discussed, the total control architecture is constructed by combining figure 5.1-5.8, which is shown in figure 5.9 on the next page. This control architecture shows the global setup of the ALTOS with all of its components presents. The three level structure with the strategic, tactical and operational level is included in the system. This makes it possible to relate the proposed control architecture to the industrial TOS part that is related to the AGV operations. The potential future dynamic components are present in this architecture.
Figure 5.9: Overall control architecture
5.4 Concluding remarks

In figure 5.9 it can be seen that the distributed system completely functions via the publish and subscription principle to the database. Each subsystem has its own entities that perform their task autonomously based on the data they request from the database. This specific distributed setting ensures subdivision of the complex calculation functions reducing the intricacy, which enhances the calculation speed. Each of the subsystems is divided over the three layers to structure the control architecture for the AGV laboratory according to the industrial application setup.

Important is to place a remark on the subsystems that have been added with potential future extensions in mind. Major improvement in the AGV laboratory would be a dynamically functioning system. In this system the subsystems would carry out their logistic functions based on actual system performance data. This would probably enhance the efficiency substantially. As the current setup of the AGV laboratory does not support these yet, further research and testing is required to apply such systems. To provide a fundament for this research the required entities have already been taken into account in the design of the control architecture of the research.

With the global setup defined the report now continues with a detailed description of the input subsystem shown in figure 5.9. It will be a continuation on the global insight that is provided in this chapter.
6. Detailed Subsystem

This chapter elaborates in detail on the input subsystem of the global concept described in Chapter 5. It will only discuss one subsystem in detail, as that one could then be seen as fundamental for the total implementation after this single part is verified, validated and functional. With the top down layer structure it is chosen to specify the input subsystem. Paragraph 6.1 presents the input master setup and paragraph 6.2–6.4 that of the three agents. Finally, paragraph 6.5 concludes the chapter with a link to the implementation.

6.1 Input master
The master of the subsystem allows the agents to interact with each other, without using extensive data communication strings. Stated by ROS (ROS) it is not required anymore to specifically state “Send this sensor data to that computer at 127.0.0.1", but an instruction in the sense of tell node 1 to send messages to node 2 is enough. Each of the entities under the supervision of the master is thus denoted as a node in the subsystem. These nodes then communicate via the publish and subscribe principle in which the master functions as lookup table for the data. The master is thus connected to the database to store the data constructed by the agents. The processing structure of the master therefore allows the data push as well as pull mechanism, of which the setup is discussed in section 5.2. In the remainder of this section the specific functions required for the master are discussed, which are partially based on information provided by ROS but adjusted to MATLAB® requirements (ROS). Specified application programming interfaces (API) and uniform resource identifiers (URI) as often mentioned by ROS are not taken into account as the ROS programming structure is not used.

Register and unregister as publisher
Each of the agents in the system should register and unregister as a publisher of data for certain predefined topics. This can be inquired with the Register_Publish or Unregister_Publish message string, which is recognized by the data scan function of the master. With initialization of the system the master should process the following data in order to allow the agent as a publisher:

- Caller_ID [TXT] – Specific ID of the agent as node in the system
- Topic [TXT] – Fully qualified name of topic to register
- Topic_Type [STR, XLSX, INT, TXT] – Data structure of the published data
- Frequency_Publish [INT] – Processing frequency of the agent

This data is send from the agent to the master and then stored in the register topic in the database, this topic is a chart with the above described data as shown in table 6.1

Table 6.1: Register_Publish chart

|-----------------------|----------------|------------|---------------------------------|------------------------|

After the system completed its task and is ready to shut down the publishers can unregister to the database via an unregister message to the master. The master then removes the agent from the register chart in the database.

Register and unregister as subscriber
Besides registration as publisher, the agent should also be able to receive data after registration as subscriber for the required topic. This can be inquired with the message Register_Subscribe or Unregister_Subscribe, which is recognized by the data scan function of the master. This registration and unregistration should also take place when the system initializes and terminates after completion of the
tasks. The data required for registration and unregistration as subscriber is the same as for publisher but now with Register_Subscribe or Unregister_Subscribe and the database topic Register_Subscribe.

**Subscribed data request**
The second major function of the master is processing data request from the agents under its supervision. The agents dispatch a data request to the master. The master at its turn reads the request and determines from which topic data should be accessed. It then pulls the data chart from the database and provides it to the specified agent. To pull the data from the right topic in the database the master should read the required data from the register chart in the database. The data request is done with the message ‘Req’, which is recognized by the data scan function of the master.

**Published data processing**
After successful registration the agent of the system starts publishing constructed data for its topics in the database. As shown in subsection 5.1.8 all output data formats are structured as charts. These output charts become available for the master, which should then place them under the specified topic in the central database to which the agent is registered. The publish application should be done with the message code ‘Sto’, which can be recognized by the data scan function.

It can be noticed that the input master can be identified as a function within the subsystems that regulates the interaction of the agents with the database. Each of the agents in the subsystems have been registered to the function and have subsequently be coded as nodes in the overall system. With this structured setup the internal system interactions can be controlled in a transparent and accessible manner. Addition or removal of subsystems, agents and connections is made uncomplicated in this way.

As illustration of the above described actions figure 6.1 and figure 6.2 show a block and state diagram of the input master. These illustrations have been based on precedent models presented in *A multi-agent architecture for control of AGV systems* (Farahvash et al., 2004), which has in essence a similar research purpose.

![Figure 6.1: Component diagram input master](image-url)
The system starts with a data scan in state S1. If this data scan is successful it can continue to state S2, state S3 and state S4 if the data concerns a register or unregister request, data pull request and data publish request respectively. If the scan is unsuccessful it repeats its actions in the same state.

The first system direction is the one after a register or unregister request. The system can then carry out the register and unregister function to proceed to state S7, while it returns to state S1 if the function declines. From state S7 the register or unregister action is verified, if this is the case the system proceeds to state S8, if not it returns to state S2. The last step of this system direction is then to send a confirmation message to the agent and return to state S1, but if this confirmation fails it returns to state S7.

The second direction is the one with a proceeding from state S4 to state S6 by preparing the data input for publishing to the database. If this preparation is declined the system returns to state S1. From state S6 the system can then proceed to state S7 by carrying out the database post function or return to state S4 if this process fails. After verification of the data storage the system proceeds to state S9 with a return to state S6 if this verification was unsuccessful. The last step is then to send a confirmation to the agent that the published data is posted to the database, if this fails the system returns to state S7.

The third and final direction with a data request by an agent starts starts basically in state S3. From state S3 the system can proceed to state S5 by preparing the request for a database data pull. If this preparation
declines the system returns to state S1. Then from state S5 the system can continue to state S7 by pulling
the required data from the database with a return to state S3 if this pull function fails. After the data pull,
a successful verification results in an advance to state S10 and an unsuccessful verification to state S5.
As last step the system sends data that is pulled from the database to the agent while it terminates the
process by proceeding back to state S1. For this process a return to state S8 is also applied in case of a
send failure.

With these three directions defined with the component and state diagrams the detailed structure of the
input master is clearly shown. Based on these two diagrams an implementation can be carried out in
MATLAB®. The remainder of this chapter elaborates with the same setup on the three agents that are
under the supervision of this master. The described data strings are used in the description of these
agents, but adapted to the in and output of these agents.

6.2 Availability agent
The availability agent is one of the calculation programs that is specifically designed for application of
dynamic methods in the control architecture. To operate this agent, the dispatch chart, AGV actual
location chart and AGV trajectory chart are required. With initialization of the system the agent thus
first registers for subscription, with the message Register_Subscribe, for the following topics:

- Dynamic_Dispatch_Chart
- Predicted_Trajectory_Chart
- OptiTrackTM_Chart

For this registering it provides the following data as mentioned in section 6.1:

- Caller_ID = [STRATEGIC_01]
- Topic = [Dynamic_Dispatch_Chart; Predicted_Trajectory_Chart; OptiTrackTM_Chart]
- Topic_Type = [SQL; SQL; SQL]
- Frequency_Subscribe = [INT in \( \frac{1}{ms} \)]

After completion of all tasks and finalization of the system the agent can unregister with the message
Unregister_Subscribe and the same data as shown above.

Besides the subscription, the agent also publishes the availability chart. With the message
Register_Publish the agent can register itself as publisher for the topic Availability_Chart. In this case
the same data as for registering for subscription is required but the topic differs:

- Caller_ID = [STRATEGIC_01]
- Topic = [Availability_Chart]
- Topic_Type = [SQL]
- Frequency_Publish = [INT in \( \frac{1}{ms} \)]

After completion of all tasks and finalization of the system the agent can unregister with the message
Unregister_Publish and the same data as shown above.

With the registering successfully carried out the agent can request data with the data strings ‘Req’ and
publish with the data string ‘Sto’, which is then identified by the input master.

Again for this agent a component and state diagram are constructed to have a clear overview of the steps
that need to be taken by this agent. Figure 6.3 shows the component diagram and figure 6.4 the state
diagram.
Figure 6.3: Component diagram availability agent

Figure 6.4: State diagram availability agent
The availability agent starts in state S1 with registering for the topics it need to be subscribed to and the topic it publishes to. If this register request fails it repeats its action. If the request is send out it proceeds to state S2 where it awaits a verification message from the input master about successful registering. If this verification is successful it proceeds to state S3, if not it returns to state S1. From state S3 the specific agent process starts. With a request for the specific data for the availability function the system continues to state S4, while it repeats its action if the request fails. Then in state S4 the incoming data from the input master is scanned for completeness, if it is complete the system proceeds to state S5, if not it returns to state S3. From state S5 the incoming data is organized for processing before it continues to state S6. If anything goes wrong with the data preparation a return is made to state S4. With the organized data the actual availability function is carried out to proceed to state S7, with a return to S5 with malfunction of the function. Subsequently, state S8 can be reached by organizing the function output data to the availability output chart with a return to sate S6 in case of failure. From state S8 the system has two states in can proceed to by sending the date to the input master. If the overall process is not finished yet, the system continues to state S3 to repeat the process with new data. If the overall process is finished and all tasks have been carried out the system proceeds to state S9. For both directions a failure means a return to state S7. From state S9 the system then advances to state S2 by requesting unregistering at the input master, which it repeats if the request fails. If a verification is received the system finishes its process by proceeding to state S1, if not a return to state S9 is made.

The system thus mainly consists of the registering and unregistering process with in between a process loop that results in the output availability chart. As long as the system is in operation that loop continues to repeat its actions. In that way, up to date data is provided by the availability agent for the other subsystems and agents in the control architecture, which could enhance the system efficiency.

### 6.3 Task Agent

The task agent is the program that functions as the main start point for the static system with a potential extension for application of the dynamic methods. To operate this agent, the human interface input is required and in the case of the dynamic version also the task updates from the monitoring system. This agent also registers with initialization of the system with the message Register_Subscribe for the following topics:

- Human_Interface_Chart
- Complete_Chart

For this registering it provides the following data:

- Caller_ID = [STRATEGIC_02]
- Topic = [Human_Interface_Chart; Complete_Chart]
- Topic_Type = [SQL; SQL]
- Frequency_Subscribe = [INT in $\frac{1}{ms}$]

As for the availability agent the task agent can unregister at the end of system operation with the message Unregister_Subscribe and the same data as for registering.

The task agent also registers and unregisters itself for publishing of the Task_Chart with the message Register_Publish and Unregister_Publish and the following data attached:

- Caller_ID = [STRATEGIC_02]
- Topic = [Task_Chart]
- Topic_Type = [SQL]
- Frequency_Publish = [INT in $\frac{1}{ms}$]
Remark for the task agent is the limited necessity of the operational frequency. As stated before in section 5.1 in the description of the task agent the operational frequency is only required in the case of dynamic functioning. For the static method it is better to use the load plan input as initializer for the task agent to carry out its function.

After the register and unregister action sequence, the agent can request and publish data with the ‘Req’ and ‘Sto’ message. Succeeding the component and state diagram of the availability agent figure 6.5 and figure 6.6 show the component and state diagram for the task agent respectively.

Figure 6.5: Component diagram task agent

Figure 6.6: State diagram task agent
This agent starts again in state S1 with registering for the topics it need to be subscribed to and the topic it publishes to. With failure of sending this registering message the system remains in state S1 to repeat its action. If the message is send it proceeds to state S2 to receive a verification of the registering. With a successful verification it then proceeds to state S3, with an unsuccessful one back to state S1. The main process then starts in state S3 with a data request and proceeding to state S4 or a repeat of the state S3 action. In state S4 the incoming data is then scanned for completeness, which results in a continuation to state S5 or return to state S3 if it is complete or not respectively. Subsequently, the data is organized for processing which results in an advance to state S6. If this data organizing somehow does not work the system returns to state S4. From state S6 the system can proceed to either state S7 or state S8 by carrying out the function that processes the human input into a task chart while returning to state S5 if this function fails. Depending on the static or dynamic methods state S7 is included in the system or not. If the dynamic method is applied, data about task completion, in the format Complete_Chart, is available from the AGV agents which is then recognized by the data scanning of this agent. In this case the system would proceed to state S7 after the human input is processed so that it can process the completed tasks before proceeding to state S8. If only the static method is applied and no task completion data is available, the system continues from state S6 to state S8 by only processing the human input. Then from state S8 the system organizes the output data of the functions to proceed to state S9. If this data organization fails it returns to either state S6 or state S7 depending on the which path the system took. With the data now complete the system can send out the task chart to the input master with proceeding back to state S3 if the overall process is not finished or forward to state S10 if the overall process is completed. From state S3 the process described above repeats while from state S10 the agent unregisters itself at the input master. With the unregister message transmitted the system goes back to state S2 to wait for the unregister verification, if the message fails it repeats its action. With verification in state S2 the system then completely resets by continuing to state S1, with again a return to state S10 in case of unsuccessful verification.

So again the main agent process is included between the registering and unregistering. With the static or dynamic methods applied in the overall system this agent has two options for the data processing. If only the static methods are applied the actions of this agent are limited to processing the human input. But if the dynamic methods are applied, the agent is also assigned with the task to keep track of the completed tasks. With this up to date task tracking a more accurate and dynamic scheduling is made possible. The agent and its potential additional dynamic function is thus important for future efficiency improvement.

### 6.4 Priority Agent

The last and third agent of the input subsystem is the priority agent, which again is an agent that can potentially be implemented to enhance the performance of the system. Besides its contribution to a more dynamic system, it thus also allows for priority calls from the terminal operator and priority assignment of certain cargo. The priority agent registers for subscription to the following charts:

- Task_Chart
- Human_Interface_Chart

Similar to the previous two agents the following information is required for this registering:

- Caller_ID = [STRATEGIC_03]
- Topic = [Task_Chart; Human_Interface_Chart]
- Topic_Type = [SQL; SQL]
- Frequency_Subscribe = [INT in $\frac{1}{ms}$]

With this data the unregistering can then also be done with the message Unregister_Subscribe.

The output data in the format of the Priority_Chart is then published after registering or unregistering with the message Register_Publish or Unregister_Publish and the following attached data:
The last step in the data communication of this agent is then again the message ‘Req’ and ‘Sto’. The state and component diagrams both are sort of similar as that of the task agent, but now mainly with different function titles. *Figure 6.7* and *figure 6.8* show the component and state diagram of the priority agent respectively.

*Figure 6.7: Component diagram priority agent*

*Figure 6.8: State diagram priority agent*
From state S1 till state S6 the procedure of the priority agent is similar to that of the previous agents. Subsequently, the system proceeds to state S7 by processing delayed tasks based on information of the Completion Chart provided by the AGV agents. In case of process failure, the system returns to state S5. From state S7 the system can then continue to state S8 by processing cargo descriptions into the priority task list with again a return to state S5 in case of incomplete processing. Thereafter, the system proceeds to state S9 by carrying out the last function of the agent that processes the priority calls placed by the operator. From this state again a return is made to state S5 if the processing fails. Then the system advances from state S9 to S10 by organizing the data into the output format of the Priority Chart. If the required data for this data restructuring is incomplete the system can return to state S8, S7 or S6 based on the missing data. As last, the system finalizes its process like the previous described agents via state S11, S2 and S1.

### 6.5 Concluding remarks

As remainder to the global control architecture in Chapter 5 this chapter elaborated on the more detailed design of the strategic layer input subsystem. With this detailed description more insight is obtained in the functioning of the overall process. The ways of processing within the agents are shown with the use of component and state diagrams with incoming and outgoing data strings attached. Important aspect of this detailed description is also the illustration of the input master. As this segment is of major importance in the control architecture a full understanding of this concept is necessary.

With the global and detailed description in mind the report now continues with a basic and preliminary implementation of the control architecture to test and prove the concept. Similar to only a detailed design for the one of the seven subsystems, the implementation is focussed on the input subsystem. In the near future the detailed design for the other subsystem can be masterminded based on the one described in this chapter. For now, the next step is a basic implementation of the input subsystem to establish a working fundament without deficiencies for a database and MATLAB® interaction.
7. Implementation

This chapter elaborates on the first fundamentals of the implementation of the control architecture described in Chapter 5 and Chapter 6. The file structures are limited to the most basic setup to provide a basic thread for future programming. First an outline of the implementation setup is provided in section 7.1. Subsequently, section 7.2 continues with a description of the database and the related most fundamental MATLAB® components. Then section 7.3 elaborates on the proof of concept comprising of the tests and results based on the implementation described in the previous two sections. Finally, section 7.4 concludes the findings of the implementation described in this chapter.

7.1 Implementation outline

To provide guidelines for the development of the control architecture described in this report, specifically Chapter 5 and Chapter 6, this chapter elaborates on the first fundamental implementation of the design. As it is assumed that the desirable order of development of a three-layer system, shown in figure 5.1, is from higher layers to the lower layers, this first implementation focuses on the strategic level. Furthermore, with the current system only based on the static methods the implementation comprises of the input master and the task agent only, both shown in figure 5.2. Figure 7.1 recaps these figures with the implemented layer encircled with blue and the implemented components with red.

![Figure 7.1: Implementation layer and components](image-url)
Besides the input master and the task agent, the database, for which the MySQL Workbench platform is chosen, is also required to complete the implementation. With the layer and components of the implementation now defined the physical setup in the AGV laboratory can be determined. As the main objective of this new control architecture is a distributed arrangement it is of importance to divide the database, the input master m-file and the task agent m-file over multiple computers in the laboratory. Besides that, the input master and task agent should be initialized after submission of the load plan. Therefore, a setup in the AGV laboratory is created as illustrated in figure 7.2, which also shows the required programs and extensions. The TCP/IP and JDBC illustrate the way of communication, which is explained in the remainder of the section. Figure 7.3 shows the corresponding main processes in a sequential order, with registering and unregistering of the agent not taken into account in this illustration.

**Figure 7.2: AGV Laboratory setup**

**Figure 7.3: Processes in sequential order**
The computer with the MySQL database has a server that starts running with starting up the computer. Via this MySQL server the MATLAB® program of the input master can call for data required by the task agent. Via the principle of TCP/IP, later elaborated on in section 7.2, the task agent communicates data with the input master. On its turn the input master then communicates with the MySQL Database to obtain or store data from the task agent. The actions of the input master and task agent will be initialized by a submission of the load plan in MATLAB® on the TUD278415 computer. The dotted line from the TUD278415 to the TUD278416 computer indicates that a TCP/IP connection is only used once and in one way to initialize the task agent. As only the static method applied in the strategic level this distributed structure can be considered as laborious, but with future system expansion and adaptions to more dynamic methods in mind it is beneficial to have a fundament that also supports these developments. The components and most fundamental codes are now briefly discussed in the next section.

7.2 Database and fundamental MATLAB® components

This section is an instruction manual for the future implementation of the total control structure as described in this report. First of all, a short introduction is given in the use of MySQL Workbench and the setup of the database with its charts. Secondly, the way of connecting to the MySQL database and the main functions of the input master, being data import, data storage and data removal are discussed. Thirdly, the elemental codes for TCP/IP communication between the task agent and the input master are elaborated on. Lastly, the code setup for the master and agent initializer with storage of the load plan in the database is provided to make sure that tests can be performed and results obtained.

7.2.1 MySQL database

The message board of the control architecture is chosen to be a MySQL database based on MathWorks® information (MathWorks), which can be connected with MATLAB®. The MySQL database is installed on the laboratory computer TUD278415 in combination with a MySQL server that is adjusted to incoming data transfers from the other two mentioned computers in the AGV laboratory. With starting up the computer this server starts running allowing incoming connections to the MySQL workbench database. The program MySQL Workbench is used to create the in subsection 5.1.8 described charts for the MySQL database. As the implementation for this research focusses on providing a fundament for the physical construction of the control architecture design, it is limited to the input subsystem only. Furthermore, based on the current capabilities of the laboratory components the implementation is limited to the static method. Therefore, the required charts for this implementation, described in section 5.1.8, are the following four:

- Register_Publish, see table 6.1
- Register_Subscribe, see table 6.1
- Task_Chart, see table 5.3
- Human_Interface_Chart, see table 5.14

The remainder of this section explains how to implement these charts in the MySQL database. Firstly, the database ‘agvdatabase’ is created as new schema, which functions as main directory for the charts of the control architecture. The charts of the control architecture can then be represented by tables that are stored in the agvdatabase directory. Figure 7.4 shows the agvdatabase schema with the three charts as tables under this schema.

![Figure 7.4: MySQL Workbench schema and tables](image-url)
The tables in the AGVDatabase are then created by assigning column names and data types. Figure 7.5 shows the setup of the Register_Publish table. The Register_Publish table has a first column called Register_Number as the first column needs an integer in MySQL. The other tables in the database have a similar layout, but with column names as described in subsection 5.1.8.

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>PK</th>
<th>NN</th>
<th>UQ</th>
<th>BIN</th>
<th>UN</th>
<th>ZF</th>
<th>Al</th>
<th>G</th>
<th>Default / Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register_Number</td>
<td>INT(11)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>Caller_ID</td>
<td>VARCHAR(45)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>Topic</td>
<td>VARCHAR(45)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>Topic_Type</td>
<td>VARCHAR(45)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>Frequency_P...</td>
<td>INT(11)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NULL</td>
</tr>
</tbody>
</table>

*Figure 7.5: Register_Publish table in MySQL Workbench*

This table thus represents the chart that can be consulted by the master in each of the subsystems. Extension of the database with the remaining charts from Chapter 5 can be done as described above.

### 7.2.2 MATLAB® - MySQL codes

In this section the most important MATLAB® codes are shown with brief explanation to create a basis for the implementation of the control architecture. Firstly, the database connection code is discussed followed by those for data import, data storage and data removal codes.

**Database connection**

The masters of each subsystem request database from the database that is stored on a segregated computer. MATLAB® has the command ‘database’ that is constructed for such a connection. However, to be able to use this command a software component is required to enable an application as MATLAB® to communicate to an external database. Particularly the open database connectivity (ODBC) driver or Java database connectivity (JDBC) driver are suitable to connect MATLAB® to a MySQL database. For this implementation it is chosen to use the JDBC driver, but ODBC is also suitable. As described by MathWorks® a connection between the database and MATLAB® can be made in two ways. The first one is with the use of the database explorer application of MATLAB®. The second is with a command line that can be used in the m-files to setup a connection at the beginning of the program. This second option is shown in figure 7.6 as in each of the master programs this connection command is necessary. The variables used in this figure are the ones to which the MySQL database on computer TUD278415 is adjusted to.

```matlab
dbname = 'agvdatabase';
username = 'AGVLab';
password = 'AGVLaboratory';
driver = 'com.mysql.jdbc.Driver';
dburl = 'jdbc:mysql://131.188.29.225:3306/agvdatabase?verifyServerCertificate=false&useSSL=false&requireSSL=false';
conn = database(dbname, username, password, driver, dburl);
```

*Figure 7.6: MATLAB® command to create a connection with MySQL*

The database name is specified as ‘agvdatabase’ where the username and password for access are currently set as ‘AGVLab’ and ‘AGVLaboratory’ respectively. The driver refers to the name of the Java® driver that implements the java.sql.Driver interface, which is specified by MathWorks®. The database URL contains the connection properties such as the server name, port number, database name and secure socket layer (SSL) specifications to potentially encrypt the communication. The server is assigned with the IP address of the TUD278415 computer and port number 3306 as the server and MySQL database are located on this computer and can be consulted via the opened port 3306.
Data import
Matlab has a build in command that can perform database operations on a SQL database file by executing a query statement, a MySQL database instruction, defined in MATLAB®. This command is defined as exec(conn, sqlquery). In this function conn is defined as shown in figure 7.6, creating the database connection. Sqlquery is then defined as the MySQL expression to execute a command in MySQL Workbench. The variable that is assigned with the exec function is called the SQL cursor. This SQL cursor can be described as a control structure that enables transversal over the data. After the execution of the database operation the rows of data from the open SQL cursor should be imported to a MATLAB® object. This procedure can be done with the MATLAB® command fetch. The last remaining step is then to assign the data of this created object to a variable in the format of a table in the workspace with closure of the object afterwards. Each of the above described steps and commands can be seen in figure 7.7 where the register_publish chart is pulled from the database and stored as table in the workspace.

```
sqlquery = ['SELECT * FROM `AGVDatabase`.Register_Publish '];
Publish = exec(conn, sqlquery);
Publish = fetch(Publish);
Register_Publish = cell2table(Publish.Data);
close(Publish);
```

Figure 7.7: Data import from MySQL to MATLAB®

Data insertion
With data provided by the agents the input master should also be able to store this data in the database of the system. For this action MATLAB® has the function insert and update, which requires the connection, the table name, the corresponding column names, the data to be inserted and in case of an update also a statement specifying which row to update. An example for this data insertion or update is provided in figure 7.8 with variables based on registering an agent in the input.

```
RegisterPublishColnames = {'Register_Number','Caller_ID','Topic','Topic_Type'...
'Frequency_Publish'};
RegisterPublishData = {Register_Number, char(Register_Publish_TaskAgent.Caller_ID),...,
char(Register_Publish_TaskAgent.Topic),...
(Register_Publish_TaskAgent.Frequency_Publish)};
RegisterPublishTablename = 'Register_Publish';

insert(conn,Tablename,TableColnames,TableData)

whereclause = ['WHERE Register_Number = 1'];
update(conn,Tablename,TableColnames,TableData, whereclause)
```

Figure 7.8: Data insertion or update from MATLAB® to MySQL

Data removal
The last function is the removal of data from the database executed by a command in the MATLAB® program. As soon as an agent wants to unregister itself at the master its data should be erased from the register chart. Thus similar to the data import a sqlquery object should be created which is subsequently executed in SQL Workbench. This query expression needs as input the table name in which the to be erased data can be found and a column entry of the specific row that has to be erased. This column entry can be defined as a number variable that is then dependent on the agent that wants to unregister itself. With this variable in place it is important to convert this variable from a number to a character so that it can be recognized in MySQL. Figure 7.9 shows the above described data removal sequence in a MATLAB® code based on the register_publish chart.
With these basic MATLAB® codes the actions of the master related to the database are defined. The command lines can be adapted and made variable based on the functioning of the system. In the m-files in Appendix B various application of these codes can be found.

7.2.3 TCP/IP communication
TCP/IP is a defined as a series of network protocols that can be used to allow communication between computers. TCP/IP is merge of the Transmission Control Protocol (TCP) and the Internet Protocol (IP). These protocols grant transfers of independent data package with software that arranges the packages in the right order. MATLAB® has an integrated interface for such TCP/IP communication between computers. It uses TCP/IP objects with the particular division of server and client. The server opens the TCP/IP object and sends out data, while the client opens the TCP/IP object and receives the data. The exact description of the components of this interface can be found in the documentation of MathWorks® (MathWorks®). Figure 7.10a-b shows an example of a server and client communication between the TUD278415 and TUD207330 computer, being server and client respectively. The basis of this TCP/IP communication example can be used for communication in the control architecture. Yet, it is important to test the achieved speed of the TCP/IP connection for each of the system components as the operational frequency is bound by the MATLAB® TCP/IP interface. Testing should determine whether basic TCP/IP communication between m-files is also suitable for lower layer subsystems.

```matlab
Register_Number = 1;
sqlquery = ['DELETE FROM register_publish WHERE ' ...
' Register_Number = ' num2str(Register_Number)];
Remove = exec(conn, sqlquery);
```

**Figure 7.9:** Data removal from MATLAB® to MySQL

The num2str and unicode2native command make sure that the message is converted to a long string of integer numbers that can be transferred easily with the fprintf function. In the client code these commands have thus been used exactly opposite to convert the message back to its original shape. Furthermore, the 0.0.0.0 in the server example means that incoming connections from all possible IP addresses are allowed while in the client example the IP of the server computer is provided. The communication protocols use port 55000 on both computers, so that port should be open for inbound

```matlab
Message = 'Success';
Message = num2str(unicode2native(Message));
ts = tcpip('0.0.0.0', 55000, 'NetworkRole', 'Server');
set(ts, 'OutputBufferSize', 1000)
fopen(ts);
fprintf(ts, Message);
fclose(ts);
```

**Figure 7.10a,b:** TCP/IP server and client example

```matlab
tr = tcpip('131.180.30.35', 55000, 'NetworkRole', 'Client');
set(tr, 'OutputBufferSize', 1000)
fopen(tr);
Message = fscanf(tr);
Message = native2unicode(str2num(Message));
fclose(tr);
```
and outbound data. Lastly, the output buffer size defines the amount of bytes that can be stored in the output buffer during a write operation. In this example it is set to 1000 bytes, but with transferring of larger data this should be increased.

### 7.2.4 Input subsystem initializer

The last important part of the implementation is the initializer for the input master and task agent. This initializer is constructed as a m-file on the same computer as the MySQL database that first stores the excel load plan in the database as `human_interface_chart` and then sends a message to the input master and task agent to start their operation. It is chosen to use the variable `System_Operation` to simulate the operation of the overall system. If this variable is set to yes, the provided load plan in excel is stored via the data insertion or update as described before followed by a start message for the input master and task agent. But if this variable is set to no the initializer program only sends a message to the input master and task agent that the system is not yet in operation. The basic idea of this initializer program is illustrated in figure 7.11, but the complete m-file can be found in *Appendix B*.

![Figure 7.11: M-file interaction with initializer](image)

A remark with this initializer system is that the functions of the task agent and input master are set to action with this start message, but not the m-file itself. MATLAB® has no possibility to implement an initializing system that makes the m-file run. So for now the input master and task agent should manually be started. As soon as both files start running they open a connection for 20 seconds to the TUD278415 computer with the initializer program. Then the initializer program should be started with the variable `System_Operation` on either yes or no. With the variable on no the master input and task agent display ‘System not in operation’. As soon as the initializer m-file is started with the variable `System_Operation` on yes the load plan is stored as `human_interface_chart` in the database and the input master and task agent start their functions till the task chart is created and stored in the database.

### 7.3 Proof of concept

With the general setup and fundamental system code components discussed this section continues with a proof of concept. This proof of concept shows the feasibility and practical potential of the designed control architecture. As the implementation for the proof of concept is bounded to the input subsystem in the strategic level the experiments are limited to the most basic data exchange possibilities. The results
of the experiments are evaluated with respect to predefined KPI’s. Based on this evaluation, recommendations related to the other subsystems are also provided. First each experiment is shortly described and afterwards the results are discussed.

7.3.1 Experimental setup
Each of the following tests is carried out on the computers in the AGV laboratory. These computers run on the windows operating system and are connected to the Delft University network via cables. The specifications of the computers and the programs they run can be found in table A.1 in Appendix A. Reproduction of the implementation and tests can be done on these computers or on a personal computer using 'localhost' as server and port 3306 in the connections commands. Using this reference means that the computer itself is used as server and client for the exchange of data. In that case the functionality of the main code is tested but the applicability to a distributed setting is not. Therefore, it is recommended to carry out the tests with programs divided over the computers in the AGV laboratory. Below five test are described that have been carried out to provide a proof of concept. The first three relate to the actual functionality of the concept, while the fourth and fifth mainly focusses on the performance in the sense of process speed. For each of these tests quantitative or qualitative KPIs have been defined. KPIs related to the functionality are primarily qualitative and the KPIs related to the performance quantitative.

Test 1 – Input/Output conversion
This first test relates to the main function of the input subsystem, converting the input to the output with via the steps as described in section 6.3. The load plan is considered to be the input of the input subsystem and the task chart the output, both described in subsection 5.1.1. A load plan as table 5.1 with ten rows, meaning ten different containers, is saved on the computer to be used in the system. Then the input master and task agent m-files are started, which means they open a TCP/IP connection for twenty seconds. Within these twenty seconds the initializer m-file has to be started. Then after running the initializer file, the task agent and input master should start their functions. As result a chart with the layout as table 5.3 should be stored in the MySQL database. This chart then contains the data of the load plan, but now converted to system data specifications and with a Task_ID to allow for relational database access in other system layers. As the task agent is the MATLAB® program that converts the load plan to the task chart, but does not have a JDBC driver installed it means that the data is successfully exchanged via TCP/IP and is stored by the input master program. For the input master on its turn the successful storage means that exchange of data with an external database is accomplished. So, in the end a complete input output conversion implicates that the TCP/IP as well as the external database connection was successfully made and that the system components thus function in a distributed manner. Table 7.1 shows the input load plan used for this test.

Table 7.1: Input load plan for test 1

<table>
<thead>
<tr>
<th>Container ID</th>
<th>Pickup Location</th>
<th>Drop off location</th>
<th>End time</th>
<th>Cargo Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>YC_1</td>
<td>QC_1</td>
<td>13:00</td>
<td>HAZ</td>
</tr>
<tr>
<td>2</td>
<td>YC_2</td>
<td>QC_3</td>
<td>13:00</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>YC_3</td>
<td>QC_2</td>
<td>13:00</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>YC_4</td>
<td>QC_4</td>
<td>13:00</td>
<td>HAZ</td>
</tr>
<tr>
<td>5</td>
<td>YC_1</td>
<td>QC_5</td>
<td>13:00</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>YC_3</td>
<td>QC_2</td>
<td>13:05</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>YC_5</td>
<td>QC_1</td>
<td>13:05</td>
<td>HAZ</td>
</tr>
<tr>
<td>8</td>
<td>YC_3</td>
<td>QC_1</td>
<td>13:05</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>YC_4</td>
<td>QC_3</td>
<td>13:05</td>
<td>HAZ</td>
</tr>
<tr>
<td>10</td>
<td>YC_1</td>
<td>QC_2</td>
<td>13:05</td>
<td>HAZ</td>
</tr>
</tbody>
</table>
The KPI for this test is not a quantitative but more a qualitative one. The input/output test can be considered as successful if the complete load plan is converted to the task chart according to the layout provided in subsection 5.1.8. To have indisputable data the percentage of successful converted rows is given as result.

Test 2 – Addition or adaption of input load plan
The current implementation is designed to accommodate multiple uploads of the load plan with different amounts of data. This means that when the load plan is uploaded for a second time with some data changed, added or removed, the unchanged data in the MySQL database should not be duplicated or overwritten and redundant data should be removed. To accommodate these requirements, each of the tables is first truncated before new data is written to the database. Testing this combination of functions can be done by changing the load plan as shown in table 7.1 and subsequently starting the task agent, input master and initializer m-files. The output in the database should then show that changed data rows are not duplicated and new rows are added to the existing data. Specifically, the human_interface_chart and task_chart in the MySQL database can be used to check this functionality.

The KPI for this test is also mainly qualitative as the test can be considered as successful if no data is duplicated or missing. But again a quantitative KPI is defined as the percentage of completely correct converted runs with ten different input sets.

Test 3 – No or wrong messaging
This test relates to the robustness of the system. The implementation of the input subsystem of the control architecture is done in such a way that the input master as well as the task agent displays messages if something in the process goes wrong. Besides that, the sub processes of both programs have been programmed in such a way that if confirmation is not received the process repeats its actions. For this test some predefined messages in the system are adapted to simulate wrong messaging. First of all, the variable System_Operation in the initialization file is set to no, which should result in displayed messages and termination of the processes. In addition to the initializer a misread of the load table is simulated to observe the message shown by the input master and task agent. Furthermore, a confirmation messages is changed to a empty one. Lastly, the req, sto, unr messages are changed to almost similar ones.

The KPI of this test defined as the amount of correct error displayed messages after testing with a wrong or missing process message.

Test 4 – Parallel database consultation
This test partly relates to the robustness of the concept and partly to the applicability for the overall control architecture. As in the total implementation multiple masters should be able to access the database it is important that the system does not collapse under multiple data store or pull request at the same time. Especially during real-time operation, it is of importance that for example the routing agent can update its trajectories while the AGV agents can also store their completed actions and receive new ones. Therefore, parallel MySQL MATLAB® data exchange is carried out to test this required functionality. Two and three different MATLAB® files are started at the same time with the same data request function. For every run the longest processing time is taken as measurement.

As KPI the time difference between a singular and parallel request is compared to a maximum allowable time difference. To provide the possibility for real time coordination this maximum allowable time difference is set to 200ms. The outcome of the test can then be compared to this maximum difference.

Test 5 – Process speed measure
This fifth test is of significant importance to determine whether the MATLAB® tools used in the implementation are suitable for the overall control architecture. The three different layers, as shown in figure 7.1 each require different operational frequency, increasing from towards the operational level. The TCP/IP and JDBC tool are both basic instruments that have been predefined by MathWorks®. The interfaces of these tools are constructed by the developers of MathWorks® and can therefore not easily
be altered. Using these tools thus bounds the functionality of the sub systems with a maximum operational frequency. So, to evaluate the used tools for this implementation measurements are carried out. First of all, the overall runtime of the three processes together is measured. Subsequently, the runtime of only the task agent and input master together is measured as the runtime of the initializer can be as a one-off action. Both these test are then repeated with a load plan that is 100 times larger than the initial one with ten rows.

For this test the KPIs are based on the three different levels of the control architecture. Furthermore, as the system is designed for a static as well as a dynamic operation a distinction between these methods is made too. Table 7.2 shows estimations of the minimal required operational repetition times and related frequencies of the different layers for the static and dynamic method. As exact operational times and frequencies can only be determined by real time testing or simulating future research could demonstrate slightly different outcomes. For now, the suitability of the implementation components can be evaluated based on a comparison with the estimated benchmarks.

Table 7.2: Required operational times and frequencies per layer and method

<table>
<thead>
<tr>
<th>System layer</th>
<th>Static [ms]</th>
<th>Dynamic [Hz]</th>
<th>Dynamic [ms]</th>
<th>Dynamic [Hz]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategic layer</td>
<td>∞</td>
<td>3000</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Tactical layer</td>
<td>3000</td>
<td>1/3</td>
<td>500</td>
<td>2</td>
</tr>
<tr>
<td>Operational layer</td>
<td>200</td>
<td>5</td>
<td>200</td>
<td>5</td>
</tr>
</tbody>
</table>

For the static method the strategic layer carries out the conversion of the load plan to a task chart only once until a new load plan if offered. Therefore, the required operational time is defined as infinite. In the tactical layer the scheduling, dispatching and routing is done based on the task chart provided by the strategic layer. For the static method all three of these functions are carried out before the AGVs start operating. But as updates about completed tasks are fed back to the tactical layer an operational time of 3000ms is appropriate based on the travelling times determined by Binneveld (Binneveld, 2017). For the operational time a minimum of 200ms is chosen as the AGVs need real time coordination based on the data from the OptiTrack™ system. With an average speed of of 0.3m/s and a safety distance of 0.2m, as defined before, this should be an appropriate operational time.

With the dynamic method applied in the future the strategic and tactical layer operational times change significantly. In the strategic level the task chart is now based on the actual performance with priority calls included and an AGV availability chart that is produced. Therefore, with the found travelling times in mind an operational time of 3000ms should satisfy this dynamic functioning of the input subsystem in the strategic layer. The tactical layer should now also update their processes at shorter operational times as the scheduling, routing and dispatching is now done based on the performance of the AGVs. As soon as tasks are completed the calculations should be updated. With an operational time of 500ms this dynamic operating should be possible. The operational time does not change with the dynamic method as the AGVs still need to be navigated on close to real time data. The only difference is that now more operation feed back is provided by the tactical layer. This means that more data is transferred, but with the same repetition times.

7.3.2 Results

In this section the results for each of the test are described and evaluated based on the KPIs defined in subsection 7.3.1. For the test where a qualitative KPI is defined the results are shown with screenshots of the produced outcome by MATLAB®.

Test 1 – Input/Output conversion

At first the load plan is read into the MATLAB® workspace resulting in the table shown in figure 7.12 and thereafter it is stored in the MySQL database as can be seen in figure 7.13.
Immediately after loading the excel file into a workspace table it is stored in the MySQL database as a load plan.

After the storage of the load plan as human_interface_chart in the MySQL database an initialization message is sent to the input master and task agent to start their functions. The first action is then to register the task agent in the database.

Figure 7.14 and figure 7.15 show the data that is stored in the register_publish and register_subscribe MySQL tables respectively.

**Figure 7.12:** human_interface_chart in MATLAB® workspace

<table>
<thead>
<tr>
<th>ContainerID</th>
<th>PickupLocation</th>
<th>DropOffLocation</th>
<th>EndTime</th>
<th>CargoDescription</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'YC_1'</td>
<td>'QC_1'</td>
<td>'13:00'</td>
<td>'HAZ'</td>
</tr>
<tr>
<td>2</td>
<td>'YC_2'</td>
<td>'QC_3'</td>
<td>'13:00'</td>
<td>'-'</td>
</tr>
<tr>
<td>3</td>
<td>'YC_3'</td>
<td>'QC_2'</td>
<td>'13:00'</td>
<td>'-'</td>
</tr>
<tr>
<td>4</td>
<td>'YC_4'</td>
<td>'QC_4'</td>
<td>'13:00'</td>
<td>'HAZ'</td>
</tr>
<tr>
<td>5</td>
<td>'YC_1'</td>
<td>'QC_5'</td>
<td>'13:00'</td>
<td>'-'</td>
</tr>
<tr>
<td>6</td>
<td>'YC_3'</td>
<td>'QC_2'</td>
<td>'13:05'</td>
<td>'-'</td>
</tr>
<tr>
<td>7</td>
<td>'YC_5'</td>
<td>'QC_1'</td>
<td>'13:05'</td>
<td>'HAZ'</td>
</tr>
<tr>
<td>8</td>
<td>'YC_3'</td>
<td>'QC_1'</td>
<td>0.54514'</td>
<td>' '</td>
</tr>
<tr>
<td>9</td>
<td>'YC_4'</td>
<td>'QC_3'</td>
<td>0.54514</td>
<td>'HAZ'</td>
</tr>
<tr>
<td>10</td>
<td>'YC_1'</td>
<td>'QC_2'</td>
<td>0.54514</td>
<td>'HAZ'</td>
</tr>
</tbody>
</table>

**Figure 7.13:** human_interface_chart in MySQL Workbench

After the storage of the load plan as human_interface_chart in the MySQL database an initialization message is send to the input master and task agent to start their functions. The first action is then to register the task agent in the database. Figure 7.14 and figure 7.15 show the data that is stored in the register_publish and register_subscribe MySQL tables respectively.

**Figure 7.14:** MySQL Workbench result for the Register_Publish table

<table>
<thead>
<tr>
<th>Register_Number</th>
<th>Caller_ID</th>
<th>Topic</th>
<th>Topic_Type</th>
<th>Frequency_Publish</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Strategic_01</td>
<td>Task_Chart</td>
<td>SQL</td>
<td>120</td>
</tr>
</tbody>
</table>

**Figure 7.15:** MySQL Workbench result for the Register_Subscribe table

<table>
<thead>
<tr>
<th>Register_Number</th>
<th>Caller_ID</th>
<th>Topic</th>
<th>Topic_Type</th>
<th>Frequency_Subscribe</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Strategic_01</td>
<td>Human_Interface_Chart</td>
<td>SQL</td>
<td>120</td>
</tr>
</tbody>
</table>
Then when the registering of the agent is carried out successful the human_interface_chart is requested by the task agent. Subsequently, the input master pulls the chart as shown in figure 7.13 and stores in its workspace as shown in figure 7.12. This table is send via TCP/IP to the task agent to use for its construction of the task_chart. This task_chart is stored in the workspace as shown in figure 7.16 before it is send to the input master to store in the MySQL database.

<table>
<thead>
<tr>
<th>Task_ID</th>
<th>Cont_ID</th>
<th>Cont_Orig</th>
<th>Cont_Dest</th>
<th>T_End</th>
<th>Cargo_Desc</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>240</td>
<td>'HAZ'</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>240</td>
<td>'-'</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>240</td>
<td>'-'</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>240</td>
<td>'HAZ'</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>240</td>
<td>'-'</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>245</td>
<td>'-'</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>5</td>
<td>1</td>
<td>245</td>
<td>'HAZ'</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>3</td>
<td>1</td>
<td>245</td>
<td>'-'</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>4</td>
<td>3</td>
<td>245</td>
<td>'HAZ'</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>NULL</td>
<td>NULL</td>
<td>245</td>
<td>'HAZ'</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 7.16: Task chart in the MATLAB® workspace

After successful transfer of this task chart to the input master it is stored as output in the MySQL database as shown in figure 7.17.

<table>
<thead>
<tr>
<th>Task_ID</th>
<th>Cont_ID</th>
<th>Cont_Orig</th>
<th>Cont_Dest</th>
<th>T_End</th>
<th>Cargo_Desc</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>240</td>
<td>HAZ</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>240</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>240</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>240</td>
<td>HAZ</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>240</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>245</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>5</td>
<td>1</td>
<td>245</td>
<td>HAZ</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>3</td>
<td>1</td>
<td>245</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>4</td>
<td>3</td>
<td>245</td>
<td>HAZ</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>NULL</td>
<td>NULL</td>
<td>245</td>
<td>HAZ</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 7.17: Task chart in MySQL Workbench

Lastly, the task agent is unregistered by removing the data from the register_publish and register_subscribe tables in the MySQL database.
The same operation is carried out ten times after each time clearing the data in the workspace and MySQL database. As result 100% of the runs was carried out successful. It can therefore be stated that the functionality of the input subsystem implementation meets the described concept and moreover that the exchange of data via TCP and JDBC with an external database is working.

**Test 2 – Addition or adaption of input load plan**

The results for this test are obtained in a similar way as for test 1. However, this time only the data in the MATLAB® files is cleared and not in the MySQL database. In this way it was possible to observe the functionality during changes of the input load plan. This time 90% of the runs was successful. Nine of the ten tests resulted in the expected and desired output. Nevertheless, one run completely failed with zero correct converted rows. Instead of the new rows the rows of the previous run were still stored in the MySQL database. Analysis showed that data conversion and storage failed due to a missing container ID in the load plan excel file. An error message showed that the empty input entry was not accepted, which caused that the task agent and input master did not receive the start message and thus did not carry out their functions. Instead of carrying out their functions the task agent and input master did show the message ‘Initialization has failed’ making the user aware of the code section of the processing failure. These messages are discussed and tested further in test 3. As the failure in this test was caused by an inaccurate input set and a specific warning came forward the test can be seen as successful. The addition or adaption functionality can thus be seen as adequate.

**Test 3 – No or wrong messaging**

For this test the used message variables are changed in the MATLAB® codes. After altering the messages to wrong strings the task agent and input master are started again. For each of the changed messages the desired output message is shown in table 7.3 with a yes or a no for the input master and the task agent.

<table>
<thead>
<tr>
<th>Message</th>
<th>Desired output message</th>
<th>Input master</th>
<th>Task agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>System operation</td>
<td>System not in operation</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Initializer failure</td>
<td>Initialization has failed</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Register confirmation</td>
<td>Registering of the agent has failed</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Req message</td>
<td>Data pull from the database has failed/ Data</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>is not received</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sto message</td>
<td>Data storage has failed</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Unr message</td>
<td>Unregistering of the agent has failed</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

From table 7.3 it can be deduced that wrong messaging within the system is recognized and displayed in the output window of MATLAB®. Remark is that with this preliminary implementation the messages are set with variables and changed manually for this test. As a result, the task agent and input master start to repeat the same part of the function indefinite as the message cannot be changed back during operation. Future research should define whether it is better to stop the system after wrong messaging or keep it looping till the right message is received. For now, according to this test the system can be considered as robust as it recognizes errors, shows those in the output and keeps repeating till the system can continue as it should.

**Test 4 – Parallel database consultation**

To carry out the parallel database consultation the ‘startat’ timer function is used in MATLAB® to start the scripts on different computers at similar times. The parallel data request consists of pulling a task chart with 25 rows from the database to store it in the workspace. Table 7.4 shows the longest processing time when one, two or three requests are done parallel. The processing time is measured using the ‘tic-toc’ function in the MATLAB® codes.

Table 7.3: Results of test 3
In table 7.4 it can be seen that the time difference between a single and two or three parallel requests is less than five thousandth of a second. With respect to the set allowable maximum difference of 200ms it is likely that the MySQL database interaction with MATLAB® is amply sufficient. Moreover, the small difference between two or three parallel requests suggest that a higher number of parallel requests will not result in an excessive increase in processing time. A small remark should be placed at the standard deviation that is increasing with more parallel requests. Yet, this increase is still relatively small and is therefore neglected.

Test 5 – Process speed measure
As result the longest processing time of the combination with the task agent and input with and without the initializer is provided. Different from previous tests the data is not cleared from the MATLAB® workspace as that is also not the case during repeated processing when the system is in operation. Again the files are simultaneously started using the ‘startat’ function, with the files in this function started in the following order; initializer, input master, task agent. The measured process times are shown in table 7.5. As specifically the operational times are compared to the KPI values, the frequencies are not shown in this table.

<table>
<thead>
<tr>
<th></th>
<th>1 Request [ms]</th>
<th>2 Requests [ms]</th>
<th>3 Request [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17.90</td>
<td>26.52</td>
<td>44.32</td>
</tr>
<tr>
<td>2</td>
<td>16.27</td>
<td>19.09</td>
<td>28.60</td>
</tr>
<tr>
<td>3</td>
<td>16.61</td>
<td>28.78</td>
<td>36.23</td>
</tr>
<tr>
<td>4</td>
<td>15.07</td>
<td>15.63</td>
<td>19.77</td>
</tr>
<tr>
<td>5</td>
<td>14.94</td>
<td>20.18</td>
<td>16.75</td>
</tr>
<tr>
<td>6</td>
<td>15.49</td>
<td>20.57</td>
<td>18.19</td>
</tr>
<tr>
<td>7</td>
<td>16.70</td>
<td>19.22</td>
<td>16.69</td>
</tr>
<tr>
<td>8</td>
<td>15.03</td>
<td>27.08</td>
<td>14.96</td>
</tr>
<tr>
<td>9</td>
<td>15.19</td>
<td>18.83</td>
<td>16.88</td>
</tr>
<tr>
<td>10</td>
<td>14.81</td>
<td>17.76</td>
<td>15.94</td>
</tr>
<tr>
<td>11</td>
<td>15.97</td>
<td>18.17</td>
<td>15.93</td>
</tr>
<tr>
<td>12</td>
<td>14.58</td>
<td>17.79</td>
<td>17.95</td>
</tr>
<tr>
<td>13</td>
<td>15.26</td>
<td>18.88</td>
<td>18.14</td>
</tr>
<tr>
<td>14</td>
<td>16.46</td>
<td>18.74</td>
<td>18.27</td>
</tr>
<tr>
<td>15</td>
<td>15.23</td>
<td>18.81</td>
<td>17.77</td>
</tr>
<tr>
<td>16</td>
<td>14.07</td>
<td>18.63</td>
<td>14.38</td>
</tr>
<tr>
<td>17</td>
<td>13.69</td>
<td>18.78</td>
<td>16.14</td>
</tr>
<tr>
<td>18</td>
<td>15.94</td>
<td>16.27</td>
<td>15.08</td>
</tr>
<tr>
<td>19</td>
<td>14.86</td>
<td>18.18</td>
<td>15.59</td>
</tr>
<tr>
<td>20</td>
<td>14.19</td>
<td>17.47</td>
<td>21.51</td>
</tr>
<tr>
<td>Mean</td>
<td>15.41</td>
<td>19.77</td>
<td>19.95</td>
</tr>
<tr>
<td>Difference</td>
<td>3.52</td>
<td>4.54</td>
<td></td>
</tr>
<tr>
<td>Standard deviation s</td>
<td>1.02</td>
<td>7.71</td>
<td></td>
</tr>
</tbody>
</table>

In table 7.4 it can be seen that the time difference between a single and two or three parallel requests is less than five thousandth of a second. With respect to the set allowable maximum difference of 200ms it is likely that the MySQL database interaction with MATLAB® is amply sufficient. Moreover, the small difference between two or three parallel requests suggest that a higher number of parallel requests will not result in an excessive increase in processing time. A small remark should be placed at the standard deviation that is increasing with more parallel requests. Yet, this increase is still relatively small and is therefore neglected.

Test 5 – Process speed measure
As result the longest processing time of the combination with the task agent and input with and without the initializer is provided. Different from previous tests the data is not cleared from the MATLAB® workspace as that is also not the case during repeated processing when the system is in operation. Again the files are simultaneously started using the ‘startat’ function, with the files in this function started in the following order; initializer, input master, task agent. The measured process times are shown in table 7.5. As specifically the operational times are compared to the KPI values, the frequencies are not shown in this table.
In Table 7.5 it can be seen that there is a significant time difference between the process time with and without initializer and that the processing time increases significantly with a larger load plan. Besides that, it is noted that the processing times are relatively high with respect to the defined KPI’s. As the initializer is only one-off, the processing times without initializer are of particular interest for the system process. Then, referring back to Table 7.2 it can be stated that the current implementation suits the strategic level using the static method, but that it is not applicable for the lower levels or dynamic methods. A significant reduction in processing time is necessary to use this system structure in all layers and with future dynamic methods. This reduction can only be made when the cause of this long processing time is known. Therefore, the run and time option of MATLAB® is used to evaluate which segments of the code take up most of the processing time. Using this option, the total time of functions in the code is assessed. Figure 7.18 shows a part of the output screen produced by this run and time option. The overall processing time shown here is slightly higher than in Table 7.6 as the run and time function can only be manually started and not with the ‘startat’ function.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Calls</th>
<th>Total Time</th>
<th>Self time</th>
<th>Total Time Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputmasteres</td>
<td>1</td>
<td>18.815 s</td>
<td>0.053 s</td>
<td></td>
</tr>
<tr>
<td>com.mathworks.toolbox.instrument.TCP/IP (Java)</td>
<td>36</td>
<td>12.606 s</td>
<td>12.606 s</td>
<td></td>
</tr>
<tr>
<td>interface.fopen</td>
<td>3</td>
<td>12.035 s</td>
<td>0.002 s</td>
<td></td>
</tr>
<tr>
<td>connection.insert</td>
<td>100</td>
<td>4.918 s</td>
<td>0.002 s</td>
<td></td>
</tr>
<tr>
<td>connection.fastinsert</td>
<td>100</td>
<td>4.917 s</td>
<td>0.180 s</td>
<td></td>
</tr>
<tr>
<td>com.mysql.jdbc.JDBC42PreparedStatement (Java)</td>
<td>1104</td>
<td>4.443 s</td>
<td>4.443 s</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7.18: Run and time output dataset
It is clearly shown in figure 7.18 that the TCP/IP calls and the data insert via the JDBC driver take up almost all of the processing time of the input master. It can thus be concluded that the basic MATLAB® toolbox functions are not suitable for real-time coordination. Therefore, it is of importance that these tools are replaced with external variants that allow significant higher data transfer speeds. Furthermore, it a secondary option is to replace MATLAB® with similar programs that are more suitable for real-time processing. Lastly, it is also possible to alter the network used by the computers to interact. In table 7.6 it can be seen that the standard deviation is rather high, indicating that the network is not completely stable. All of these recommendations need further research to determine how they can reduce the overall processing times and which suites the best. Advantage of the control architecture with communication through a central database is that it is possible to use different tools and setups for each subsystem. This means that these setups and tools can be specifically constructed based on the requirements for that particular subsystem.

7.4 Concluding remarks

This chapter described the first preliminary implementation of the control architecture as designed in Chapter 5 and Chapter 6. The implementation was based on the input subsystem in the strategic layer, which should provide a fundament for the future development of the lower layers. As the current system in the AGV laboratory is based on static methods only the input master and task agent were selected for the implementation. For the central database the MySQL Workbench platform was used in combination with a MySQL server. To simulate the distributed setting of the system the MySQL database, input master program and task agent program were divided over three different computers in the AGV laboratory. Communication between the MySQL database and the input master was established with the MATLAB® function ‘database’ in combination with a JDBC driver. For the communication between the MATLAB® files the standard TCP/IP toolbox from MATLAB® was used.

With the general setup in place a brief instruction manual was provided to allow for easier development in future research. The use of MySQL workbench was discussed with the fundamentals to create new storage tables that can be accessed from MATLAB®. Subsequently, the connection parameters for the database connection were presented along with the most elemental data processing codes. Thereafter, the TCP/IP MATLAB® toolbox function was elaborated on to show how communication between MATLAB® files on different computers could be established. Lastly, it was demonstrated how to activate the input master and task agent by an initializing program that also stores the delivered excel load plan in the MySQL database. With these four elemental system parts elucidated further implementation and extension of the system is less complicated.

The third section of this chapter then continued with a proof of concept comprising of tests and their results. To test the functionality as well as the feasibility of the control architecture and its preliminary implementation the following five test were established:

1. Input/output conversion test to examine whether the implementation functions according to the description provided in chapter 5 and chapter 6
2. Addition or adaption of the load plan test to examine whether the system implementation is able to cope with changes in the load plan and is thus flexible
3. No or wrong messaging test to examine the robustness of the system implementation
4. Parallel database test to examine whether the implemented MySQL database is able to quickly process parallel data requests
5. Process speed measurement test to examine the feasibility of the implementation with its used tools for each of the control architecture layers

The results of these tests were compared with predefined KPIs to be able to quantitatively assess the present performance. The first three tests showed that the system functioned according to the predefined control architecture and that is was flexible and to a certain extent robust. The fourth test showed that the MySQL database is able to execute parallel data operations without significant increase in operational time. This meant that the MySQL platform is suitable for the designed control architecture.
as multiple subsystems can interact with it at the same time. The fifth test showed process time results that to a remarkable extent did not satisfy the predefined KPI operational times. It demonstrated that the JDBC and TCP/IP interfaces constructed by MathWorks® seriously limit the performance of the system. As recommendation it was stated that these basic tools could be replaced with higher performance ones, a different program than MATLAB® could be used or that a more stable connection between the computers could be established. Lastly, it was stated that the flexibility of the proposed control architecture with segregated subsystem exchanging data with the central database makes it possible to adjust these recommendations for each subsystem specifically.
8. Conclusion & Recommendations

This chapter starts with a conclusion in section 8.1, which answers the sub questions and subsequently the main research question. At the end of section 8.1 an illustration is provided that comprises all the designed control architecture components. Then in section 8.2 recommendations are provided for future research that continues on this fundament.

8.1 Conclusion

With the main objective to design a control architecture for the AGV system in the laboratory of the Delft University of Technology a constructive development process was run through. The first step comprised of gaining insight in the industrial AGV system and the simulated version in the AGV laboratory. Subsequently, a literature survey was carried out to create an overview of the possibilities for control architectures. With all the possibilities defined concept choices were made regarding the system layout and communication and cooperation methods. Thereafter, these choices were formatted into a proposed global and detailed design of a suitable control architecture. Finally, with the system structure defined a preliminary implementation of the top layer of the system was done to establish a proof of concept based on the results of five tests. Each of these steps are now briefly elaborated on in the remainder of this chapter.

The AGV system was characterized as a system that comprised of the operations carried out by the QCs, YCs and AGVs. The AGV was described as a robot that navigate autonomously based on several navigation methods, in this case particularly applied for container transport. In the industrial application a distinction was made between necessary equipment and logistic functions. As equipment the AGVs, QCs and YCs, navigation equipment and control computer were cited. For the logistic functions the strategic, tactical and operational layer setup contained the job scheduling, dispatching, routing and navigation functions. In the AGV lab the industrial AGVs were simulated with a scaled simplified mechanism of a 1:25 scale. The current control architecture consisted of a tactical, operational and physical layer where the job scheduling, dispatching and routing function appeared again. However, these functions were considered to be in the tactical layer of the system while the observer and controller and the motive tracker and Arduino board were in the so-called operational and physical respectively. With the description of these components and the final statement that the quay transport operations are reasonably simulated, but that the simulation of the QC and YC operations and the system component coherence still lack the first sub question was answered. Specifically, resolving this lack of system component coherence became the focus of the remainder of the report.

To come up with possibilities for a control architecture design to create system component coherence in the AGV lab the literature survey was carried out. For the global system layout, the (de)centralized, distributed, coordinated and multilevel control architectures were found as options with MAS as side note by distributed control. Then, regarding subsystem collaboration coupled and decoupled and cooperating or non-cooperating methods were weighted. Lastly, the communication possibilities were discussed. A main division was made between the push and pull principles and direct and indirect communication. With these segments outlined sub question two was answered.

With future complexity and dynamic extension in mind a distributed, multilevel, coordinated system structure with the MAS concept was considered as the best option. Especially, the MAS concept with the potential to implement intelligent data processing programs provided much possibilities for advancements of the system. Furthermore, the subsystems should be coupled and collaborate in a
cooperative manner to enhance system efficiency. Lastly, the application of a hybrid push and pull mechanism seemed best suited for inter subsystem communication as it allowed different operational frequencies. A message board was the best option to accommodate this communication construction. For communication within subsystems direct interaction still fitted the best to expedite data transfers. These three main design directions answered sub question three.

To combine these choices, it was chosen to construct a system divided in three layers; the strategic, tactical and operational. Each of these layers consisted of subsystems that comprised of a data transfer arranging master and several agents under the supervision of that master. These agents were described as data processing programs with future potential to become intelligent. Data generated by the agents is stored in a central database that functions as message board via the subsystem master. Data required by the agents is also requested via the master, which then pulls that data from the database. In this way a structured and clear system setup is provided with each of the AGV lab components placed in an overall network. Figure 5.9 can be consulted for an illustration of the answer to sub question four.

The implementation of the control architecture had the objective to show the system operating in a distributed manner. Therefore, it was chosen to use three computers with on one a database, on the other the master and on the last one an agent. This meant that the database, chosen to be a MySQL database, had to be accessed from an external computer and that MATLAB® files on two segregated computers had to communicate via the network. Since it was a preliminary implementation the basic tools were used to set up these communications, with a JDBC driver for the MATLAB® to MySQL communication and TCP/IP for communication between the master and agent. Specific command codes and login data were created and showed in the implementation chapter. Figure 7.2 and figure 7.3 show the above described structure so that a fundament was laid to answer the last sub question.

It was decided that the implementation had to be tested on mainly functionality and feasibility. First of all, the distributed setting had to be proven functional and secondly whether the setup and used tools were feasibility for the overall design. This resulted in three functionality tests; a general input/output conversion test, an adaption or addition of load plan test and a robustness test with errors triggered on purpose. The other two tests comprised of testing parallel processing of database requests to show that multiple subsystems could consult the database at the same time without large processing delays and measuring the overall processing time to show whether real time operation is possible. Results of the test showed a flawless functionality but a poor feasibility for real time operation. The parallel database requests were processed with hardly significant time delays, but the overall processing time of the master and task agent combination was too long. It was concluded that the basic tools of MATLAB® could not achieve the desired and required operational speed. For the static method processing the current implementation processing time would be sufficient for the strategic layer of the system, but not for any lower one. Besides that, for the potential dynamic methods the processing time was insufficient for all three of the system layers. So, the implementation and the tests with their results answered the last sub question.

With the answers to the sub questions provided the main research question can be addressed: “What is a suitable architecture for a DC network that facilitates efficient functioning of the overall system and how could it be implemented in real container terminals?” Based on the answers of the sub questions a control architecture with multiple subsystems was defined. These subsystems are coupled via a central database and work together in a cooperative manner. The inter subsystem communication goes through the database based on a hybrid push and pull mechanism while direct communication is used between components within the subsystems. Figure 8.1 once again shows the layered structure with the present subsystems and their agents together the central database to illustrate the above described setup.
After experimenting with a basic implementation is can be concluded that a distributed network with a multilevel coordinated structure and implementation of the the MAS concept on purely functionality facilitates the AGV system. Yet, the experiments showed that the current implementation with basic tools did not fulfil the real time and dynamic coordination requirements. Therefore, further extensive research and development is necessary to implement the complete control architecture. As result it is not yet possible to decide how such a system, accommodating real time coordination, could be implemented in a real container terminal.

8.2 Recommendations

With the basic fundament of the new AGV lab control architecture provided subsequent research should continue to develop the system from a written and illustrative manner to a fully functioning real time coordination of the AGVs in the lab. Therefore, some recommendations are done in this section to enhance this process. The recommendations are based on the findings of this research and are thus considered as right directions at this point in time.

First of all, it is important to stick to the design setup constructed in this research. As the current AGV lab system components are self-contained programs it is hard to combine them. Each of the programs has been developed in an isolated manner, which makes interactions and combinations hard. To prevent this, it is recommended that the system implementation is carried out according to the structure and protocols of this research. As it is likely that during further development of the system improved solutions come forward, it is important to implement and use these changes in each of the follow-up researches.

The second recommendation concerns testing of the implemented system parts. The experiments done during this research showed potential inconsistencies in the program interactions. Different program starting orders showed slight differences in the processing times. As eventually the experiments were carried out with the ‘startat’ command line the programs all started at the exact same time. Still, this slight difference could be of influence during testing of other system components. Therefore, it is recommended to further investigate this inconsistency or rule it out by using commands like the ‘startat’.

Continuing on process time inconsistencies it is suspected that the stability of the internet connection of the Delft University of Technology is irregular. Parallel network speed tests showed that the process times became longer with slower internet connections. Therefore, a recommendation is to setup a more stable and more isolated network for the program interactions of the control architecture.

A last note is placed at the present equipment and toolboxes. The experiment results of subsection 7.3.2 clearly showed that the current basic tools did not accommodate processing times necessary for real time coordination. Therefore, it is recommended to use more advanced programs and non-basic tools. Especially the data transfer speed should be enhanced by using external data transfer tools. Which programs or which tools better suite the system layers should be examined with further research.
Bibliography


## Appendix

### Appendix A: Computer and program specifications

<table>
<thead>
<tr>
<th>Operating system</th>
<th><strong>TUD278415</strong></th>
<th><strong>TUD207330</strong></th>
<th><strong>TUD278416</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Windows 7 Enterprise</td>
<td>Windows 7 Enterprise</td>
<td>Windows 7 Enterprise</td>
</tr>
<tr>
<td>System type</td>
<td>Intel® Xeon® CPU E5-1620 v3 @ 3.50GHz</td>
<td>Intel® Xeon® CPU E5-1620 v3 @ 3.50GHz</td>
<td>Intel® Xeon® CPU E5-1620 v3 @ 3.50GHz</td>
</tr>
<tr>
<td>RAM Memory</td>
<td>64-bit</td>
<td>64-bit</td>
<td>64-bit</td>
</tr>
<tr>
<td>RAM Memory</td>
<td>8GB</td>
<td>8GB</td>
<td>8GB</td>
</tr>
<tr>
<td>Program versions</td>
<td>MATLAB® R2017B MySQL Workbench V6.3.10 MySQL Server V5.7 JDBC Driver V5.1.46</td>
<td>MATLAB® R2017B JDBC Driver V5.1.46</td>
<td>MATLAB® R2017B JDBC Driver V5.1.46</td>
</tr>
</tbody>
</table>
Appendix B: MATLAB® codes

B.1: Initializer M-file

```
% Process timer command
tic;

% Variables with regards to the TCP/IP Connection
ts = tcpip('0.0.0.0', 55000, 'NetworkRole', 'Server');
set (ts, 'OutputBufferSize', 200)
System_Operation = 'Yes';
Operation = num2str(unicode2native(System_Operation));

% Read the stored excel load plan
human_interface_chart = (readtable('Load_plan.xlsx'));
[m, n] = size(human_interface_chart);
if strcmpi(System_Operation, 'Yes')

% Make connection to database using JDBC driver.
dbname = 'agvdatabase';
username = 'AGVlab';
password = 'AGVLaboratory';
driver = 'com.mysql.jdbc.Driver';
conn = database(dbname, username, password, driver, dburl);

% Remove database data
Topicstorage = 'human_interface_chart';
sqlquery = ['TRUNCATE ' char(topicstorage)];
Remove = exec(conn, sqlquery);

% Insert each row of the human_interface_chart with a loop where the i
% depends on the maximum number of rows of the inserted load plan
for i = 1:m

% Determine data to be inserted in the database
LoadplanColnames = {'Cont_ID', 'Pickup_Loc', 'Dropoff_Loc', 'T_End', ...
'Cargo_Des';
LoadplanData = {table2array(human_interface_chart(i, 1)), char(table2array(human_interface_chart(i, 2)));
char(table2array(human_interface_chart(i, 3)));
char(table2array(human_interface_chart(i, 4)));
char(table2array(human_interface_chart(i, 5)))};
LoadplanTablename = 'human_interface_chart';
insert(conn, LoadplanTablename, LoadplanColnames, LoadplanData);
end

% Send the messages to the task agent and input master to start their
% operation.
fprintf(ts, Operation);
fclose(ts)
fprintf(ts, Operation);
fclose(ts)

% Send message not to start operation is System_Operation is no.
else
fprintf(ts, Operation);
fclose(ts)
fprintf(ts, Operation);
fclose(ts)
end
toc;
```

Figure B.1: Initializer m-file
B.2 Input master M-file

```matlab
%% Process timer command
tic;
%% Process only starts if message is received that the system is in operation
if start >= 1
    start = tcpip(131.180.29.225, 55000, 'NetworkRole', 'Client');
    fopen(start);
end
%% if no connection, display system not in operation
if strcmpi(start, 'No')
    disp('System not in operation')
    return
else
    Make connection to database using JDBC driver.
    dbname = 'agvdatabase';
    username = 'AGVLab';
    password = 'AGVLaboratory';
    driver = 'com.mysql.jdbc.Driver';
    conn = database(dbname, username, password, driver, dburl);

    % Set TCP connections
    tr = tcpip(131.180.30.35, 55000, 'NetworkRole', 'Client');
    ts = tcpip(0.0.0.0, 55000, 'NetworkRole', 'Server');
    set(str, 'InputBufferSize', 50000)
    fopen(tr);
    fopen(ts);

    % Register function for agents that publish and subscribe data
    % Register register data from Task Agent
    j = 1;
    while j<2
        Publish = tcpip(tr);
        Publish = native2unicode(str2num(Publish));
        dlmwrite('Publish_TaskAgent.txt', Publish, 'delimiter', ');
        Publish_TaskAgent = readable('Publish_TaskAgent.txt');
        Subscribe = tcpip(tr);
        Subscribe = native2unicode(str2num(Subscribe));
        dlmwrite('Subscribe_TaskAgent.txt', Subscribe, 'delimiter', ');
        Subscribe_TaskAgent = readable('Subscribe_TaskAgent.txt');
    end

    % Register the agent as publisher
    Register_ID = Publish_TaskAgent.Caller_ID;
    sqlquery = ['SELECT Register_Number FROM register_publish WHERE ...'
                ' Caller_ID = ', Register_ID, '']=';'
    Existence = exec(conn, sqlquery);
    Existence = fetch(Existence);
    Existence = cell2mat(Existence.Data);
    if Existence is numeric(Existence)
        else

        % Select the maximum value that is already used in the registering chart
        sqlquery = ['SELECT MAX(Register_Number) FROM register_publish'];
        MaxValue = fetch(MaxValue);
        MaxValue = cell2mat(MaxValue.Data);
        % Create a new Register_Number that is 1 higher than the max value in the
        % chart. If there is no one registered yet Register Number 1 is assigned
        if isnumeric(MaxValue)
            Register_Number = 1;
        else
            Register_Number = MaxValue+1;
        end
    end
    % Register the name of the variables required for the dataInsert command
    RegisterPublishColnames = {'Register_Number', 'Caller_ID', 'Topic', 'Topic_Type', '
                                RegisterPublishedData = {'Register_Number', char(Publish_TaskAgent.Caller_ID),
                                char(Publish_TaskAgent.Topic),
                                char(Publish_TaskAgent.Topic_Type),
                                char(Publish_TaskAgent.Frequency),
                               _sql_insert(conn, RegisterPublishTablename, RegisterPublishColnames, RegisterPublishData)
```
% Register the agent as subscriber
% Check whether the agent is already registered
caller_id = subscribe_taskagent.caller_id;
squery = ['SELECT Register_Number FROM register_subscribe WHERE '...
      'caller_id = ', caller_id, '];
existence = exec(conn, squery);
existence = fetch(existence);
existence = cell2mat(existence.data);

% If existence is numeric the agent is registered and thus no action should
% be taken else the register process can be executed
if isnumeric(existence)
    else
% Select the maximum value that is already used in the registering chart
squery = ['SELECT MAX(Register_Number) FROM register_subscribe;
maxvalue = exec(conn, squery);
maxvalue = cell2mat(maxvalue.data);

% Create a new Register_Number that is 1 higher than the max value in the
% chart. If there is no one registered yet Register_Number 1 is assigned
if isnan(maxvalue)
    register_number = 1;
else
    register_number = maxvalue + 1;
end

% Set up the variables required for the datainsert command
registersubscribecolnames = ['Register_Number', 'Caller_ID', 'Topic', 'Topic_Type' ...  ...
'Frequency_Subscriber'];
registersubscribe = (register_number, char(subscribe_taskagent.caller_id),
char(subscribe_taskagent.topic), ...
char(subscribe_taskagent.topic_type), ...
(subscribe_taskagent.frequency_subscriber));
datainsert(conn, registersubscribe tablename, registersubscribecolnames, registersubscribe dataplot)
end

% Send confirmation message to the task agent and proceed if done
confirmation = 'Success';
confirmationcheck = cls2str(unicode2native(confirmationcheck));
fprintf(ts, Confirmation);
if strcmpi(confirmationcheck, 'Success')
    j = j+1;
else
    disp('Registering of the agents has failed')
    j = 3;
end

% Request data function
% Now the input master checks for the data request
j = 1;
while j=2
message = fscanf(tr);
message = native2unicode(str2num(message));
if strcmpi(message, 'Req')
% Select the name of the agent to find the corresponding topic in the
% registration table
caller_id = subscribe_taskagent.caller_id;
squery = ['SELECT Topic FROM register_subscribe WHERE '...
      'caller_id = ', caller_id, '];
% Create an output variable with the required table name
topicselect = exec(conn, squery);
topicselect = fetch(topicselect);
topicselect = cell2mat(topicselect.data);
% Select all the data of the chart required by the agent, which is
% specified. squery = ['SELECT * FROM ', (topicselect)];
% Convert the object data into a table
table = exec(conn, squery);
table = fetch(table);
table = cell2table(table.data);
% Create a cell array with columnnames depending on the amount of columns
% of the requested database table and then convert to a cell that can be
% used in the property function to name the columns of the workspace table
[out, in] = size(table);
for i = 1:n
    names{i,1} = char(columns{i,1});
end
end
% Name the columns of the created chart to the columns in the database
Chart.Properties.VariableNames = Names;

% Rename the output table to the name requested by the agent
eval(sprintf('TopicSelect' 'Chart'));

% Send the pulled chart to the Task Agent
writeTable('human_interface_chart', 'human_interface_chart.txt')
human_interface_chart = fwrite('human_interface_chart.txt',
human_interface_chart = num2str(unicode2native(human_interface_chart));
fprintf(ts, human_interface_chart);

% Receive confirmation message from task agent
ConfirmationRec = fscanf(tr);
ConfirmationRec = native2unicode(str2num(ConfirmationRec));

% If a confirmation of receiving of the task agent is received while
% loop can be left
if strcmpi(ConfirmationRec, 'Success')
j = j+1;
else
    disp('Data pull from the database has failed')
    j = 3;
end
else
    disp('Data pull from the database has failed')
    j = 3;
end

% Data store function
% If the message Sto is send the input master starts its data storing
% function
j = 3;
while j<2
    Message = fscanf(tr);
    Message = native2unicode(str2num(Message));
    if strcmpi(Message, 'Sto')
        task_chart = fscanf(tr);
        task_chart = native2unicode(str2num(task_chart));
        dlmwrite('task_chart.txt', task_chart, 'delimiter', ' ')
        task_chart = readtable('task_chart.txt');
        % Select the name of the agent to find the corresponding topic in the
        % registration table
        Topic = Publish_TaskAgent.Topic;
        sqlquery = ['SELECT topic FROM register_publish WHERE '...
                    Topic = ', char(Topic), ''];
        % Create an output variable with the table name of the data that needs to be
        % stored
        Topicstorage = exec(conn, sqlquery);
        Topicstorage = fetch(Topicstorage);
        Topicstorage = cell2mat(Topicstorage.Data);
        % Create a dataset with the data of the chart, defined by the string Topicstorage
        % that needs to be stored
eval(['Dataset = ', Topicstorage, ';']);
        [n,m] = size(Dataset);
        Topicstorage = 'task_chart';
        sqlquery = ['TRUNCATE ' char(Topicstorage); ];
        remove = exec(conn, sqlquery);
        % Insert or update each row of the published chart with a loop where the i
        % depends on the maximum number of rows of the inserted load plan
        for i = 1:m
            % Determine the variables for the update or insert command
            TableColumnNames = Dataset.Properties.VariableNames;
            TableName = Topicstorage;
            TableData = Dataset(i,:);
            insert(conn, TableName, TableColumnNames, TableData)
        end
        % Send confirmation message to the task agent
        ConfirmationStor = 'Success';
        ConfirmationStor = num2str(unicode2native(ConfirmationStor));
        fprintf(ts, ConfirmationStor);
        j = j+1;
    % If there is no data store message the function does stays in the while
    % loop
    else
        disp('Data storage has failed')
        j = 3;
    end
end
\texttt{\% Unregistering function}
\texttt{j = 1;}
\texttt{\% Select the Caller_ID from the table provided by the agent to assign the}
\texttt{\% input row to be removed}
\texttt{Caller_ID = Publish\_TaskAgent.Caller\_ID ;}
\texttt{\% Define the query that has to be executed in SQL and use the executive}
\texttt{\% command}
\texttt{sqlquery = \"DELETE FROM register\_publish WHERE \"...}
\texttt{\"\"\"Call_ID = \',\',\'\' char(Caller\_ID),\',\',\'\\"\"\\\"\"\";
\texttt{Remove = exec(conn, sqlquery);}
\texttt{\% Select the Caller_ID from the table provided by the agent to assign the}
\texttt{\% input row to be removed}
\texttt{Caller_ID = Subscribe\_TaskAgent.Caller\_ID ;}
\texttt{\% Define the query that has to be executed in SQL and use the executive}
\texttt{\% command}
\texttt{sqlquery = \"DELETE FROM register\_subscribe WHERE \"...}
\texttt{\"\"\"Call_ID = \',\',\'\' char(Caller\_ID),\',\',\'\\"\"\";
\texttt{Remove = exec(conn, sqlquery);}
\texttt{ConfirmationUnr = \"Success\" ;}
\texttt{ConfirmationUnr = num2str(unicode2native(ConfirmationUnr));}
\texttt{fprintf(ts, \{ConfirmationUnr\};}
\texttt{j = j+1;}
\texttt{\% In case of failure display message and send fail message to the task}
\texttt{\% agent}
\texttt{disp(\texttt{\textquoteleft}Unregistering of the agent has failed\textquoteright\texttt{)} ;}
\texttt{ConfirmationUnr = \textquoteleft}Fail\textquoteright\texttt{ ;}
\texttt{ConfirmationUnr = num2str(unicode2native(ConfirmationUnr));}
\texttt{fprintf(ts, \{ConfirmationUnr\};}
\texttt{j = j+1;}
\texttt{end}
\texttt{end}
\texttt{\% Close connections}
\texttt{fclose(ts);}
\texttt{fclose(tr);}
\texttt{fclose(tstart);}
\texttt{else}
\texttt{disp(\texttt{\textquoteleft}Initialization has failed\textquoteright\texttt{)} ;}
\texttt{end}
\texttt{toc;}

\textbf{Figure B.2: Input master M-File}
B.3 Task agent M-file

```matlab
% Process timer command
tic;

% Process only starts if a message is received that the system is in operation
st = tcpip('131.188.259.255', 55000, 'NetworkRole', 'Client');
set(st, 'InputBufferSize', 200)

fopen(st);
Start = fscanf(st);
Start = native2unicode(st2num(Start));

if strcmpi(Start, 'No')
disp('System not in operation')
return
else strcmpi(Start, 'Yes')

% This is a preliminary simulation of the Task_Agent of the control
% architecture designed for the AGV Laboratory at the Delft University of Technology
% Registering data for the task agent
Publish_TaskAgent = table({{'Strategic_01'},{'task_chart'},... {'SQL'},[128]});
Publish_TaskAgent.Properties.VariableNames = {'caller_ID',...
'Topic', 'Topic_type', 'Frequency_Publish'};

Subscribe_TaskAgent = table({{'Strategic_01'},...
{'human_interface_chart'},{'SQL'},[128]});
Subscribe_TaskAgent.Properties.VariableNames = {'caller_ID',...
'Topic', 'Topic_type', 'Frequency_Subscribe'};

% TCP connections initialization
tr = tcpip('131.188.156.214', 55000, 'NetworkRole', 'Client');
tt = tcpip('0.0.0.0', 55000, 'NetworkRole', 'Server');
set(tr, 'OutputBufferSize', 50000)
set(tt, 'InputBufferSize', 50000)
fopen(tt);

% The task agent only carries out actions if the system is in operation
if strcmpi(Start, 'Yes')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Registering of the agent
% If the system is in operation the task agent should register itself as a
% publisher and subscriber and only proceed if it receives confirmation
% that that registering has completed
j = 1;
while j<2

% Send out register message for publishing and subscription
write(table(Subscribe_TaskAgent, 'Subscribe_TaskAgent.txt'))
Publish_TaskAgent = fileread(Publish_TaskAgent.txt);
Subscribe_TaskAgent = fileread(Subscribe_TaskAgent.txt);
Publish_TaskAgent = num2str(native2unicode(Publish_TaskAgent));
Subscribe_TaskAgent = num2str(native2unicode(Subscribe_TaskAgent));
fprintf(tt, Publish_TaskAgent);
fprintf(tt, Subscribe_TaskAgent);

% Receive confirmation message
ConfirmationReg = fscanf(tr);
ConfirmationReg = native2unicode(st2num(ConfirmationReg));

% Only if confirmation is provided the system can get out of the while loop
if strcmp(ConfirmationReg, 'Success')
j = j+1;
else
disp('Registering of the agents has failed')
j = 3;
return
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Data request and receive function
j = 1;
while j<2

% Send Request for data
Message = 'Req';
Message = num2str(unicode2native(Message));
fprintf(tt, Message);
if strcmp(Message, 'Req')

% Receive the data from the input master
human_interface_chart = fscanf(tr);
human_interface_chart = native2unicode(st2num(human_interface_chart));
dlmwrite('human_interface_chart.txt', human_interface_chart, 'delimiter', '');
face_chart = readtable('human_interface_chart(1,1)');
Check = table2array(human_interface_chart(1,1));
```

2018_TEL_8237
% Send confirmation if table is received
ConfirmationRec = 'Success';
ConfirmationRec = num2str(unicode2native(ConfirmationRec));
fprintf(1, ConfirmationRec);
end
else
    disp('Data is not received')
end

% Main function of the Task Agent
% As the data is now requested and the confirmation is received that the
% data is pulled from the database the main function of the task agent can
% be carried out. The Human_Interface_Chart is used to create the
% Task_Chart. Later on the Create_Chart can be implemented too.

% [n,m] = size(human_interface_chart);
Task_ID = transpose([1:m]);
Cont_ID = human_interface_chart.Cont_ID;
Cont_Orig = human_interface_chart.Pickup_Loc;
Cont_Dest = human_interface_chart.Dropoff_Loc;
T_End = human_interface_chart.T_End;
Cargo_Desc = human_interface_chart.Cargo_Desc;

% For each row of the Human_Interface_Chart the input is renamed to a
% variable that is used in the system. Depending on the input of the load
% plan the 'OC_X' strings should be adapted. For now 5 QCs are assumed to
% be present.
for i = 1:m
    if strncmpi(Cont_Orig(i,1), 'QC_1')
        Cont_Orig(i,1) = num2cell(1);
    elseif strncmpi(Cont_Orig(i,1), 'QC_2')
        Cont_Orig(i,1) = num2cell(2);
    elseif strncmpi(Cont_Orig(i,1), 'QC_3')
        Cont_Orig(i,1) = num2cell(3);
    elseif strncmpi(Cont_Orig(i,1), 'QC_4')
        Cont_Orig(i,1) = num2cell(4);
    elseif strncmpi(Cont_Orig(i,1), 'QC_5')
        Cont_Orig(i,1) = num2cell(5);
    end
end

% The same done for the YCs, with again the assumption of 5 present YCs
for i = 1:m
    if strncmpi(Cont_Dest(i,1), 'OC_1')
        Cont_Dest(i,1) = num2cell(1);
    elseif strncmpi(Cont_Dest(i,1), 'OC_2')
        Cont_Dest(i,1) = num2cell(2);
    elseif strncmpi(Cont_Dest(i,1), 'OC_3')
        Cont_Dest(i,1) = num2cell(3);
    elseif strncmpi(Cont_Dest(i,1), 'OC_4')
        Cont_Dest(i,1) = num2cell(4);
    elseif strncmpi(Cont_Dest(i,1), 'OC_5')
        Cont_Dest(i,1) = num2cell(5);
    end
end

% Convert the time to the time in minutes that is left for completion of
% the task
for i = 1:m
    EndTime = T_End(i,1);
    [Hours, Minutes] = strtok(EndTime;');
    Hours = str2double(Hours);
    Minutes = str2double(Minutes;');
    Finishtime = Hours * 60 + Minutes;
    clock
end

% For now it is simulated that the operation starts at 00:00 else use
% c(1,4) as hours and c(1,5) as minutes
Currenttime = 9600+00;
OperationTime = Finishtime - Currenttime;
T_End(i,1) = num2cell(OperationTime);
end

% For the static case that is now considered the status is not of
% importance. The three strings complete, pending or executed could easily
% be implemented using data from lower levels
clear Status
for i = 1:m
    Status(i,1) = 0;
end

% Create the output chart of the task agent with the variables determined
% above
Task_chart = table(Task_ID, Cont_ID, cell2mat([Cont_Orig]),...
                cell2mat([Cont_Dest]), cell2mat([T_End]), [Cargo_Desc], [Status]);
Task_chart.Properties.VariableNames = {'Task_ID', 'Cont_ID', 'Cont_Orig',...
                                        'Cont_Dest', 'T_End', 'Cargo_Desc', 'Status'};
Figure B.3: Task agent M-file