University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Task-level Collaboration between Automated Guided Vehicles (AGV) for Efficient Warehouse Logistics

Tobias Frauenfelder
Zurich, Switzerland
Student ID: 18-914-986

Supervisor: Prof. Dr. Bruno Rodrigues, Thomas Grübl,
Prof. Dr. Burkhard Stiller
Date of Submission: September 24, 2024

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmuehlestrasse 14, CH-8050 Zurich, Switzerland
URL: http://www.csg.uzh.ch/

Declaration of Independence

I hereby declare that I have composed this work independently and without the use of any aids other than those declared (including generative AI such as ChatGPT). I am aware that I take full responsibility for the scientific character of the submitted text myself, even if AI aids were used and declared (after written confirmation by the supervising professor). All passages taken verbatim or in sense from published or unpublished writings are identified as such. The work has not yet been submitted in the same or similar form or in excerpts as part of another examination.

Zürich, 24. September 2024

Signature of student

Abstract

Fahrerlose Transportfahrzeuge (FTFs) sind eine Schlüsseltechnologie der aktuellen Lagerhausautomatisierung. Diese Technologie führt zu einer markanten Effizienzsteigerung und zur Kostenreduzierung von Prozessen. In den vergangenen Jahren gab es viel Weiterentwicklung in Robotersystemen, welche von der Industrie 4.0 genutzt wurden, jedoch wurde die Aufgabenteilung und Kollaboration von FTFs zum Erreichen eines gemeinsamen Ziels noch wenig erforscht.

Die Kollaboration von FTFs zur Konfliktlösung und zu Koordinationszwecken existiert bereits heutzutage und ist auch weitverbreitet. Jedoch befassen sich nur wenige Studien damit, ob es möglich ist, dass FTFs aufgabenbezogen miteinander kollaborieren und Aufgaben durch das Teilen einer Aufgabe in kleinere Teilaufgaben effizienter bewältigen können.

Diese Arbeit verfolgt das Ziel, die aufgabenbezogene Kollaboration zwischen Robotern zu erforschen, indem ein Prototyp implementiert und in einem fiktiven Lagerhausszenario evaluiert wird. Das Ziel ist es, herauszufinden, ob die kollaborative Aufgabenteilung und Ausführung zu einer erhöhten Effizienz und zu einer Reduktion der Lieferungsdauer in Lagerhäusern führen kann.

Dazu wird zuerst die verfügbare Literatur zu den Grundlagen sowie verwandte Forschungsprojekte begutachtet. Basierend auf den gewonnenen Erkenntnissen wird eine Lösung designt und ein Prototyp mit dem DJI Robomaster EP Core Roboter implementiert. Der Prototyp wird anschliessend experimentell getestet und evaluiert und verschiedene weitere Szenarien werden berechnet.

Die Resultate zeigen, dass die aufgabenbezogene Zusammenarbeit zwischen zwei FTFs theoretisch zwar Potenzial hat, jedoch mit dem implementierten Prototyp keine Effizienzsteigerung erzielt werden kann. Aufgrund der hohen Dauer für Frachtübergaben lohnte es sich für die FTFs nicht, Aufgaben aufzuteilen. Weitere Berechnungen zeigen jedoch, dass sich durch die Reduktion der Dauer von Frachtübergaben in bestimmten Situationen eine Kollaboration lohnen könnte.

Automated guided vehicles (AGVs) are a key technology in current warehouse automation. This technology leads to significant increases in efficiency and cost reductions in processes. In recent years, there has been much further development in robotic systems used by Industry 4.0, but little research has been done into task-level collaboration of AGVs to achieve a common goal.

Collaboration of AGVs for conflict resolution and coordination purposes already exists today and is widespread. However, few studies have investigated whether it is possible for AGVs to collaborate with each other on a task-level and to accomplish tasks more efficiently by dividing a task into smaller subtasks.

This thesis aims to explore task-level collaboration between robots by implementing a prototype and evaluating it in a fictitious warehouse scenario. The aim is to find out whether collaborative task sharing and execution can lead to increased efficiency and reduced delivery time in warehouses.

To this end, the available literature on the basic principles and related research projects will first be reviewed. Based on the knowledge gained, a solution is designed and a prototype with the DJI Robomaster EP Core robot is implemented. The prototype is then tested and evaluated experimentally, and various further scenarios are calculated.

The results show that although task-level collaboration between two AGVs has potential in theory, the implemented prototypes cannot increase efficiency. Due to the long duration of freight handovers, it is not cost-effective for the AGVs to split tasks. However, calculations show that reducing the duration of freight handovers could make collaboration worthwhile in certain situations.

Acknowledgments

Foremost, I would like to express my gratitude to my supervisor, Prof. Dr. Bruno Rodrigues for his support in the preparation and implementation of the Master's thesis and for the effort he has made to always provide me with the suiting hardware in a short time. His expertise, advice, and feedback led me in the right direction to realize this thesis.

I would also like to thank Prof. Dr. Burkhard Stiller for giving me the opportunity to write my Master's thesis at the Communication Systems Group.

Contents

Chapter 1

Introduction and Motivation

Efficiency is a cornerstone within the logistics world, where striking a balance between cost reduction and increased productivity has been the target of research for several years [30]. Research spans across different knowledge areas, ranging from the planning of physical spaces [36] and optimization of business processes [30] to the continuous digitization and automation of warehouse operations [2, 6]. Aligning physical space planning with business strategies is a fundamental step for efficient warehouses, but recently, the modernization and automation of internal operations have gained a lot of prominence. In particular, the digitization topic has been the major driver of innovations, being referred to as Industry 4.0, a term that refers to the integration of a myriad of sensors, actuators, and robots, combined with analytics and cognitive algorithms to deliver a more autonomous, flexible, and responsive logistics ecosystem [7].

Among several advances, the use of robots is one of the most prominent transformative forces in production and warehouse logistics [30]. Robots facilitate the automation of manual and repetitive tasks, such as receiving parcels from production, precise storage and inventory within the warehouse, and automated pick-up/delivery to distributors [23]. Literature highlights two main types of robots typically used in warehouses [3, 20, 35]: Automated Guided Vehicles (AGV) and Autonomous Mobile Robots (AMR). While AMRs operate freely in unstructured environments using real-time decision-making algorithms, AGVs use simpler algorithms and navigate through predetermined paths in warehouses and factory floors. Another important difference lies in the level of autonomy of AMRs, which are typically equipped with more sensors and ad-hoc communication capabilities to perceive and adjust to real-world obstacles. AGVs, however, due to their design geared toward an operation on predefined paths, present a much higher efficiency in several tasks such as storing and delivering parcels. A step beyond simply using robots to streamline warehouse processes is the collaboration between robots [35]. While collaboration between AMRs is more common due to their design [3, 35], AGVs are typically not collaborative when it comes to accomplishing tasks. In this sense, the literature recognizes cooperation between AGVs in the area of route planning since [3, 19], by following predefined routes, efficient planning prevents one AGV from blocking another. However, introducing cooperation between AGVs, especially those operating on rails within a warehouse, presents an opportunity to further enhance productivity at the risk that failures could block an entire warehouse flow determined by an AGV path.

1.1 Thesis Goals

- **Literature Overview**: Perform a theoretical analysis of suitable algorithms included in the proposal, encompassing the collaboration between robots beyond the AGVs' scope, including the literature on task-level collaboration aiming to maximize performance. This goal must output an accurate mapping of literature on the topic, as well as a suitable collaborative algorithm applicable to organizing tasks between two or more robots.

- **Design and Implementation**: Involves the proposal of an approach to self-organize warehouse tasks (e.g., storing parcels) among two or more robots. This goal requires the proposal of an algorithm focusing on the performance optimization of the tasks' execution in contrast to the same task executed by a single robot. Such an algorithm may also consider elements and scenarios where single-task execution performs better than a collaborative execution.

- **Tool Integration**: Involves prototyping the algorithm into the robots' logic using the existing SDK. The thesis will rely on the commercially available DJI RoboMaster EP Core [10], as a popular educational robot, where the algorithm may be implemented and tested. Thus, a task may be determined by a central system, and two or more robots operating on the same predetermined path, would need to coordinate on a task-level to complete the task.

- **Prototype Evaluation**: Once the system is designed and implemented, the goal is to benchmark the collaborative algorithm against the single robot execution to measure the time to elapsed to conclude the task. The evaluation must test the hypothesis that adding two or more AGVs operating over the same path in a collaborative manner can effectively improve warehouse efficiency. In addition, such analysis should discuss in which conditions and tasks a single-robot execution performs better than a collaborative execution.

1.2 Methodology

The following methodology is applied to achieve the above stated goals:

Literature Review: In the first part of this thesis, the important background concepts are covered. An initial understanding of Industry 4.0, robotics used in warehouse automation, as well as collaboration between multiple robots is gained. Furthermore, related work is examined, which provides insights on current research on multi-robot collaboration.

Prototypical Evaluation: Following the completion of the literature review, the practical component of the thesis is initiated and consists of three parts:

- **Design:** In this part, based on the information gained in the literature, the concept of the prototype is designed. This encompasses the scenario in which the prototype should operate, as well as the different problems it should be capable of solving.

- **Implementation:** This part focuses on bringing the design into reality. Hardware as well as a suiting technology stack is selected, upon which a prototype is implemented.
- **Evaluation:** After the implementation process, the prototype is evaluated. Real-world metrics are measured, and it is discussed if the research goal is met.

1.3 Thesis Outline

This report is structured into different chapters:

Chapter 2 covers all the fundamental concepts of this thesis. An overview of the Topic Industry 4.0 is provided, as well as an overlook of different robotic systems used in warehouse automation. A look at theoretical concepts of how robots collaborate and communicate is given. Furthermore, various related works are presented, discussed and put in relation to this master's thesis. In **Chapter 3** a prototype is designed based on the knowledge gained in the fundamentals and the related work. The scenario for which the prototype is designed is specified and the individual actors and the environment are explained. Furthermore, a focus is placed on the tasks that the prototype should perform and how the communication process should work. **Chapter 4** then focuses on concretizing the design by implementing a prototype. The focus of this chapter is on the hardware required for implementation, as well as on the software architecture and the individual modules. In order to test the software together with the hardware, the software is evaluated in **Chapter 5** by measuring and evaluating key metrics. This chapter is also dedicated to the discussion of the results and the limitations of the work. Finally, **Chapter 6** summarizes the work, draws conclusions, and gives an outlook on future work.

Chapter 2

Fundamentals

2.1 Background

2.1.1 Industry 4.0

The industry in human history has gone through significant changes in the past. The changes are often divided into industrial revolutions, which are defined by different technological advancements. The First Industrial Revolution took place in the late 18th century and early 19th century and was marked by the transition from hand production to methods making use of machines, which were steam powered. In the late 19th century to early 20th century, the discovery of electricity led to the second industrial revolution and significantly impacted the manufacturing processes. The third industrial revolution was driven by programmable controllers and computers, which took place mid-20th century to late 20th century and led to automation of some production processes [15]. We are currently in the midst of the fourth industrial revolution, often referred to as Industry 4.0. However, researchers frequently disagree on the meaning of this term.

According to Hofmann and Rüsch [17] Industry 4.0 is often used as a buzzword and lacks of a clear definition. They mention however that Industry 4.0 is characterized by a paradigm shift towards a more decentralized and self-regulating model of value creation. According to them, some important concepts are the Just-in-Time (JIT)/Just-in-Sequence (JIS) production concepts as well as technologies as Cyber-Physical Systems (CPS), Internet of Things (IoT), Smart Factories, cloud computing or additive manufacturing and Smart Factories. Furthermore, Weyer et al. [34] mentions that Industry 4.0 is powered by modern advancements in Information and Communication Technology (ICT) and is enabled by CPS and IoT, which enable factories to become smart factories.

To understand Industry 4.0 it is important to gain an understanding of the different technologies which enable it. Hence, we provide a detailed examination of CPS, IoT, and Smart Factories:

- **Cyber-Physical Systems (CPS)**: CPS are systems that combine the physical and computational world together and furthermore enable the interaction with humans

[1]. More practically, this means that embedded systems and networks monitor physical processes, for example in a warehouse or factory. After the term CPS was coined, it still lacked of a clear guideline how such systems can be integrated into the industry. Therefore, Lee et al. [24] came up with the *CPS 5C level architecture*, which is a step-by-step guideline consisting of five levels to integrate CPS into the manufacturing process: The smart connection level (1) involves the data collection of the sensors. The Data-to-Information Conversion Level (2) processes the gained raw data from the sensors and processes it into meaningful information that can be used to make decisions. The Cyber Level (3) is responsible for creating a virtual layer of the physical system. The Cognition Level (4) provides knowledge and analytics to expert users to make decisions. Lastly, the Configuration Level (5) takes the feedback from the cyberspace to the physical space and reconfigures the machines according to the information gained through the level 4.

- **Internet of Things (IoT)**: The term Internet of Things refers to objects, sensors, or other items for everyday use which are equipped with computing capabilities and have a connection to the internet. This added technology gives the devices the capability to collect, send and receive data to or from other sources without requiring any or little action from a human being [32]. In a warehouse or manufacturing setting, IoTs offer multiple applications: On the one hand, IoTs can be used in a factory to integrate data into an Enterprise Resource Planning (ERP) system. This helps with information gathering and can contribute to strategic decisions about the manufacturing process. Furthermore, sensors can be attached to production or warehousing equipment that checks the condition of the device and can thus independently report when they need maintenance to prevent downtime in the factory. Another area of application, especially in a department store, is real-time tracking of inventory and orders. This can help to optimize inventory levels [29].

- **Smart Factory**: The Smart Factory is a term that is synonymously used with the terms Ubiquitous Factory, Factory of Things and Real-time Factory. What people expect when they mention smart factories is that they are flexible, reconfigurable, operating at low cost and are agile. However, there does not exist a clear definition of what a Smart Factory exactly is. Nonetheless, what is clear is that all these characteristics should be achieved through automation, which is enabled by dedicated hardware and software. Another important enabler of smart factories which is often mentioned is the IoT [18].

2.1.2 Robotics in Warehouse Automation

To get an overview of how robots can be used in warehouse automation, it is important to have an overview of which robot technology is available nowadays. In a warehouse setting, different kinds of robots specialize in different tasks are used. A selection of robots, which, however, is not exhaustive, is listed below:

- **Automated Storage and Retrieval System (ASR)**: This system is an automated framework whose main purpose is to store and retrieve items in a warehouse or

production facility. The system is especially efficient, if a warehouse has a high item movement rate. Likewise, it is suitable if a facility has a high storage and efficiency rate. The ASR improves the organization within a warehouse can reduce labor cost and decrease the demand for workforce [31].

- **Goods-To-Person (G2P) Technology**: This technology is well suited for e-commerce retailers. Back in the days, if a delivery store received an order, a person had to pick up every product and then pack it together. This is called the Person-To-Goods model, which is very labor-intensive and a waste of time. The G2P technology takes the product from the place where it is stored and delivers it to the person which wraps the products for shipping [8].

- **Automatic Guided Vehicles (AGV)**: AGVs are the most common solution when it comes to moving items around in a warehouse or also in other environments. AGVs are automated robots which are not completely autonomous but dependent on rails, magnetic wires, colored lines, or even lasers to reach their target. The first AGV was launched in the 1950s by a US-based company. This AGV did not really work with advanced technology, more it was a tow truck attached to a cable. Nowadays, AGVs are so advanced that they can communicate with their surroundings to make warehouses or production facilities a safer place and make the processes smoother [8, 33].

- **Automated Guided Carts (AGC)**: AGCs are a very cost-effective solution for cargo moving. AGCs can also be considered AGVs, but are more of a slimmed down version of them and possess the capability of moving smaller loads around. However, AGC are reliable, durable, and cheap, and they often depend on magnetic wires which count as the most affordable solution. They are mostly used to move contents from A to B [4].

- **Autonomous Mobile Robots (AMR)**: AMRs are the most advance types of robots used in warehouses. Those robots often make use of advanced camera systems as well as laser scanners, 3D-vision cameras, and proximity sensors and artificial intelligence. They are smarter and use less fixed infrastructure, to plan their routes. Depending on the specific AMR, it is capable of avoiding obstacles in its way and recognizing and interacting with people. All these capabilities make AMRs well-suited of making real-time decisions [16].

- **Articulated Robotic Arms**: Articulated Robotic Arms are standalone arms which are used for repetitive tasks in industry. They follow pre-programmed movements but are equipped with various sensors that can detect an object even if it is not exactly in the expected position. Those robots are also often revered to as joint robot manipulators or full robots, but can also be categorized further to cartesian robots, cylindrical arms, spherical arms as well as SCARA robots, which are used for pick-in-place assignments [8].

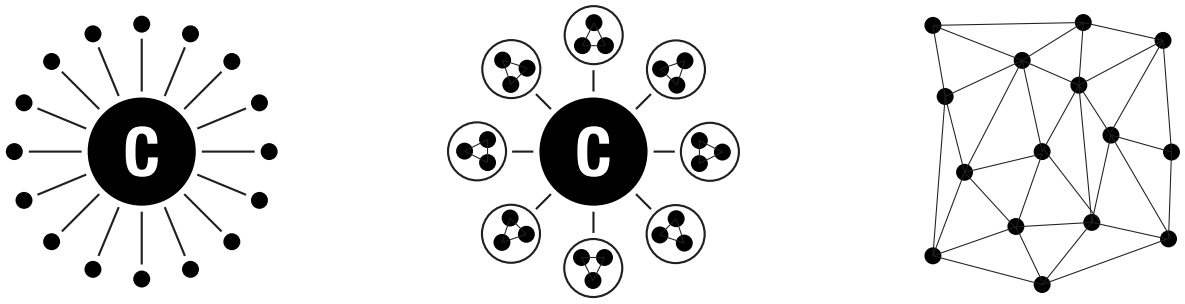
2.1.3 Collaboration Between Robots in a Warehouse Setting

As shown in Subsection 2.1.2 there exist many types of robots in a warehouse setting. All robots have their specialty for efficiently getting work done. The task of certain robots is to collaborate directly with humans in order to enhance their efficiency. However, this thesis will focus on collaboration between two or more robots. Therefore, a closer look is taken at mobile robots, like AGVs or AMRs.

Liu et al. [26] mention in their paper that when multiple robots are used in a collaborative setting, their coordination is a key concern. An entire task can typically be considered a chain of many smaller subtasks. These subtasks must be distributed among several robots, which requires some kind of scheduling. After the subtasks have been assigned to the robots, collaboration between the robots is required. In conclusion, it can be said that robot collaboration can be divided into two sub-problems: scheduling and robot interaction.

A multi-robots task involves, as the name already says, multiple robots that carry out a partial job of the bigger task. In this system, it is often the case that a single robot does not have full information on the task and does not know what the other robots task are, since they are only a small part of a distributed system. A challenge in dividing these tasks is, that it has been proven that task scheduling in a distributed system is NP-hard, however, research has found multiple strategies to schedule tasks in a distributed system as efficient as possible.

The scheduling of a task can happen in a *centralized*, *semi-distributed* or *fully distributed* way. If the scheduling is carried out centrally, there's a centralized instance that manages the task distribution between the robots. In this case, the robot just has to follow the command of the centralized system and execute it. In a fully distributed approach, there is no central instance for scheduling. With an approach like this, all the robots are equal agents and each of them can carry out the scheduling. The semi-distributed scheduling, in contrast, is a mix of the last two mentioned approaches. In this case, every robot is part of one cell that consists of multiple robots. The central instance does not assign a task to a single robot but to a cell. The robots in the cell then decide, on their own, who will carry out what part of the task [26].



(a) Centralized Scheduling

(b) Semi-Distributed Scheduling

(c) Fully Distributed Scheduling

Figure 2.1: Comparison of task scheduling strategies.

In order for multiple robots to collaborate, it is necessary to address the issue of local coordination. In theory, robots are always in a loop of three different states: *Look*, *Compute* or *Move* [11]. In the *Look* phase, the robot observes its environment with its sensors, in the *Compute* phase the robot computes its next action and in the move phase the robot carries out the computed action. From these different states, the theoretical concept of synchronous, asynchronous and semi-synchronous models can be derived. Robots in a fully synchronous model must always be in the same state out of the three. For example, if one is in the *Move* state, all the others are too. In contrast, in a semi-synchronous robot model, all active robots are only required to be in one of the three states concurrently. In an asynchronous model, it is not relevant which robot is in which state, all robots can be in a different state.

2.1.4 Multi Robot Communication

In order for robots to collaborate, they require some kind of communication between them so that they can plan their task and know which part of the task is assigned to them. As we have seen in Subsection 2.1.3 we can distinguish between centralized, semi-distributed, and fully distributed approaches in robot communication. For the last two approaches, robots have to communicate with each other directly. Ismail and Sariff [21] identify in their research two different kinds of communication between multi-agent robot systems (MARS).

One type of communication is implicit communication. In that case, robots make use of sensors embedded in or attached to the robot. With the sensors, robots in a MARS can sense other robots as well as obstacles in their way. One example of this is robots that are equipped with ultrasound sensors or cameras, which are used for local navigation. However, this type of communication is limited by the simplicity of the sensors.

To overcome this limitation, a more complex type of communication is required, called explicit communication. This kind of communication permits robots to directly exchange data between agents via broadcast messages or even one-on-one communication. The agents make use of built-in communication modules using radio modems, and wireless Ethernet cards. Over protocols like TCP/IP or UDP/IP, the messages can be exchanged. However, explicit communication also has its challenges. A control framework has to be designed which provides an efficient message transmission adapted to the environment. A suitable protocol has to be developed so multiple agents can exchange messages. Communication has not to be either full implicit or explicit. Some researchers also combine both communication methods.

2.2 Related Work

While this thesis aims to design and implement a framework that enables AGVs to collaborate on a task-level in a warehouse setting, there are projects with a similar, though not identical, approach. Cooperation among AGVs is not commonly observed due to their

design, which typically relies on following predetermined paths, limiting their adaptability and interaction capabilities. This thesis argues that introducing task-level cooperation between AGVs presents an intriguing opportunity to enhance efficiency. For example, a warehouse management system can group AGVs within a cell and attribute tasks to cells. Within such cells, AGVs could cooperate on a task-level to autonomously and optimally divide the task into non-conflicting sub-tasks.

However, it is important to cover these related works, since they may provide valuable insights and show existing challenges and solutions that can inspire the design of the proposed framework. The related work refers to solutions that involve multiple robots collaborating with each other, but will not be limited only to AGVs.

(a) Drive Unit Agent (DUA) (b) Pod (c) Inventory Station Agent

Figure 2.2: The most important components of the Kiva System.

One relevant and probably most famously known robot collaboration system used for warehousing is *Kiva Systems*, which was acquired by the Amazon Robotics in 2012 [22]. Kiva Systems consists of multiple autonomous robots. Their solution enables warehouses to efficiently pick up a large amount of customer orders in a warehouse. The system consists of three key parts: Mobile Robots (Drive Unit Agent (DUA)), pods, human workers with desktop computer (Inventory Station Agent (ISA)) and Software (Job Manager (JM)). The robots are of low height, which enables them to move under pods, which serve as a shelf for products that can be brought to human operators processing the items in the pods. The JM then coordinates the tasks, which are then executed by the AMRs. The implementation of the system eliminates the need for manual walking previously required by human workers. In a warehouse that makes use of the Kiva systems, workers are stationed at inventory points. The station tells the worker with lights, from what storage unit they have to pick the product, so that picking errors are reduced. To achieve the warehouse goals, the DUAs as well as the JM communicate with each other. The JM is responsible for handling the orders and has full knowledge of which pods contain which products. When an order is received, the JM assigns a task indicating that DUA X should deliver Pod Y to ISA Z. The JM then asks the ISA Z to ask for DUA X to pick up Pod Y. However, the ISA does not have to have knowledge where the DUA and the Pod

are located. The DUA can handle this by itself. The agents communicate over Extensible Markup Language (XML) messages with each other [35].

In another warehouse setting, Liu et al. [25] propose a *multi-AGV real-time collaborative operation (MARTCO)* method to resolve conflicts between multiple AGVs operating concurrency in a warehouse. They mention that AGVs often get in conflict with each other. MARTCO however, allows AGVs to autonomously determine their path with the help of multiple algorithms: Algorithm 1 ensures that the initial path that the AGV must follow is free from obstacles and other AGVs in its way. Additionally, it is assured that there is a minimal delay of the movement from the current location to the location where it starts its task. To find the most efficient route, an improved A* algorithm is used. Algorithm 2 is designed in a way that it guides the AGV safely from the starting point to the end point. It does not simply take the shortest path between those points. Rather, it has the capability to dynamically plan the route so that, if the path should be blocked unexpectedly by an obstacle, it can replan the route in real-time to evade the blockage. Algorithm 3 is responsible to maneuver the AGVs in the densely packed shelf area of the warehouse. It makes sure that the AGVs can enter the shelf area from the aisles without conflicting with other AGVs, by carefully managing the movements in the restricted area. Algorithm 4 is responsible for managing the AGVs movements in the aisles. It makes use of priority rules and adjusts paths in real-time to prevent collisions in this part of the warehouse.

With the *MARTCO* method, all the tasks arrive in a queue at a central instance, and are prioritized and sorted according to heir urgency. A list of all the AGVs is stored in a queue and sorted based on their current state (idle or executing task). The AGVs are assigned to a task according to the earliest available time first principle. After the task assignment, it is the responsibility of the robot to find the most efficient path. In their paper, it is assumed that the robots will have access to a 5G system, which enables them to communicate with each other. The aim of the communication between the robots is to exchange location information with other robots in real time.

Cardarelli et al. [5] propose in their paper a multi-agent reinforcement learning approach to enable collaboration between AGVs in a warehouse setting. The chosen approach should solve the challenges of traditional AGV systems, like collisions, congestion and general inefficiencies, in letting the robots learn and adapt to the environment. The algorithm is based on QMIX, which is a popular cooperative multi-agent reinforcement learning algorithm, but is extended furthermore to improve collision avoidance and collaboration in the path planing. The tasks that the robots get assigned are scheduled according to the request of the warehouse workers. AGVs are tasked with bringing the shelves to the workers, without collaborating with other robots. They only collaborate in this system for path planning, but not task execution. The team found that the proposed system outperforms traditional ones in terms of robustness, throughput, and travel time.

Another related work worth mentioning is the ARC5 from Servus. Servus develops various warehouse automation solutions that can be combined like a modular system based on customer needs. The Autonomous Robotic Carrier 5 (ARC5) is a robot that can perform various transport tasks on rails, depending on the equipment. The ARC5 can be equipped with totes to transport multiple items, but it can also transport individual

larger cardboard boxes or be equipped with workpieces. Because the robot operates on rails, it can reach speeds of up to 5 m/s. The software called Servus Manager manages the coordination of the ARCs at a central location over WLAN and uses algorithms to coordinate that the robots have as few idle runs as possible. However, collaboration between multiple robots to divide a single task and collaborate on a task-level in order to further increase efficiency is not the aim [14].

	Kiva Systems [35]	**MARTCO [25]**	**Multi-Agent RL [5]**	**ARC5 [14]**
Type of Robots	Autonomous Mobile Robots (AMRs)	Autonomous Guided Vehicles (AGVs)	Autonomous Guided Vehicles (AGVs)	Autonomous Robotic Carrier (ARC5)
Task Assignment	Centralized by Job Manager	Centralized with priority-based queuing	Centralized scheduling based on requests	Centralized via Servus Manager
Path Planning	Handled by individual robots	Multiple algorithms, including improved A* for efficiency	QMIX-based reinforcement learning for collision avoidance	Predefined rail system
Communication	XML-based communication between agents	Real-time communication, assumes 5G	Communication primarily for path planning	Centralized communication over Wi-Fi
Robot Collaboration Level	Medium	Medium	Medium	Medium
Collaboration Efforts	Conflict Resolution	Conflict Resolution and path optimization	Path planning	Swarm intelligence

Table 2.1: Comparison of related works concerning multi-AGV collaboration.

All the related works mentioned in this chapter are summarized in Table 2.1. All the solutions make use of warehouse automation in using robots to ensure efficient delivery. The robotic systems mentioned in all these related work contribute towards a lower demand for repetitive and expensive human labor.

While in warehouses, which make use of solutions by KIVA systems, still require human action to pick the products from the delivery pods, solutions like the ARC5 by Servus does not require any human interaction to complete its task. What also can be seen is that in theory, there are clear differences between AGVs and AMRs. However, the related work shows that in the case of MARTCO, it is referred to as AGVs, while in the case of KIVA Systems, it is referred to as AMRs, even though both systems perform very similar tasks.

As far as task scheduling is concerned, all robots receive their orders from a job manager, a

priority queue or some kind of manager. This means they all rely on a central instance that takes care of scheduling the individual tasks. Regarding multi-robot collaboration, some of them are mentioned as communicating with each other. However, this only concerns communication, either for path planning or conflict resolution. None of the related works collaborate with other robots to split and accomplish a single task together.

Chapter 3

Design

Based on the background concepts and related work covered in Chapter 2, this chapter will design a solution to enable AGVs to split and complete tasks through task-level collaboration. Section 3.1 will describe in what setting the robots will operate and what is assumed that the robots are capable of. Section 3.2 will give an overview of what tasks the AGVs should be more efficient to solve than traditional AGVs. Section 3.3 will describe how robots can communicate with each other.

3.1 Scenario

The prototype that is designed in this thesis focuses on the software of the AGV itself. However, there are multiple other components which have to be considered for designing the prototype. The most crucial components next to the AGV itself are the environment the AGVs operate in, the WMS that schedules tasks, as well as the freight that has to be transported. To give an overview of the characteristics and abilities of each component, every component is explained in a subsection below.

3.1.1 Environment

The environment in which the AGVs operate is a warehouse. The warehouse area may have obstacles, which can be shelves or other static obstacles. The warehouse area is divided into multiple cells. For each cell, there is a fixed number of AGVs that operate in it, and the AGVs do not leave the cell. In every cell, there are multiple handover points, at which AGVs can pick up or drop-off freight.

In order that the AGVs can perceive the environment, the floor is equipped with different types of guides. At every handover point, there is some kind of marker. The marker tells the AGV at which location inside the cell it is located. The marker also functions as an alignment aid so that the freight transported by the AGV can be placed as precisely as possible. The second type of guides are colored lines, which always connect two handover points. The lines allow the AGV to move from one handover point to another.

3.1.2 Warehouse Management System (WMS)

The WMS is the instance of the warehouse which has the knowledge of all the orders that have to be processed, as well as knowledge about which fright is located at which location. However, the WMS does not know the location of every single AGV. Since the warehouse makes use of semi-distributed scheduling and is divided into different cells, it just has to know the coordinator AGV of each cell.

If the WMS processes, for example, an order to transport freight from a specific location to another, it just has to identify in which cell the freight is located in. Upon identifying the cell, the WMS assigns the task to the responsible coordinator, which then further distributes the task to the AGVs in its cell.

3.1.3 Automated Guided Vehicle (AGV)

The AGV is a vehicle capable of moving around in its environment on four wheels. Every wheel can rotate in both directions, enabling the robot to turn on the spot and drive curves. The robot has a single arm capable of moving on the Y and Z axis. A gripper attached to the arm, making it capable of gripping freight like parcels. The central sensor which makes the robot aware of its environment is the camera, which can detect lines and markers. The brain of the robot is a computer which takes video input from the camera and gives output to the wheels, robotic arm and griper. It additionally has the capability to communicate over a wireless network.

3.1.4 Freight

The freight, a cubic shaped parcel that has the size to be picked up by the gripper of the robot and the weight that the robot is capable of carrying it through the warehouse.

3.2 Tasks

As shown in the related work, the documented AGV systems make little use of collaboration through communication. They only collaborate in the way that they can resolve conflicts which would hinder them to execute their task. While conflict resolution is critical to ensure a resilient system, this thesis tries to go beyond it.

This design uses explicit communication to create a semi-distributed system, where multiple AGVs make use of task level collaboration. The system is designed in a way that a single task is split into multiple smaller tasks. This thesis focuses on AGVs that have a task of moving freight from one place to another. AGVs that deliver this freight have to move along lines which serve as guides to orient themselves, which makes it not possible that two AGVs can pass each other. Instead of just handling conflicts, they should take advantage of task splitting and collaboration.

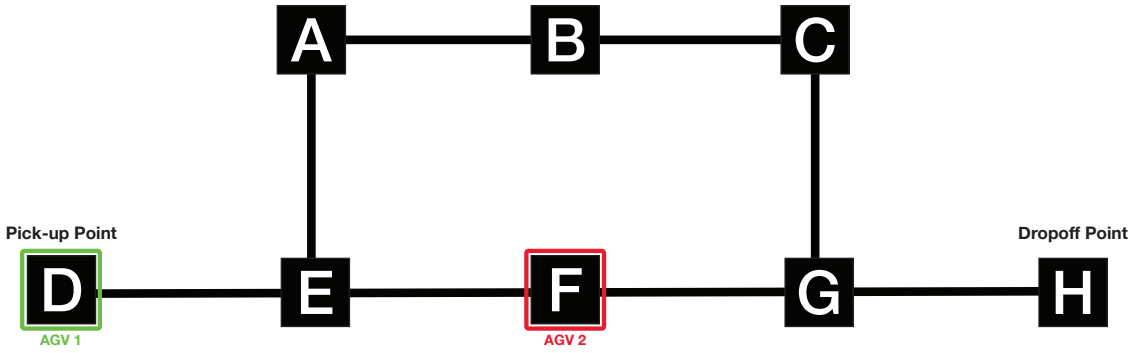


Figure 3.1: An example of a task environment.

This can be shown best in an example. In Figure 3.1 we have a network in which two AGVs operate in. Each node could be a potential pick-up or drop-off point. In this example, the cell has the task of moving freight from the node D to the node H. The cost for an AGV to move from one node to another is always the same. The shortest path would be to follow the route $D \rightarrow E \rightarrow F \rightarrow G \rightarrow H$. However, there is an AGV located at node F which AGV 1 cannot pass. In a system where AGVs are only capable of conflict resolution, the AGV would move the path $D \rightarrow E \rightarrow A \rightarrow B \rightarrow C \rightarrow G \rightarrow H$. Therefore, it would have to pass two edges more, making the path longer and requiring the AGV to take turns. The prototype designed in the scope of this thesis should be capable of taking all possibilities in account to achieve this task. It should be able to split the task between two AGVs if this leads to a faster task execution. In the case of the shown example, the first AGV would deliver the freight to node F where the second AGV would take over and deliver it to the dropoff point H.

3.3 Communication Protocol

The components that are communicating which each other are depicted in Figure 3.2. The WMS and all the robots from different cells are all connected over the same network. Since they are all connected, they could directly talk to each other. However, since the protocol tries that every actor in the system only has to have as little knowledge as necessary, the protocol will make use of a semi-distributed approach and divide different environments into different cells. The WMS initially starts a task with, assigning it to a specific cell in which the task should be completed. It just has to know who the responsible coordinator in the specific cell is and assigns it the task. The coordinator then splits the task into subtasks and assigns them to the other agents in its cell that are needed for execution.

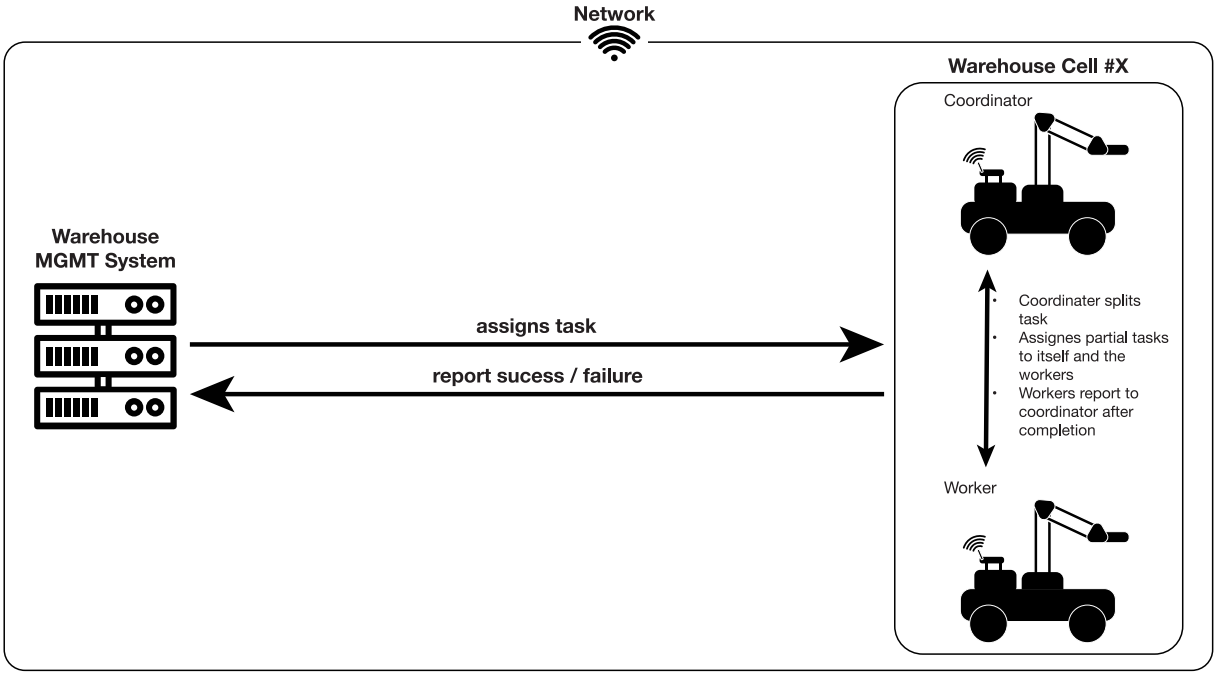


Figure 3.2: Communication between the AGVs.

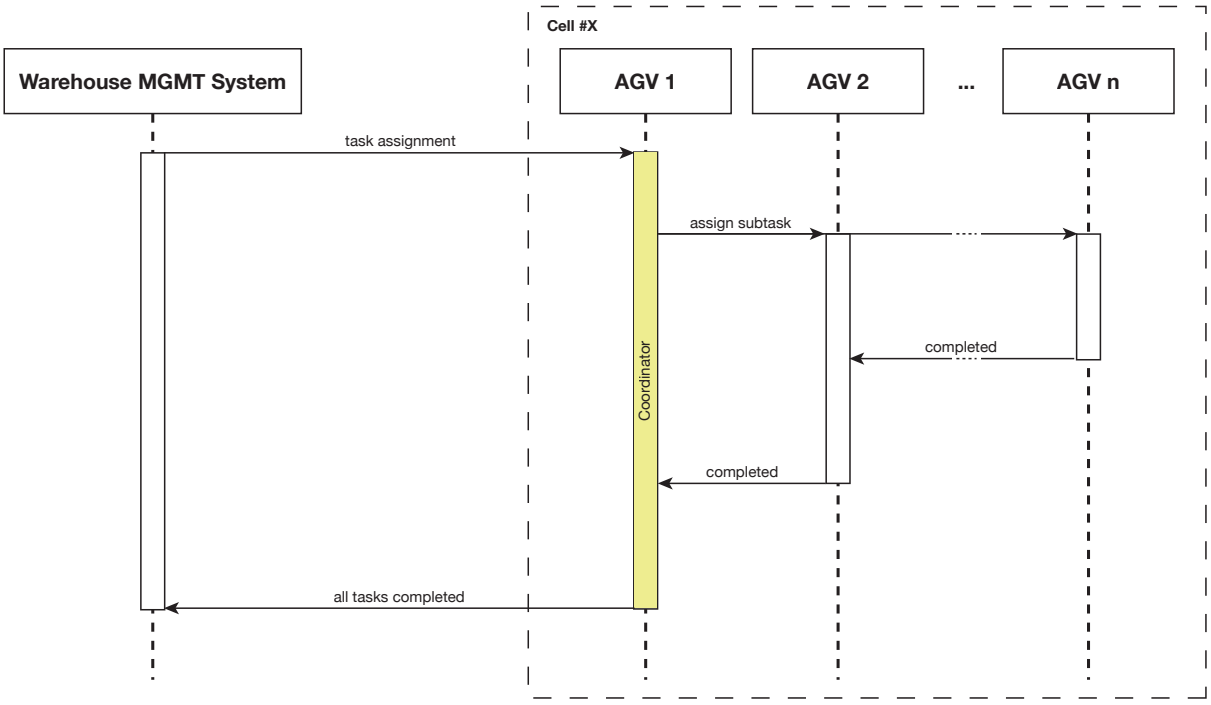


Figure 3.3: Sequence diagram of the communication.

The process can be shown in a sequence diagram (cf. Figure 3.3). The WMS assigns the complete delivery task, containing initial pickup point and final dropoff point, to the coordinator of the cell, where the task should be executed in. Upon receipt of the task, the coordinator uses its knowledge about the other AGVs in its cell to decide on how to split the task into multiple subtasks. After dividing the task, the coordinator sends a

multicast message to all AGVs in the cell, informing them about the subtasks. After all AGVs have been informed, the first AGV in line starts with the execution of the initial subtask. Upon completion, it notifies the next AGV in line. This process will persist until the last subtask is completed. Upon completion of the last subtask, the coordinator is notified, which will report back to the WMS that the complete delivery task has been achieved.

Chapter 4

Implementation

To implement the designed prototype, various hardware, as well as software, is required. With the hardware, there come certain requirements, which must be considered. This chapter will first focus on the hardware requirements and in the second part on the software aspects.

4.1 Hardware

4.1.1 AGV

Since this thesis focuses on multiple AGVs in a warehouse setting, a suiting robot must be selected. AGVs as already mentioned in Chapter 2 often follow magnetic strips, colored lines or operate on rails. Since the latter is not feasible in the scope of this thesis, it was decided to make use of a robot operating on wheels, which also can simulate operation on rails to a certain degree. Since the focus of this thesis is mainly on the communication and task splitting and less on the robot itself, the selection was limited to a robot that was ready to use out of the box. For this reason, the choice fell on the DJI Robomaster EP Core, which fulfilled all requirements to achieve this project.

The Robomaster EP Core [10] is an educational robot that is designed for experimenting and learning about robotics and engineering in general. The robot has omnidirectional wheels which allow it to move in the X, Y and Z directions. The robot has an arm which is controlled by two servo motors and has a gripper, which can be used to lift, transport and lower smaller objects. For orientation purposes, the vehicle is equipped with a video camera. All these parts are controlled by an intelligent controller, to which the operator can connect over Wi-Fi remotely. The robot can either be remotely controlled over a smartphone app which provides a real time video stream, or it can be programmatically controlled with a graphical programming language, which makes use of different building blocks. More advanced users can also program it using Python. The controller also features a USB port that allows to connect a single-board computer for issuing commands to the robot.

Figure 4.1: The DJI Robomaster EP Core.

4.1.2 Single-Board Computer (SBC)

The aim of the robot is that it can receive tasks from the WMS or communicate with other AGVs. Since the controller of the DJI Robomaster EP-Core is incapable of autonomously operating and communicating with other robots, an additional controller has to take over this responsibility. Therefore, an SBC has to be used. The SBC can also be connected to the USB port of the robot to give commands directly to the robot, while using its wireless communicating capabilities to exchange information with other robots. Additional requirements for the SBC aside from communicating wirelessly and having a USB interface are that it is powerful enough to run the software to operate the AGV while using power sustainably. Sustainable energy use is critical since the SBC takes power from the battery that is also used for running the robot. Due to the energy consumption requirement, the Raspberry Pi Zero 2W was favored over the more powerful Raspberry Pi 5 due to the energy consumption of 2.7W compared to 0.7W in the idle state [13].

Here is an overview of the specifications of the Raspberry Pi Zero 2W [12]:

- 1GHz quad-core 64-bit Arm Cortex-A53 CPU
- 512 MB SDRAM
- 2.4GHz 802.11 b/g/n wireless LAN
- Bluetooth 4.2, Bluetooth Low Energy (BLE)

4.2 Robomaster's Software Development Kit (SDK)

The Robomaster SDK [9] is used to write software to operate the robot. The SDK is available for Python Versions up to 3.8.9 and can be downloaded over pip or built on the machine itself.

A robot object can be created over the robot object exported by the robomaster library. Afterward, it has to be initialized in connecting to the robot. The SDK offers multiple ways to connect to the robot: `ap, sta, rndis`. `ap` is used to connect to the robot through its access point. Therefore, the robot has to be configured so that it creates an access point and the device running the code has to be connected to it. The `sta` connection type enables to communicate through an external Wi-Fi network, to which the robot and the command device have to be connected to. The `rndis` mode allows that a device that is linked to the robot via its USB port to connect to it.

```
from robomaster import robot

ep_robot = robot.Robot()
ep_robot.initialize(conn_type="sta")
```

Listing 4.1: Initialization of a DJI RoboMaster robot with station mode connection.

After the initialization, the robot object offers multiple modules, the ones important in the scope of this thesis are: `chassis, camera, vision, robotic_arm, gripper`

The chassis provides the functionality to move the robot around. With the `move` function, it can be specified in which direction the robot should move in meters on the X, Y or Z axis. Moreover, it can be specified at what speed the robot should be moved. The `wait_for_completed()` function ensures that the program does not continue until the movement is finished. Additionally, the `drive_speed` function allows the robot to move with a specific speed in one of the directions until the timeout specified in seconds elapses.

```
chassis = ep_robot.chassis
chassis.move(x=0.5, y=0, z=0, xy_speed=0.5).wait_for_completed()
ep_chassis.drive_speed(x=0.5, y=0, z=0, timeout=5)
```

Listing 4.2: Chassis Control

The camera module of the robot can be used to stream the video captured by the robots' camera to the device. With the function, `read_cv2_image` a frame from the video stream can be captured which can be used for further image processing on the device.

```
camera = ep_robot.camera
camera.start_video_stream(display=True)
img = camera.read_cv2_image(strategy='newest')
camera.stop_video_stream()
```

Listing 4.3: Camera Control

The SDK also comes with a vision module, which has predefined vision options. The `sub_detect_info` function can be used to subscribe to lines in predefined colors red, blue, and green. Furthermore, it can observe markers that come with the robot. In the case of the example shown in Listing 4.4 the robot will observe if it sees blue lines. If this is the case, it will call the function `on_detect_line`, which can be defined by the programmer, with a list of points where the line is. It has to be noted that the robot cannot observe a marker and a line or lines in different colors at the same time, which makes this functionality very limited.

```
vision = ep_robot.vision
vision.sub_detect_info(name="line", color="blue", callback=on_detect_line)
```

Listing 4.4: Vision Module

The gripper module of the robot can also be controlled directly. The strength that the gripper should use to open and close can be specified, so that the freight the robot picks up is not exposed to excessive forces. Next to the gripper, the robotic arm can be brought into the desired position so that it can pick up freight or position the camera that is mounted on top of the robotic arm.

```
robotic_arm = ep_robot.robotic_arm
gripper = ep_robot.gripper
gripper.open(power=50)
robotic_arm.move(x=130, y=-110).wait_for_completed()
gripper.close(power=50)
robotic_arm.move(x=0, y=0).wait_for_completed()
```

Listing 4.5: Robotic Arm and Gripper Control

4.3 System Design and Architecture

In the setting of this project, the AGV has different tasks. The robot has to have information about its environment and hold information about other robots in its cell. Additionally, the robot needs to be able to send and receive messages from the WMS as well as from other robots. Additionally, it needs the capability to do estimates of execution costs and schedule tasks. Lastly, it also has to be able to coordinate the robots' movement.

To achieve all these, an Object-Oriented Software System is designed, which consists of multiple classes with different responsibilities. Python is selected as the programming language, since the SDK is available in this language.

To understand the architecture and the relationship between the classes, every class is explained below, with insights into the code where necessary.

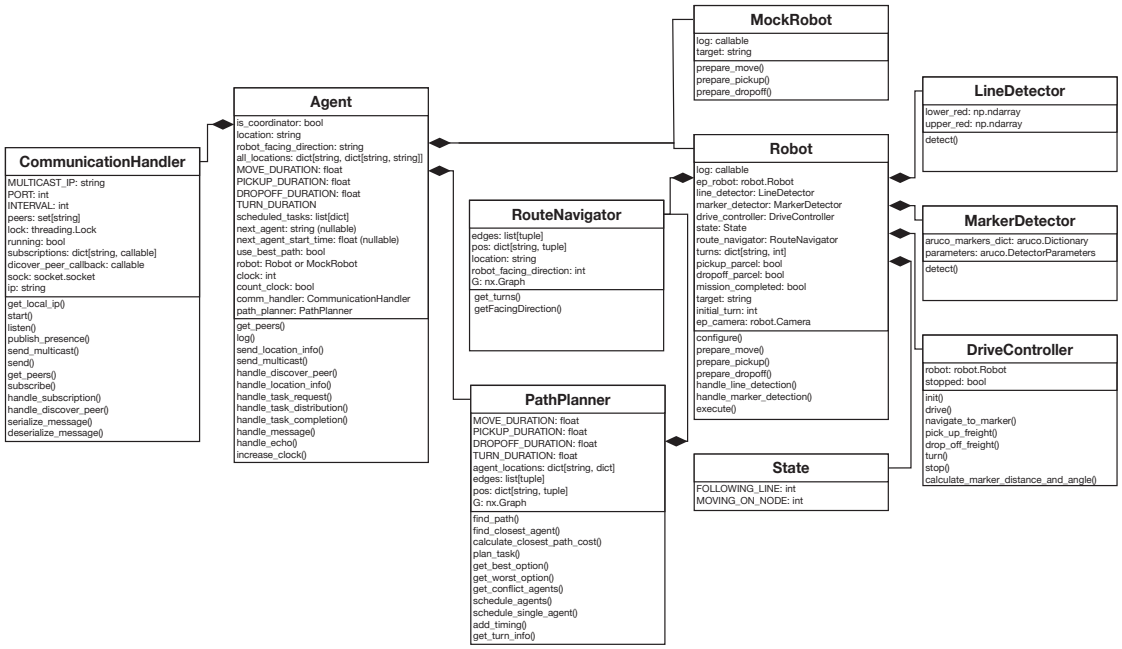


Figure 4.2: The UML diagram of the architcture.

4.3.1 Agent

The agent object is the core of the entire system and is initialized when the program is started. All the necessary information is passed on to the agent object through environment variables so that the agent has all the knowledge it needs to operate. As noted in the design, the same software runs on every AGV in the warehouse. The agents only differ in that one agent per cell operates as a coordinator and receives the individual tasks from the WMS and divides them up further. As a result, when the agent object is initialized, the variable `is_coordinator` tells the agent whether it should act as a coordinator or not.

The agent also needs information about its environment so that it can operate. For this reason, the variables `location`, `pos`, `edges` and `robot_facing_direction` are given to the agent. The variable `location` specifies which node the robot is at when it starts. The variable `pos`, which stands for position, contains the information where the individual nodes are located, which is later important information for the `RouteNavigator` and the `Pathplanner`. The variable `edges` contains the information about which nodes are connected to each other. Another piece of information that should not be neglected is the direction in which the robot is facing when starting because the turns are calculated based on this information, which is why this information is passed on in the `robot_facing_direction`.

To calculate the costs of the individual paths, the agent is also provided information about how to carry out certain operations. The variables `MOVE_DURATION`, `PICKUP_DURATION`, `DROPOFF_DURATION`, `TURN_DURATION`, which are passed on to the agent as JSON as durations, specify how long the AGV needs to move from the current node to a neighboring node, how long the AGV needs to pick up a load and put down a load, and finally, how long

an AGV needs to turn 90 degrees. Furthermore, there is the variable `use_best_path` which is only needed for debugging and per default set to `True`. When set to `False` the AGVs always choose the to split the task into as many subtasks as possible.

Since it would be too extensive to discuss every detail of the agent class, only selected code snippets are presented here which are of high importance for the core functionality of the class.

The architecture of the prototype tries to encapsulate different responsibilities of the system into different classes. Nevertheless, information must be exchanged between them. One challenge was to exchange information between the `Agent` and the `CommunicationHandler` class. The `CommunicationHandler` always has to listen for incoming messages also when the `Agent` object is busy. This problem was solved with a subscription approach, as can be seen in Listing 4.6.

```
1 self.comm_handler = CommunicationHandler(self.handle_discover_peer)
2 self.comm_handler.subscribe("LOCATION_REQUEST", self.send_location_info)
3 self.comm_handler.subscribe("LOCATION_RESPONSE", self.handle_location_info)
4 self.comm_handler.subscribe("TASK_REQUEST", self.handle_task_request)
5 self.comm_handler.subscribe("TASK_DISTRIBUTION", self.handle_task_distribution)
6 self.comm_handler.subscribe("EXECUTE_TASK", self.handle_task_completion)
7 self.comm_handler.subscribe("MESSAGE", self.handle_message)
8 self.comm_handler.subscribe("ECHO", self.handle_echo)
```

Listing 4.6: Initialization of subscriptions.

In the initialization process of the `Agent` object, the agent creates an `CommunicationHandler` object. After this process, the agent subscribes to the type of message as well as which function should be called upon receiving this type of message. This allows the agent to execute different commands based on the type of message received. As shown in Listing 4.6, the agent is subscribed to seven different types of messages:

- **LOCATION_REQUEST**: Upon detecting another agent in the same network, the detecting agent sends a request for location information of the newly detected agent. The receiving agent of this request then sends the information on which node the agent is located, together with the information in which direction the robot faces.
- **LOCATION_RESPONSE**: After asking another agent for its location, the agent optimally receives a location response. Having received this message, the agent stores the information in a dictionary for task splitting in the future.
- **TASK_REQUEST**: If the WMS wants freight to be transported, it sends a task request to the coordinator in the cell. When the coordinator agent receives this message, it splits the task and informs all other agents needed for the operation about their subtasks.
- **TASK_DISTRIBUTION**: When all subtasks are created, all the agents that are needed for this task get a task distribution message, where they get information about all the subtasks of all agents involved. Based on this information, every agent

knows its assigned subtask and knows which agent does a subtask before or after themselves.

- **EXECUTE_TASK**: When the coordinator informs all the agents about their subtasks, it sends an execute task to the AGV that has to start the execution. When the first one has executed its subtask, it informs the second one. This goes on until all the agents have completed their subtask. The last agent in line sends an execute task message back to the coordinator, which then knows that all the tasks have been completed.

- **MESSAGE**: This type of message is used for debugging and checking the message transmission. Upon receiving a message of this type, the agent logs the message to the console.

- **ECHO**: This message type also serves for debugging purposes. Upon receiving this message, the agent answers back with the same message to the sender.

4.3.2 CommunicationHandler

The `CommunicationHandler` is concerned with the exchange of information between the robots and provides a communication interface for the agent class and takes over the technical concerns of transferring data.

The class uses the socket library of python and uses UDP sockets to transmit the messages to minimize overhead a more high-level library would have had. While TCP sockets would have offered a more reliable data transfer, the decision fell on UDP sockets. UDP sockets do not only offer a faster message transmission, but they also allow multicasting messages, which was made use of by building this prototype.

In the initialization process, a UDP socket is initialized, which communicates over the IPv4 protocol. The socket is configured in a way so that reuse the address immediately after the socket is closed. The socket is bound to the multicast address `224.1.1.1` and listens on port `5004`. Upon starting the `CommunicationHandler` the object creates a separate thread in which it listens for incoming messages.

As already mentioned in Subsection 4.3.1 does the `CommunicationHandler` provide a subscription service for the `Agent` class. Additionally to that, however, the class also provides two methods for sending data. With the `send_multicast` an agent, can send data to all peers in the network. This is used in the case of subtask distribution, where all agents have to have full information about all the tasks. When the notification of all the peers is not necessary, the agent can also use the `send` function, where it has to specify the IPv4 address of the receiver and only the agent with this address will receive the message. This type of message transmission is used in the case of the `EXECUTE_TASK` message.

Since the class makes use of the low-level socket library, which transmits data in a sequence of bytes, the serialization, and deserialization of the messages exchanged has to be done by the class itself.

```
def serialize_message(self, type, message, address):
    return json.dumps({
        "type": type,
        "message": message,
        "address": address
    })

def deserialize_message(self, payload):
    payload = json.loads(payload)
    return [payload["type"], payload["message"], payload["address"]]
```

Listing 4.7: Serialization and deserialization of messages.

As shown in Listing 4.7 the message type and the message itself have to be transmitted together with the address of the sender. To serialize the message, all the information is encoded in a JSON string and, when received, decoded back into JSON and returned in a list. This approach keeps everything simple and makes the encoded string still human-readable.

4.3.3 PathPlanner

The `PathPlanner` class takes on all the responsibilities for task splitting and path planning. Upon initialization, the agent passes all the information about the environment and the durations of all the task types to the object so that it can split the task into optimal subtasks.

Since the `PathPlanner` makes decisions based on a network, the class makes use of pythons networkx library. This library provides all the functionality for graph problems like finding the shortest path. In the initialization process of a `PathPlanner` object, all the edges are added to a networkx graph, which is later on used for splitting the tasks.

When an agent receives a task from the WMS and has to split it, it calls the `plan_task` function and passes the locations of all the AGVs together with the location and end destination of the freight to the path planner. The path planner goes through different steps to split the task into the most optimal subtasks.

1. All simple paths from the location to the destination in the graph are calculated.
2. For every path, it is calculated which AGV is the closes to the freight. This AGV is selected as the first to move to the location and pick it up
3. Given that the first AGV has picked up the freight, it is checked if there are any other AGVs located on the delivery path, which would pose an obstacle to the AGV and could not be bypassed (cf. Listing 4.8).
4. To complete the task, the AGV has to collaborate with the AGVs in its way. Therefore, handover points are calculated.

5. For each AGV on the path, given the handover points, it is calculated how the AGV can move to this point, deliver the freight to the next handover point and make room for the next AGV in line.

6. After dividing the task into subtasks, a function calculates the total duration of all operations performed by the AGVs for each subtask, based on their moves, turns, pickups, and drop-offs.

7. After the durations which were calculated for all paths found in the first step, the option which takes the least time is selected and further executed by the agent.

```
def get_conflict_agents(self, path, agent_locations, exclude):
    conflict_agents = []
    for agent in agent_locations:
        if agent_locations[agent]["node"] in path and agent != exclude:
            conflict_index = path.index(agent_locations[agent]["node"])
            conflict_agents.append((agent, conflict_index))

    sorted_conflict_agents = sorted(conflict_agents, key=lambda x: x[1])
    return [a[1] for a in sorted_conflict_agents], [a[0] for a in
        sorted_conflict_agents]
```

Listing 4.8: The identification of conflict agents on a path.

4.3.4 Robot

The `Robot` class coordinates all functionalities so that the robot can execute the tasks assigned by the agent. It evaluates the environment and based on that it decides how it can achieve its task.

The implemented class is strongly dependent on two computer vision modules, one that is capable of detecting colored lines, which are paths for AGVs to move one and another to detect nodes which are handover points for freight.

An additional module important to the `Robot` class is the `DriveController`. This controller allows the class to give operation commands to the AGV. And lastly, the robot needs the Route navigator module to calculate the turns the AGV has to take at each node to move to its destination. All these modules are initialized during the initialization process of the robot. So that the computer vision modules can detect lines and nodes in the environment, they need images they can process. For that, a connection to the DJI Robomaster EP Core is established, and a video stream is started. The `Robot` object is ready to operate after all these initialization steps.

The agent makes use of three functions provided by the `Robot` class, namely the `prepare_move`, `prepare_pickup`, `prepare_dropoff`, which all take the name of the target node as an input. They are named with the prefix *prepare* because they do not directly execute the command to the robot but make use of the computer vision modules, to detect the target and

then pass the information to the `DriveController` object which then executes commands on the robot.

To demonstrate how the robot operates, a closer look must be taken at the process that starts when the agent wants to move the robot. The agent calls the method shown in Listing 4.9 and specifies the target it wants to move to. Upon logging and storing the target as a class variable, the robot uses the `route_navigator` to calculate the turns it has to take. The `mission_complete` variable is set to `False` and the robot goes in the `FOLLOWING_LINE` state. After setting up all those variables, the robot calls its execute function.

```
def prepare_move(self, target):
        self.log(f"Move to {target}")
        self.target = target
        self.turns, self.initial_turn = self.route_navigator.get_turns(target)
        self.turns[target] = 0
        self.mission_completed = False
        self.pickup_parcel = False
        self.state = State.FOLLOWING_LINE
        self.execute()
```

Listing 4.9: Preparing the robot to move.

The execute function is the core of the `Robot` class and makes use of its computer vision modules. First the `drive_controller` is commanded to execute its initial turn, thereafter, the robot goes in an endless loop.

At the beginning of the loop, the newest image available is stored. The image is then processed so that two new images are created: the `marker_image` and the `line_image`, which are both used for the concerning computer vision module. The image used for detecting node markers is cut so that the middle third and the lower half of the image remain, and the image used for detecting the lines is cut in a way that only the middle third of the image remains. These are both measures so that the robot does not detect any neighboring lines or markers in the distance it should not detect.

Upon preparing the images for further processing, the images are passed to the corresponding modules. If a node marker is detected, the robot changes its state to the `MOVING_ON_NODE` state so that no further actions are executed. After changing state, the class starts a new thread where the detection of the marker is handled based on the corners and ID of the markers that were detected.

If no marker is detected, the robot follows the line based and the centroid on the line that is detected. After processing the tasks specified in the `handle_marker_detection` or the `handle_line_detection`, the `mission_completed` variable is set to true, which breaks the endless loop, which makes the robot ready for further commands.

```
def execute(self):
        self.drive_controller.turn(self.initial_turn)
        try:
                while True:
                img = self.ep_camera.read_cv2_image(strategy='newest')
                marker_image = img[img.shape[0]//2:,img.shape[1]//3:img.shape[1]*2//3]

                line_img = img[:, img.shape[1]//3:img.shape[1]*2//3]

                line_detected, centroid, img_with_line =
                    self.line_detector.detect(line_img)
                ids, corners, img_with_markers =
                    self.marker_detector.detect(marker_image)

                threads = []

                if ids is not None and self.state == State.FOLLOWING_LINE:
                        self.state = State.MOVING_ON_NODE
                        marker_thread =
                            threading.Thread(target=self.handle_marker_detection,
                            args=(ids, corners, img_with_markers))
                        threads.append(marker_thread)

                elif line_detected and self.state == State.FOLLOWING_LINE:

                        self.handle_line_detection(img_with_line, centroid)

                for thread in threads:
                        thread.start()

                if self.mission_completed:
                        break
        except KeyboardInterrupt:
                print("Stopping the application...")
```

Listing 4.10: Executing the move specified by the agent.

4.3.5 MockRobot

As the name suggests, the `MockRobot` class is used to simulate the behavior of the real robot. As noted in Subsection 4.3.1, the constructor of the `Agent` class includes a parameter called `use_mock_robot`. The variable enables conditional injection. When set to `True`, a `MockRobot` object is initialized on the robot field instead of a `Robot` object. The agent does not to change behavior due to that, since the `MockRobot` class implements the same functions which are implemented in the `Robot` class. When for the function `prepare_move` is called on the mock object, it only logs the execution of the command and takes no further action.

The `MockRobot` robot class is crucial for testing the software. When the software is run on the DJI Robomaster EP Core, the robot has to be placed back to its starting point after every time, which can be time-consuming. Having mock robots also allows testing on more robots than which are available.

4.3.6 RouteNavigator

The `RouteNavigator` class is responsible for the turns of the robots. When feeding the edges of the graph, that the robot is operating on, to a networkx graph, paths can be found on it, but it cannot be determined how the nodes are located to each other. This makes it impossible to determine the turns a robot has to take at every node to reach its destination. There comes the `RouteNavigator` class into play, which provides two functions: the function `get_turns`, which takes a target as a parameter, and the function `getFacingDirection`.

The first function calculates, based on the current location and facing direction of the robot, which are passed through the constructor, the turns that have to be taken at each node to reach the destination. This information is then returned as a dictionary, which allows fast access for the robot. The second function, when executed, returns the current facing direction of the robot, that is stored inside the object.

4.3.7 Computer Vision Classes

As already mentioned in Section 4.2 is the SDK of the DJI Robomaster EP Core equipped with a vision module, that is capable of detecting lines and markers. However, as it turned out in first experiments with the robot, this functionality is very limited in multiple ways: The SDK only allowed to detect lines in predefined colors (red, green, blue). This appears to be sufficient in the beginning, but depending on the light or colored tape used can lead to lines not being detected correctly. Another problem that appears is that the SDK does not allow to subscribe to detecting markers and colored lines, or to detecting two different colored lines concurrently, which makes it unfeasible to navigate the robot to the correct target.

After those findings in the preliminary experiments, it was decided to implement the computer vision functionality separately, which allowed creating the features that the SDK did not offer. Both classes make extensive use of pythons OpenCV [28] library.

4.3.7.1 MarkerDetector

The `MarkerDetector` class provides the functionality to detect ArUco markers (cf. Figure 4.3), which are used in this project to mark handover nodes. These markers are easily detectable by the ArUco module of the OpenCV library, making them suitable for this project. The class initializes a dictionary of all ArUco markers available together with the detector parameters. When the `dectect` function of the class is called, markers are detected on the image. If markers are found, the coordinates of the corners and the IDs of it are returned, as well as the modified image with contours around the markers for debugging reasons. The IDs of the markers, which are integers, are mapped to the nodes named with letters in the `Robot` class.

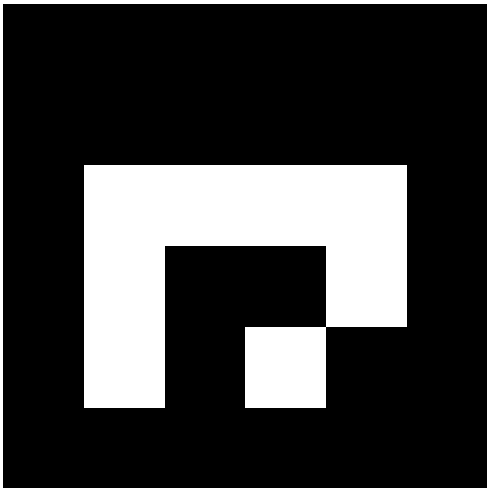


Figure 4.3: An ArUco marker with the ID 1.

```
def detect(self, image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    corners, ids, _ = aruco.detectMarkers(gray, self.aruco_markers_dict,
        parameters=self.parameters)

    if ids is not None:
        image = aruco.drawDetectedMarkers(image, corners, ids)
        return ids, corners, image
    return None, None, image
```

Listing 4.11: Detection of a marker in the image captured by the AGV's camera.

4.3.7.2 LineDetector

The `LineDetector` class provides the functionality to detect lines. In the initialization process, upper and lower HSV values for the color red are defined to create a range that allows the robot to detect the lines it should follow.

The class offers a single function called, `detect` which takes an image detected by the robot as an input. It is checked, if the contours of these colors are detected on the image. Additionally, it is checked if the contour is large enough, so that areas in the image that resemble the line but are not actually the line are excluded. If a real line is detected, the centroid of the line is returned together with the manipulated image that has drawn the contour around the line for debugging reasons. With this information, the robot is capable of following the line.

```
def detect(self, image, min_size=100):
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv, self.lower_red, self.upper_red)
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

```
    if contours:
        largest_contour = max(contours, key=cv2.contourArea)

        if cv2.contourArea(largest_contour) >= min_size:
            epsilon = 0.01 * cv2.arcLength(largest_contour, True)
            approx = cv2.approxPolyDP(largest_contour, epsilon, True)
            cv2.drawContours(image, [approx], -1, (0, 255, 0), 2)

            M = cv2.moments(largest_contour)
            if M['m00'] != 0:
                cx = int(M['m10'] / M['m00'])
                cy = int(M['m01'] / M['m00'])
            else:
                cx, cy = 0, 0

            return True, (cx, cy), image

    return False, (0, 0), image
```

Listing 4.12: Detection of a line in the image captured by the AGV's camera.

4.3.8 DriveController

All command chains that have to be given to the AGV are located in the `DriveController` class. The class encapsulates all the functionalities of the robot and provides them to the `Robot` class.

This can be best shown in a code snipped shown in Listing 4.13. The function makes use of the classes utility function `calculate_marker_distance_and_angle` which translates the corners of the detected markers into a distance in meters and also returns the angle that the AGV has to turn to center the gripper to the pickup destination. Before the AGV does its first move, the griper is opened and the robotic arm is brought into position to pick up the parcel. After completion, the robot moves two thirds of the distance, grips the freight and moves its arm back to the initial position. Upon completion of this, the robot moves on the node and turns into the specified direction. As one can notice, the function calls the `sleep` function of the time module quite often. This is necessary since the Robomaster SDK only offers on some functions the `wait_for_completed()` option, and on others not. The robot would not be capable of running all the commands without them.

```
def pick_up_freight(self, corners, turn_angle):
        angle, distance = self.calculate_marker_distance_and_angle(corners)
        self.turn(angle)
        time.sleep(1)
        self.robot.gripper.open()
        time.sleep(2)
        self.robot.robotic_arm.move(x=130, y=0).wait_for_completed()
        self.robot.robotic_arm.move(x=0, y=-120).wait_for_completed()
        self.robot.chassis.move(x=2*distance/3, y=0, z=0, xy_speed=0.6,
            z_speed=0).wait_for_completed()
```

```
10          self.robot.gripper.close()
11          time.sleep(2)
12          self.robot.robotic_arm.move(x=0, y=120).wait_for_completed()
13          self.robot.robotic_arm.move(x=-130, y=0).wait_for_completed()
14          time.sleep(0.5)
15          self.robot.chassis.move(x=distance/3, y=0, z=0, xy_speed=0.6,
                z_speed=0).wait_for_completed()
16          self.turn(turn_angle)
```

Listing 4.13: Command chain of picking up a parcel

Chapter 5

Evaluation

5.1 Evaluation Criteria

The question pursued by this master's thesis is whether the collaboration of two or more AGVs instead of a single one to complete a task can lead to increased efficiency and speed. AGVs typically do not cooperate at the task-level, by design, however they operate at considerably faster speeds given they have a prepared environment (*e.g.,* track or rail). For this reason, this thesis created an experimentation scenario and used a prototype, implemented in Chapter 4, that allows the collaboration of several AGVs to complete a task.

In the evaluation process, the implemented prototype is tested for various criteria. There are several sub-criteria that must be achieved in order to reach the main criterion. The sub-criteria concern the communication and navigation of the AGV. The main criterion concerns the overall goal of this master's thesis. The criteria are as follows:

- **Main Criterion**
  - A situation exists where the software prefers, of all possible task execution options, the one that favors multi AGV collaboration over single AGV execution.
- **Communication Criteria**
  - An AGV can detect other AGVs in the network.
  - The coordinator AGV can distribute the tasks to all AGVs over multicast.
  - AGVs can directly send messages to other AGVs.
  - The communication between the AGVs is fast and reliable.
- **Navigation Criteria**
  - The AGV detects lines and is capable of following them.
  - The AGV detects markers and can determine where it is located in.

- The AGV turns in the right direction at a handover point.
- The AGV can carry out the pick-up and drop-off movements at the relevant handover point.
- The AGV reliably carries out the movement commands given by the software.

5.2 Experimental Setup

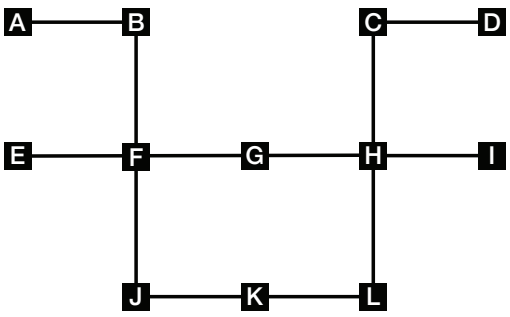
In order to evaluate the above criteria, an experimental setup had to be created. This required creating an environment in which the AGVs could operate, as well as preparing the AGVs themselves to operate in this environment.

5.2.1 Environment

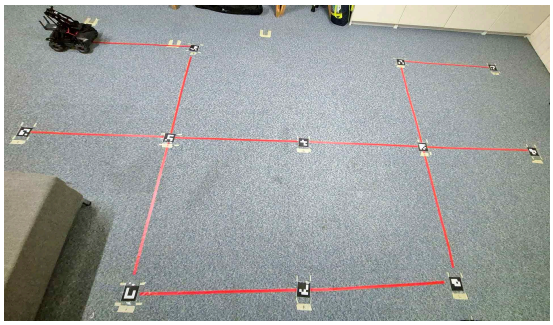
To set up the experiment, a physical environment has to be created, in which the robots can operate in. The first step of creating the environment is to design the network of paths and handover points.

The following points were considered in the decision-making process for creating the network:

- It has to be big enough for multiple robots to operate in.
- It has to be a connected graph so that all the AGVs can theoretically reach every handover point.
- It has to have redundant paths, so that when one AGV poses an obstacle for another AGV, it still avoids the other AGV in completing its task.
- It must be of a size that is practical to implement in the real world.



(a) Digital environment



(b) Real-world environment

The network graph was first digitally designed, to fulfill all mentioned criteria, and was then recreated in the real world. In the real-world network, red tape was used to create

the connections between the handover nodes. The high contrast of the tape compared to the underground makes it easier for the AGV to recognize the lines. ArUco markers were used to create the handover points, so that the AGV knows at which point in the network it is located. The room in which the network is located in is also well lit, this is necessary that the camera of the AGV can perceive all markers and lines correctly.

5.2.2 AGVs

As mentioned in Section 4.1, the robot used as an AGV is the DJI Robomaster EP Core. Two robots were provided for the scope of this work. One was available from the beginning to test the software during the implementation process. The second robot was available at the end of the implementation process. Both robots were delivered by DJI in individual parts and had to be assembled first. Two Raspberry Pis were provided for this purpose, which are to take over the control and communication of the AGV.

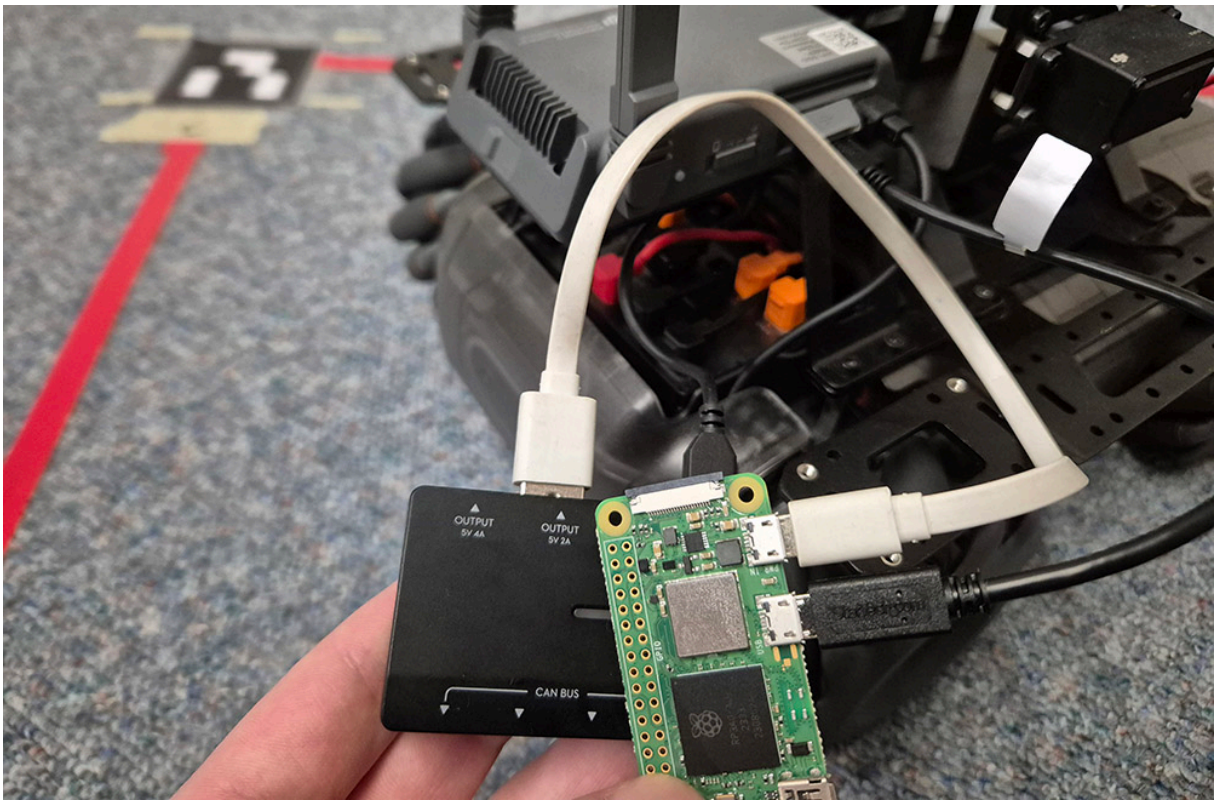


Figure 5.2: The connection of the Raspberry Pi Zero 2W to the Robomaster EP Core

The Raspberry Pi Zero 2W was connected to the robot according to the manufacturer's instructions (cf. Figure 5.2). Despite the correct connection, this led to a short circuit. Since the robots can be controlled not only via cable connections but also via commands over a local Wi-Fi network, this approach was pursued due to the short circuits. However, it turned out that it was not possible to establish a wireless connection with the second robot, which led to a change in the test scenario.

5.2.3 Test Scenarios

The prototype is evaluated using two different types of test scenarios. On the one hand, scenarios are evaluated in which the robot operates in the physically created test environment. On the other hand, scenarios are evaluated by only calculating them without physical robots actually carrying out the tasks.

5.2.3.1 Physical Test Scenarios

The goal of evaluating the physical scenarios is to evaluate the communication and navigation of the AGV. This scenario involves the cell in which the AGVs operate receiving a task from the WMS that they must perform. On the one hand, it is checked whether the communication works reliably and all AGVs receive their tasks. Furthermore, it is tested whether the AGV can maneuver successfully. In addition, times for individual operation steps are measured. Finally, a concrete scenario is executed in which the two AGVs are expected to collaborate to perform task-level collaboration.

Due to the failure described in Subsection 5.2.2, only one physical robot can be used for testing. Therefore, the second robot is simulated on a Raspberry Pi using the `MockRobot` class.

5.2.3.2 Calculated Test Scenarios

Tests with the physical prototype are very time-consuming. Since it is not feasible to examine all scenarios within the scope of this master's thesis, all possible task scenarios on the selected network of paths are calculated. If no situation can be found in the physical scenario in which multi-robot collaboration is preferred over a single-robot execution, it can be calculated whether such a situation even exists. If such a situation cannot be found, measured values of the operation duration can be reduced for the calculations. It can then be seen whether reducing these times helps to find a situation in which multi-robot collaboration is preferred over a single-robot execution.

5.3 Experiments

5.3.1 Physical Experiments

To configure the parameters used in the cost calculations, five test runs were executed by the AGV to measure the durations for each operation. The durations have to be known, since the cost calculation done by the prototype decides if a task is split into multiple subtasks and carried out by multiple AGVs or if a single robot carries out the full task.

Parameters used for cost calculations are the following:

- **Move Duration:** The time, measured in seconds, that is required by a robot to move from one handover point to the other.
- **Turn Duration:** The time, measured in seconds, an AGV needs to do a turn by 90 degrees.
- **Pickup Duration:** The time, measured in seconds, that an AGV requires to perform the necessary manipulation actions to pick up a parcel, in addition to the time needed to travel from the handover point to the pick-up location.
- **Dropoff Duration:** The time, measured in seconds, that an AGV requires to perform the necessary manipulation actions to drop off a parcel, in addition to the time needed to travel from the handover point to the dropoff location.

Duration Type [s]	**Samples** [s]					**Median** [s]
	1	**2**	**3**	**4**	**5**	
Move	9	5	6	4	5	5
Turn	4	4	4	5	4	4
Pickup	10	12	13	13	14	13
Dropoff	18	12	12	13	15	13

Table 5.1: Execution time for the different parameters used for cost calculations.

Based on the samples measured in the five runs, a median is calculated (cf. Table 5.1). These median values are then passed to the robot so that it can do more precise cost calculations to select the most suiting task distribution in further tries.

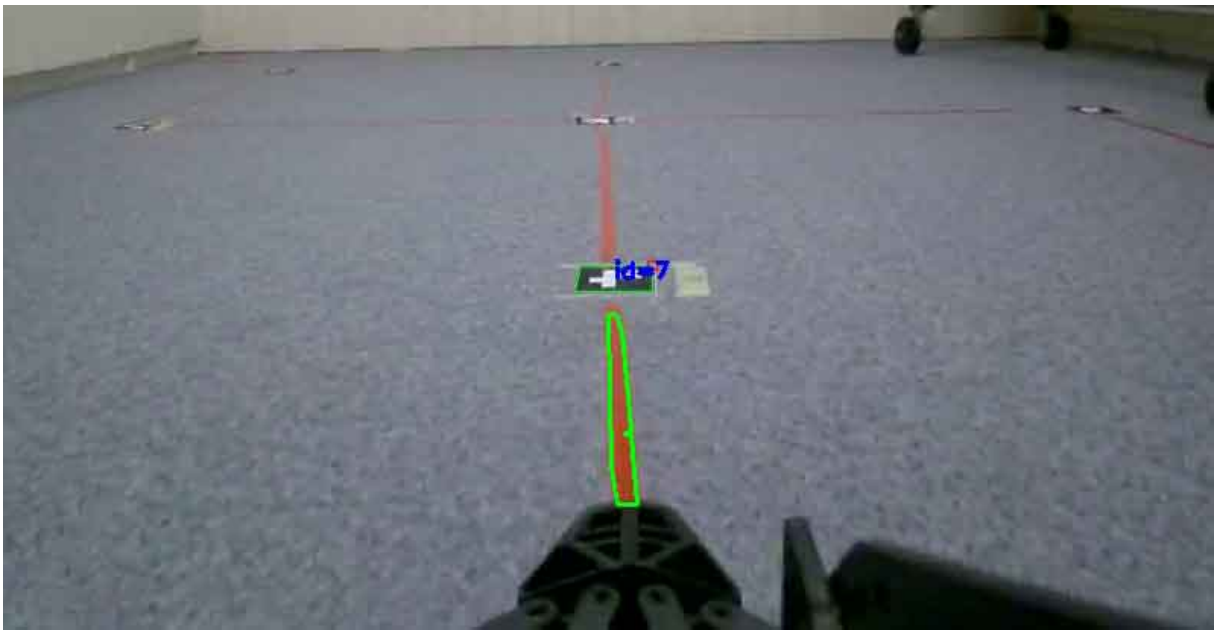


Figure 5.3: The detection of lines and markers from the AGV's point of view.

The first test runs also show that the AGV is capable of following the red lines that represent paths in the network and of recognizing the handover point markers. In Figure 5.3, the AGV's view can be seen. The green lines in the image outline the borders of what is recognized by the computer vision module. The lower green border marks the red line that the AGV follows. The upper green border marks the handover point that is perceived. The blue label indicates the value of the ArUco marker. The AGV therefore knows that the handover point G is located there.

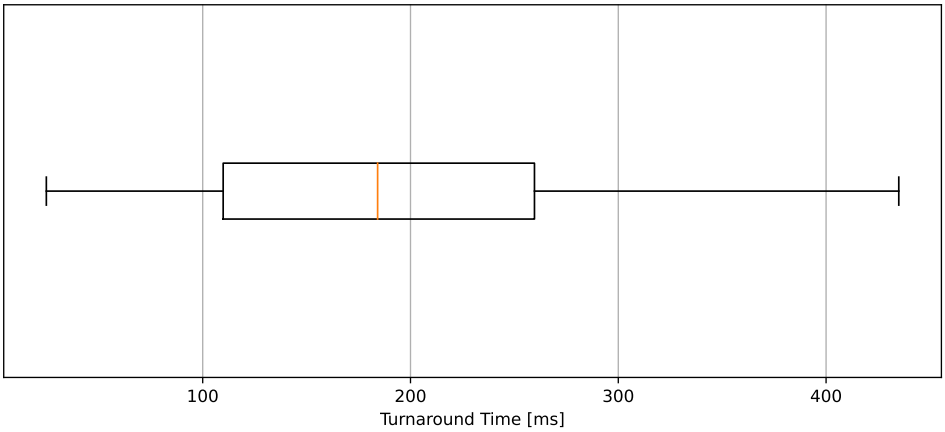


Figure 5.4: The turnaround time of a message sent by an AGV.

In addition to the navigation capabilities, the communication capabilities were also evaluated. For this purpose, 100 messages were sent from the coordinator to another AGV using the AGV's echo functionality. The turnaround times of the individual messages were measured. Results show that the median turnaround time is 184 ms, with the minimum being 25 ms and the maximum being 435 ms. The distribution of the turnaround times is shown in Figure 5.4.

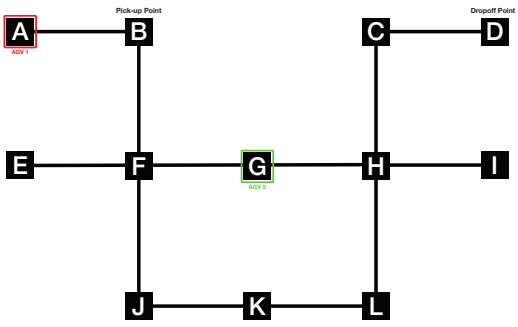


Figure 5.5: Initial experiment scenario.

After determining the duration of each operation, a physical scenario can be tested. The first hypothesis is that multi-AGV collaboration is faster than single AGV execution, in the case shown in Figure 5.5. In this situation, the freight needs to be transported from node B to node D. AGV 1 is located on node A and AGV 2 is located on node G. AGV 1 is closer to the freight to be transported than AGV 2, so it will make the first step in the

delivery and pick up the freight at node B. Since the paths are redundant, AGV 2 has two options after picking up the freight: either it takes path $B \rightarrow F \rightarrow G \rightarrow H \rightarrow C \rightarrow D$ or path $B \rightarrow F \rightarrow J \rightarrow K \rightarrow L \rightarrow H \rightarrow C \rightarrow D$. Since the former is shorter than the latter, but the path is blocked by an AGV, it is expected that the coordinator will aim for multi-AGV collaboration in this situation.

However, when the task execution of the setup shown in Figure 5.5 is executed, AGV 1 completes the whole delivery process on its own. The prototype calculates both possible delivery options and makes a cost calculation. The prototype always chooses the more cost-efficient option.

5.3.2 Calculated Experiments

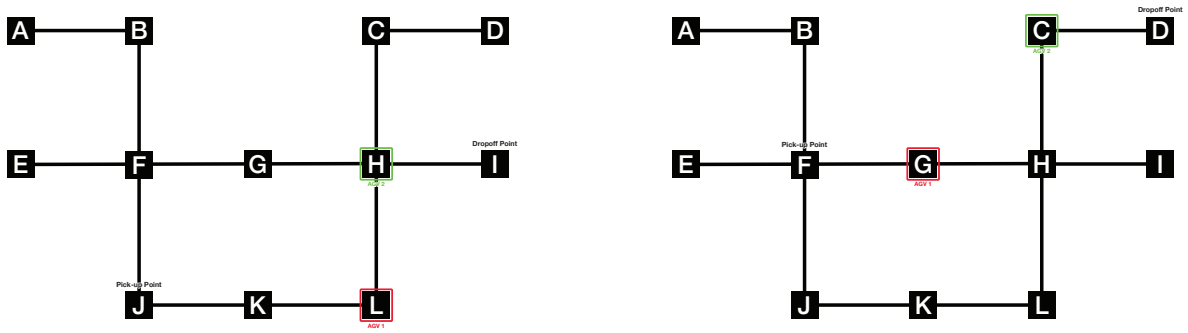
Since the evaluation of a single task setup takes up significant time, a further evaluation method is used. With the help of the median durations, all task scheduling possibilities for two AGVs involved can be calculated without need for operating the AGV.

To achieve this calculation, a script was written which initialized a `PathPlanner` object. Calculations are made with the following approach. All scenarios for tasks involving the delivery of freight between different handover points were created. For every scenario, all possibilities of AGV locations were calculated. There were three constraints to the calculations:

- The initial pickup and dropoff point cannot be the same.
- Both AGVs are not allowed to be located at the initial pickup or dropoff point.
- Both AGVs are not allowed to have the same location initially.

For every scenario, the task was split into multiple options of subtasks. Then it was checked if the path planner favors in any situation the collaboration of multiple AGVs over the task execution by a single AGV.

The calculation of 11'880 scenarios, based on the median durations shown in Table 5.1, revealed that there is not a single scenario where the algorithm favors multi-AGV task collaboration over single-AGV execution when the algorithm can choose between both. The collaboration between multiple AGVs is only pursued, when there is no alternative to it. Figure 5.6 shows two examples of when multi-AGV task collaboration is performed. The reason for the collaboration only lies in the fact that one of the AGV blocks the other from executing the task on its on, making it necessary to collaborate.



(a) Example 1 (b) Example 2

Figure 5.6: Situations where multi-AGV task collaboration is selected.

While a single-AGV execution only needs one pickup and dropoff operation, a multi-AGV task collaboration needs an additional one. Since the measured pickup and dropoff time are comparably high to other durations, further calculations were made lowering the time of those two operations until the algorithm favored multi-AGV task collaboration.

Results show, that when the pickup and drop-off durations are lowered to 6 seconds, multi-AGV task collaboration is favored, although there is also a possibility for a single-AGV execution. The multi-AGV task collaboration is selected due to a lower execution time. This scenario is shown in Figure 5.7, where the freight could be from E to I completely by the AGV 1 alone or by splitting the task with AGV 2. However, it has to be noted that if AGV would not block the path for AGV 2 to move to the node E, it would be faster if AGV 2 located at node G would complete the task in a single-AGV execution manner.

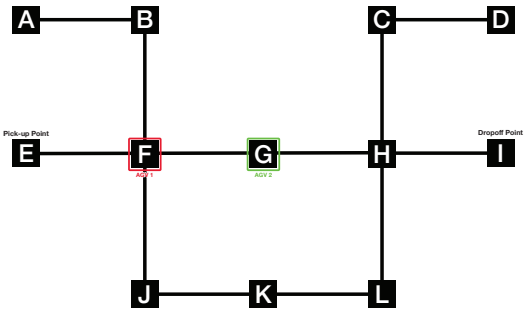


Figure 5.7: Multi-AGV task collaboration is favored over single-AGV execution, due to blocking by AGV 1.

Further calculations are run where pickup and dropoff durations were reduced to 4 seconds. Results indicated that there are new scenarios where Multi-AGV task collaboration is favored over single-AGV execution also when the initial pickup location is not blocked by another AGV. This scenario is depicted in Figure 5.8. In this case, either AGV 2 would pick up the freight at node B and deliver it on the path, not blocked by AGV 1 to node

I or AGV 2 would pick up the freight at node B, drop it off at the handover point F and AGV 2 would take over the second part of the delivery task.

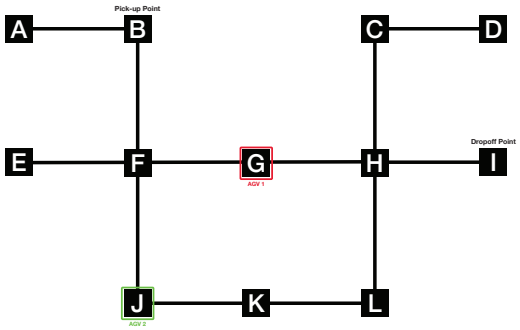


Figure 5.8: Multi-AGV task collaboration is favored over single-AGV execution.

5.3.3 Analysis of Complexity

An important question that also needs to be answered is whether the warehouse setup can be scaled. Since the path planning is done by the coordinator AGV, which in most cases does not have powerful computer hardware compared to a central scheduling instance, the resources required for path planning must not be too high. For path planning, as already described in Chapter 4, use is made of the `networkx` library to find paths in the graph. To find the optimal path from the initial pick-up to the final drop-off point, use is made of the `nx.all_simple_paths` function.

The `nx.all_simple_paths` function finds a single path using a modified depth-first search which has the complexity $O(V + E)$, where V is the number of all handover points and E is the number of all paths. However, the number of simple paths in a graph can be very large, e.g., $O(n!)$ in the complete graph of order n [27].

Furthermore, the algorithm schedules every single AGV A for every simple path P in the graph, which leads to a complexity of $O(P \times A)$. Since P is factorial in the worst case, the entire algorithm has a factorial worst-case complexity.

5.4 Discussion

In this discussion, the various results collected in this evaluation process are discussed based on the set evaluation criteria.

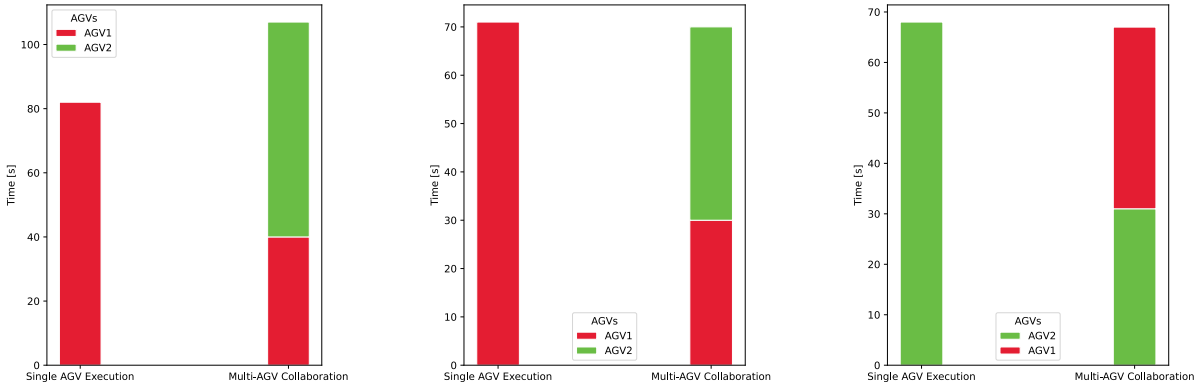
One sub-criterion of the evaluation concerns the navigation of the AGV. The criteria related to the AGV's ability to follow lines and correctly detect markers. Results showed that with the help of the camera, the AGV could usually reliably detect the red lines and markers. Sporadically, the AGV failed to detect them, but only in rare cases. This was mostly due to the markers not being flat enough on the ground, and therefore the AGV could not detect the markers. The timing of the various operations of the AGV were

measured and presented in Table 5.1. Measurements showed that the operation duration can vary depending on the trial, although all distances between handover points are the same. It is also striking that the pick-up and drop-off times are very high compared to other operations of the AGV.

Part of the evaluation criteria additionally related to the communication between the AGVs. Good communication is essential, as this is the basis for the collaboration of several AGVs. The results of the measured turnaround times of two AGVs showed that the times are in the several 100 millisecond range. Compared to the operating times for individual movements of the AGV, the turnaround time is relatively low. As mentioned in Chapter 4, the prototype uses UDP sockets to transmit messages, which contributes to a low latency and enables multicast, but limits the reliability of the messages. However, no loss of reliability was found during the evaluation.

Regarding the time complexity of the implemented scheduling algorithm, the analysis showed that the algorithm has a factorial worst-case complexity, which means that the network of paths cannot be scaled arbitrarily. However, it must be clearly understood that this is a worst-case scenario. Since the evaluation assumes that the warehouse is divided into several smaller cells, it can be assumed that the average case does not have such a high complexity.

Results of the evaluation regarding the main goal show, that in experiments, where the prototype used the real-world parameters, did not favor multi-AGV task collaboration over single-AGV execution. Pickup and dropoff durations of 13 seconds have a major impact on the calculation of the optimal division of tasks (cf. Figure 5.9a).



(a) Scenario depicted in Figure 5.5.

(b) Scenario depicted in Figure 5.7.

(c) Scenario depicted in Figure 5.8.

Figure 5.9: Comparison of task completion times between single AGV execution and multi-AGV collaboration

In the part of the experiment that is carried out by calculations, it is shown that multi-AGV collaboration is desirable when the pick-up and drop-off duration is reduced to 6 seconds, as this shortens the execution time (cf. Figure 5.9b). However, it turned out that only benefit was gained from the multi-AGV collaboration, as the better positioned AGV was blocked by the other.

However, calculations with decreased pickup and dropoff durations of 4 seconds showed that there are cases, given the network of paths used in the evaluation, where multi-AGV task collaboration is favored over single-AGV execution, due to a lower execution time (cf. Figure 5.9c). However, it has to be noted that those results are only based on calculations by the algorithm and have not been verified in a setup with physical AGVs.

The sometimes high numbers measured for individual operations on an AGV are not due to physical limitations. Rather, the numbers are given by the DJI Robomaster EP Core, which was the central element for effectively carrying out the operations. As already mentioned in Chapter 4, the scope of its SDK was very modest and in some cases very error-prone. As a result, separate computer vision classes had to be written during the implementation process, as the SDK did not offer the desired range of functions required for this project. What repeatedly led to problems with the robot was that when executing the same program with clearly defined movement sequences, the robot did not always perform the same movements and sometimes simply stopped. This necessitated that waiting times had to be added between certain instructions, which clearly increased the operation times for certain operations. Since the robot sometimes deviated from clear options, it was also not possible to test the prototype with real freight, as the robot could not be programmed to reliably pick up the freight on every attempt.

However, subsequent calculations have shown that if AGVs take less time to pickup and drop-off freight, it would be possible to achieve an increase in efficiency in transporting loads if tasks are divided into subtasks and distributed between multiple AGVs.

5.5 Limitations

Although this evaluation provides important insights into this master's thesis, it is important to consider certain limitations that may have influenced the results and their interpretation.

The warehouse environment chosen for this evaluation is fictitious. For example, colored lines for orientation of the AGV are taken from real-world warehouse concepts, but the layout of the paths was developed especially for this thesis. The performance of the AGV in other and also real warehouse environments could therefore differ and the distances between the lines also can differ. In addition, the warehouse layout was designed so that the handover points are on the network itself. This layout was chosen because the pickup and drop-off capabilities of the AGV are limited due to the robot's design, and the robot does not have to deviate from the path.

Further limitations of this project are that the prototype was implemented tailored for the DJI Robomaster EP Core and the performance of this robot is clearly decisive for the results, for example for the operation durations.

One more factor to consider is the limitation regarding the communication evaluation is that the latency was investigated in an idle network and therefore no message failure occurred. This might be different in a real warehouse, where the network used for communication might be busier.

During the evaluation process, only situations were considered in which two robots operate in one cell. This setup was chosen because two physical robots were available for the evaluation. A further constraint of this thesis is that only single delivery scenarios were considered. In a real warehouse, several delivery scenarios would probably run in parallel. While this could make the collaboration of AGVs on a task-level more efficient, it could also lead to further collision problems.

An additional limiting factor is that the evaluation could ultimately only be carried out with one physical and one mock AGV because there were connection problems with the second one via the local Wi-Fi network. At first, the connection issues did not seem very concerning to the evaluation since it was planned to use a Raspberry Pi Zero 2 W, which would transmit the commands over cable to the robot and use its Wi-Fi chip to communicate with the other AGVs. However, after connecting the Raspberry Pi to the Robot, according to the information in the documentation, and turning on the robot, the Raspberry Pi became overheated and stopped working. To connect the Raspberry Pi and the robot, a power cable must be connected to the robot on the one hand and a data transfer cable must be connected to the robot on the other. The first suspicion was that the power supply was too strong for the robot. Based on this assumption, the logical conclusion was to use a power bank as a power source, which provides the right amount of power for a Raspberry Pi. When trying this approach with a new Raspberry Pi, however, it turned out that the short circuit was probably caused by the data cable. Due to those issues, the second physical AGV could not be operated.

Chapter 6

Final Considerations

6.1 Summary

The thesis aims to analyze and implement a collaborative prototype for robots to execute warehouse tasks, improving efficiency compared to single-robot execution. This involves a literature review, designing and implementing a prototype, and evaluating its performance.

An initial literature review showed how the concept of Industry 4.0 has affected warehouses and production facilities. It also examines the context in which AGVs operate and how this type of robot differs from others. It additionally shows that in theory there are three different types of scheduling in the context of robots, namely central, semi-distributed and fully distributed. The different types of communication, implicit and explicit communication, are examined, and it is shown that both can be an advantage or disadvantage for certain specific situations. In order to gain an insight into current research on collaborative robots, various related works in the areas of AGVs and AMRs are studied. It can be seen that in these related works the robots collaborate, but do not aim for collaboration at the task-level in order to divide individual tasks into subtasks in appropriate situations.

With insights into the fundamentals and related work, a prototype design is developed based on this. In the elaboration of the scenario in which the prototype operates, the environment is defined, which should resemble a warehouse, the tasks of the WMS are explained in more detail and the functional scope of the AGV is described, as well as the freight that the AGV should transport. In addition, it is specified and shown using an example how and in which situation a task should be divided into subtasks and distributed between several AGVs. Beyond that, it is defined that the semi-distributed approach should be chosen when dividing the tasks. Furthermore, it is outlined that the AGVs should make use of an explicit communication approach to communicate with each other, and it is shown how the communication between the WMS, the coordinator AGV and the remaining AGVs in a cell works in a scenario.

After the software is designed, a prototype of the design is implemented. To begin with, the DJI Robomaster EP Core is selected as the robot that carries out the AGV's tasks.

The Raspberry Pi Zero 2 W is selected for direct control of the robot and for communication between the robots. In a further step, the range of functions of the Robomaster SDK is shown in order to gain an overview of the robot's capabilities. The software architecture is then worked out and the responsibilities of the individual modules of the software is explained. It is shown how the communication works, how the tasks are split into subtasks, how the environment is recognized with the help of computer vision and how the DJI Robomaster EP Core is controlled.

To evaluate the prototype, evaluation criteria are defined to check whether task-related collaboration between multiple AGVs speeds up the freight delivery process in a warehouse. To achieve this, a warehouse environment is physically set up and tested in a scenario with one real and one Mock-AGV and the execution times are measured. To obtain even more results, all possible task scenarios are run through. The calculated data is used to show how the execution times affect the efficiency of task sharing, and that task sharing is only worthwhile in very specific scenarios.

6.2 Conclusions

This thesis seeks to answer whether AGVs that make use of task-level collaboration can increase warehouse efficiency in splitting tasks and dividing the subtask between them to achieve a common goal. It has to be noted that the prototype could not achieve the desired goal. The time an AGV needs for operations like pick-up and drop off is too long compared to operations like turns and moves between handover points. However, it is shown that if durations of pick-ups and dropoffs can be reduced, there would be situations in which task-level collaboration could accelerate the warehouse processes.

The greatest challenge of the master's thesis posed the implementation of the prototype. It turned out that the robot that was chosen for implementation did not offer the scope that was needed in its SDK. As a result, a solution for computer vision, for example, had to be implemented, which it was assumed that the SDK already provided. The accuracy and reliability of the robot was also not as high as expected, which meant that functions such as actually lifting freight did not work. Moreover, the second robot, intended for use in the evaluation, malfunctioned due to connectivity issues. Furthermore, connecting the Raspberry Pie to the robot caused a short circuit. As a result, the evaluation setup had to be modified at short notice.

6.3 Future Work

Further future work could be developed based on this master's thesis.

Possible future work resulting from this work would be the implementation of a prototype with an alternative robot to the DJI Robomaster EP Core. This could be used to compare whether the pick-up and drop-off processes could be accelerated.

Furthermore, it could be investigated whether task-level collaboration of AGVs increases efficiency when an AGV completes several tasks in succession. Through collaboration, the AGV that takes on the initial subtask could, start taking on another subtask after completing that one.

In addition, it would be interesting to see whether it would be possible to design the network in a warehouse in such a way that it is ideal for the collaboration of AGVs at the task level.

Bibliography

[1] Radhakisan Baheti and Helen Gill. „Cyber-physical systems“. In: *The impact of control technology* 12.1 (2011), pp. 161–166.

[2] Peter Baker and Zaheed Halim. „An Exploration of Warehouse Automation Implementations: Cost, Service and Flexibility issues“. In: *Supply Chain Management: An International Journal* 12.2 (2007), pp. 129–138.

[3] E. Cardarelli et al. „Cooperative Cloud Robotics Architecture for the Coordination of Multi-AGV Systems in Industrial Warehouses“. In: *Mechatronics* 45 (2017), pp. 1–13.

[4] Gareth J. Cawood and Igor A. Gorlach. „Navigation and locomotion of a low-cost Automated Guided Cart“. In: *2015 Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech)*. 2015, pp. 83–88.

[5] Ho-Bin Choi et al. „MARL-based cooperative multi-AGV control in warehouse systems“. In: *IEEE Access* 10 (2022), pp. 100478–100488.

[6] Larissa Custodio and Ricardo Machado. „Flexible Automated Warehouse: A Literature Review and An Innovative Framework“. In: *The International Journal of Advanced Manufacturing Technology* 106 (2020), pp. 533–558.

[7] Lucas Santos Dalenogare et al. „The Expected Contribution of Industry 4.0 Technologies for Industrial Performance“. In: *International Journal of Production Economics* 204 (2018), pp. 383–394.

[8] Amandeep Dhaliwal. „The rise of automation and robotics in warehouse management“. In: *Transforming Management Using Artificial Intelligence Techniques*. 2020, pp. 63–72.

[9] DJI. *RoboMaster Development Documentation*. https://robomaster-dev.readthedocs.io/. 2021. (Visited on Aug. 16, 2024).

[10] DJI. *RoboMaster EP Core - Learning Unleashed*. https://www.dji.com/ch/robomaster-ep-core. 2020. (Visited on Aug. 16, 2024).

[11] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. *Distributed computing by oblivious mobile robots*. 2022.

[12] Raspberry Pi Foundation. *Raspberry Pi Zero 2 W*. https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/. Raspberry Pi Foundation, 2024. (Visited on Aug. 16, 2024).

[13] Jeff Geerling. *Power Consumption Benchmarks*. https://www.pidramble.com/wiki/benchmarks/power-consumption. 2024. (Visited on Aug. 16, 2024).

[14] Servus Intralogistics GmbH. *The ARC5 - optimized for your success*. 2024. URL: https://3834836.fs1.hubspotusercontent-na1.net/hubfs/3834836/ARC5%20Online%20Folder_EN.pdf (visited on Sept. 19, 2024).

[15] Peter P. Groumpos. „A Critical Historical and Scientific Overview of all Industrial Revolutions". In: *IFAC-PapersOnLine* 54.13 (2021), pp. 464–471.

[16] Abhay Kumar Grover and Muhammad Hasan Ashraf. „Leveraging autonomous mobile robots for Industry 4.0 warehouses: a multiple case study analysis". In: *The International Journal of Logistics Management* (2023).

[17] Erik Hofmann and Marco Rüsch. „Industry 4.0 and the current status as well as future prospects on logistics". In: *Computers in Industry* 89 (Aug. 2017), pp. 23–34.

[18] Elvis Hozdić. „Smart factory for industry 4.0: A review". In: *International Journal of Modern Manufacturing Technologies* 7.1 (2015), pp. 28–35.

[19] Firhan Huzaefa and Yen-Chen Liu. „Centralized Control Architecture for Cooperative Object Transportation using Multiple Omnidirectional AGVs". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 6526–6532.

[20] Intel. *Types of Robots: How Robotics Technologies Are Shaping Today's World*. https://www.intel.com/content/www/us/en/robotics/types-and-applications.html.

[21] Zool Hilmi Ismail, Nohaidda Sariff, and E Hurtado. „A survey and analysis of cooperative multi-agent robot systems: challenges and directions". In: *Applications of Mobile Robots* 5 (2018), pp. 8–14.

[22] Scott Kirsner. „Acquisition puts Amazon rivals in awkward spot". In: *The Boston Globe* (Dec. 2013).

[23] René BM de Koster. „Automated and Robotic Warehouses: Developments and Research Opportunities". In: *Logistics and Transport* 38.2 (2018), pp. 33–40.

[24] Jay Lee, Behrad Bagheri, and Hung-An Kao. „A cyber-physical systems architecture for industry 4.0-based manufacturing systems". In: *Manufacturing letters* 3 (2015), pp. 18–23.

[25] MengTing Liu, Yan Qiao, and NaiQi Wu. „Efficient Multi-AGV Real-time Collaborative Operation in Large-scale Intelligent Warehouses". In: *IEEE Transactions on Intelligent Vehicles* (2024).

[26] Xiulong Liu et al. „CPS-based smart warehouse for industry 4.0: A survey of the underlying technologies". In: *Computers* 7.1 (2018), p. 13.

[27] NetworkX Developers. *all_simple_paths*. 2024. URL: https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.simple_paths.all_simple_paths.html (visited on Sept. 23, 2024).

[28] OpenCV. *Open Computer Vision Library*. https://opencv.org/. 2024. (Visited on Sept. 10, 2024).

[29] Nikolaos Papakostas, James O'Connor, and Gerald Byrne. „Internet of things technologies in manufacturing: Application areas, challenges and outlook". In: *2016 International Conference on Information Society (i-Society)*. 2016, pp. 126–131.

[30] Gwynne Richards. *Warehouse Management: A Complete Guide to Improving Efficiency and Minimizing Costs in the Modern Warehouse*. 2017.

[31] Kees Jan Roodbergen and Iris F.A. Vis. „A survey of literature on automated storage and retrieval systems". In: *European Journal of Operational Research* 194.2 (2009), pp. 343–362.

[32] Karen Rose, Scott Eldridge, and Lyman Chapin. „The internet of things: An overview“. In: *The internet society (ISOC)* 80.15 (2015), pp. 1–53.

[33] Iris F.A. Vis. „Survey of research in the design and control of automated guided vehicle systems“. In: *European Journal of Operational Research* 170.3 (2006), pp. 677–709.

[34] Stephan Weyer et al. „Towards Industry 4.0 - Standardization as the crucial challenge for highly modular, multi-vendor production systems“. In: *IFAC-PapersOnLine* 48.3 (2015), pp. 579–584.

[35] P. R. Wurman, R. D'Andrea, and M. Mountz. „Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses“. In: *AI magazine* 29.1 (2008), pp. 9–9.

[36] Guoqing Zhang et al. „An Integrated Strategy for a Production Planning and Warehouse Layout Problem: Modeling and Solution Approaches“. In: *Omega* 68 (2017), pp. 85–94.

Abbreviations

AGC	Automated Guided Carts
AGV	Automated Guided Vehicle
ARC	Autonomous Robotic Carrier
ASR	Automated Storage And Retrieval System
AMR	Autonomous Mobile Robot
CPS	Cyber-Physical Systems
DUA	Drive Unit Agent
ERP	Enterprise Resource Planning
G2P	Goods-To-Person
ICT	Information and Communication Technology
IoT	Internet of Things
IP	Internet Protocol
ISA	(Inventory Station Agen
JIT	Just-in-Time
JIS	Just-in-Sequence
JM	Job Manager
MARS	Multi-Agent Robot Systems
MARTCO	Multi-AGV real-time collaborative operation
SBC	Single-Board Computer
SDK	Software Development Kit
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WMS	Warehouse Management System
XML	Extensible Markup Language

List of Figures

List of Tables

Listings

Appendix A

Code and Installation Guide

The code for the prototype of the AGV can be found on GitHub at the following link:

https://github.com/tobifra/multi-AGV-task-level-collaboration

To get the prototype running, the following instructions must be followed:

1. **Clone the repository**
   Clone the project from the Git repository using the following command:

   ```
   git clone https://github.com/tobifra/multi-AGV-task-level-collaboration
   ```

2. **Navigate to the project directory**
   Change into the project directory:

   ```
   cd multi-AGV-task-level-collaboration
   ```

3. **Install Python**
   Ensure that Python is installed on your machine, with a minimum version of 3.6.6 and a maximum version of 3.8.9. You can check your Python version with:

   ```
   python --version
   ```

4. **Install dependencies**
   Install all required Python packages listed in `requirements.txt`:

   ```
   pip install -r requirements.txt
   ```

5. **Navigate to the code directory**
   Change into the code directory:

```
cd code
```

6. **Set up environment variables**
   Rename the `env.example` file to `.env`:

   ```
   mv env.example .env
   ```

7. **Run the program**
   Finally, run the program using the following command:

   ```
   python main.py
   ```