Mälardalens University
School of Innovation Design and Engineering
Västerås, Sweden

Master Thesis - 30.0 Credits
Course Code: DVA502 / DVA503

DYNAMIC CONNECTION HANDLING FOR SCALABLE ROBOTIC SYSTEMS USING ROS2

Emil Persson
epn17006@student.mdh.se

Lukas Johannes Dust
ldt20001@student.mdh.se

Examiner: Cristina Seceleanu
Mälardalen University, Västerås

Supervisor(s): Mikael Ekström and Saad Mubeen
Mälardalen University, Västerås

Company Supervisor(s): Per-Lage Götvall
Volvo GTO, Göteborg

Academic Supervisor(s): Emmanuel Dean
Chalmers University of Technology, Göteborg

2022-06-09

Abstract

Multi-agent robot systems, especially for mobile robots in dynamic environments interacting with humans, have seen an increased interest over the past years. Many vehicle manufactures (e.g. Volvo GTO) have been following the trend and has started investigating a possible implementation of an autonomous-transport robot system for material delivery in production environments. First implementations of a system have been built using Robot Operating System 2 (ROS2) and initialising static amounts of participating robots. Throughout this thesis, scalability is emphasised to enhance and add new use cases to the system. This thesis investigates possible improvements for the system by adding a dynamic connection handling, which allows robots to connect and disconnect under the system's run time. Furthermore, the performance of the connection handling in the system is evaluated in simulation for increasing system complexity in terms of the amount of connected robots. The first part of the thesis presents an approach for the dynamic connection and disconnection of robots to the network using service client communication approaches. An implementation is tested in a simulation based on an excerpt from the legacy system. Furthermore, two methods are proposed for detecting possible communication losses. The thesis work simulates the increase of the number of robots in the system at different publishing rates. It compares a many to one communication approach, where multiple robots communicate to a central node over one topic, to the one to one communication approach, where multiple robots communicate over particular topics to a central node. The simulations have shown that with an increase of nodes, the average data age and the data miss ratio in the one to one approach were significantly lower than in the multi to one approach.

Keywords: *Multi-agent robot system, ROS2, GPSS system, dynamic connection handling, multi-to-one communication, one-to-one communication, scalability*

Acknowledgements

We want to extend our sincere thanks to our supervisors, Saad Mubeen and Micke Ekström, for the support and guidance during the process of creating and writing the thesis and allowing us to participate in the research work at Mälardalen University. Secondly, we would like to express our gratitude to Emmanuel Dean for guiding us through the process of creating the idea for the thesis and helping us to find and address fundamental issues in the provided system. Furthermore, we would like to thank Moris Behnam for his guidance and help to find significance for the thesis project. Last but not least, we would like to thank our families for supporting us throughout the entire process of our studies. *Thank you very much!*

Lukas Dust & Emil Persson, Västerås, May 2022

Contents

List of Figures

List of Tables

List of Abbreviation

RTOS	Real-time Operating System
ROS	Robot Operating System
ROS2	Robot Operating System 2
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
ATR	Autonomous Transport Robots
RViz	ROS visualization
GPSS	Generic Photogrammetry-based Sensor System
DDS	Data Distribution Service
QoS	Quality of Service
RCL	ROS Client Library
A0	Architecture 0
A1	Architecture 1

List of Algorithms

1. Introduction

The number of robots acting in environments populated by humans has increased significantly in recent years and is expected to grow in the foreseeable future. This increase is supported by the transition from the goal in the field of robotics, as described by Martin Ford [1]. He describes the change from creating robots with highly specialised tasks operating in a controlled and defined environment to the goal of building robots able to execute a variety of tasks while operating in a dynamic environment or even collaborating with humans. Not only the opportunity of letting a robot execute various tasks in a dynamic environment but also letting multiple robots work together to achieve more complex tasks, building multi-agent robot systems has seen an extensive research interest, as by Arai and Ota [2], [3], Mataric [4] and Rus et al. [5]. Nowadays, applications for mobile robot systems can be found in industrial production lines as described by Khalif and Jutvik[6], over assistance robots at airports described by Xu and Wang [7] to robots supporting art exhibits invented by Gomaa et al. [8]. Besides their use case-specific challenges, the main question behind the latest approaches remains: how can multiple mobile robots move and navigate in a dependable way and maintaining security in dynamic environments executing tasks and guaranteeing safety while interacting with humans. Several open issues need to be addressed on topics and disciplines such as sensor technology, actuators, and control techniques to approach the complex question. Especially the aspect of the safety and the dynamic environment are challenging parts of developing such systems. One of the main approaches to tackling the issues is to gain as much information about a created system as possible and focus on adaption techniques for unknown scenarios as researched by Bozhinoski et al. [9] as well as Yili et al. [10]. For simplicity, predictability and better control of those systems, the first approaches to multi-agent robot systems are working with a defined amount of robots, which are sufficient for most use cases. Especially in industrial and commercial use cases, it is in the interest of developers of such systems to gain scalability in the area a robot system might work in and the number of agents participating in the robot system. The goal in terms of scalability is to create the system in a way that, without complex reconfigurations, robots might be added or removed to a system, like in complex drone systems which are described by Schilling et al. [11] and Aydin et al. [12].

1.1 Motivation and Goal

Companies, like Volvo GTO have followed the trend of developing and researching the creation of a multi-agent robot system for assisting in the transport of material in manufacturing plants using ROS2. A centralised system has been created, where the robots are controlled and tracked using central computers, keeping the computing power and the intelligence for each robot limited to a minimum. The detection of robots and obstacles is done by using cameras placed in the factory where the robots should operate. Instructions for each specific movement of the robots are computed and sent from those central computing algorithms are created in the work of several master thesis [6], [13], [14] and a prototype was built for real-life testing purposes. The centralised system architecture opens possibilities for those algorithms to let the robots collaborate efficiently, as, at all times, the complete information about all robots is known by the algorithm. However, the chosen centralised processing also relies on each robot's information. This information might contain data about battery state, odometry data from the robots, and the load status. The data transmission is implemented wirelessly, where all robots are connected to the same computing device, and where all the incoming data is processed centrally. Besides a centralised system causing higher data streams, the more robots participate in the network the more processing power for this central computation unit is needed. For the implementation of the system, ROS2 was chosen, which provides a framework for the distribution and communication between the different algorithms. Regarding the communication interface between the robots and the tracking unit, a one-to-one communication architecture is chosen, where each robot publishes its data in a particular topic. When the number of robots increases, system complexity and requested network resources are increased, and a many-to-one architecture might be another possible option. In this architecture all the robots are publishing their data together on one topic. For the processing of the algorithms and the safety of humans in this system, it is vital to know the status and position of robots participating in the operation. The information about active and launched robots is provided to the system as startup parameters to simplify the system's initial development and allow for its control and simulation. Even though most of the algorithms are created in a way to handle a dynamic load of connected robots, this configuration setup for the system startup is making the system harder to scale. For example if the number of active robots is increased, the system has to be restarted with new parameters. Therefore, the concept of dynamic connection handling would help the system gain scalability during execution. Another important aspect that must be handled by the system is the disconnection of the robots, e.g. due to maintenance issues or when the robots are at a charging station. Furthermore, dynamic connection handling can help to decentralise the system into different networks, where the robots can switch between. The dynamic connection handling for mobile robots, when the robot state and malfunction detection is needed, introduces new challenges which need to be tackled.

Motivated by the described use cases and their potential applications, this thesis aims to create and implement a connection handling and create a better understanding of the two above communication architectures of the system of Autonomous Transport Robots regarding the communication between the robots and central tracking algorithms.

1.2 Problem Formulation

From the presented motivation and the goal, the problem description is derived in this section. Allowing dynamic scalability in a multi-agent-robot system is vital to identifying and describing limitations in the build system and implementing solutions that may allow scaling until a specific number of agents in a system. The selection of the centralised computation and system architecture introduces the problem of how the robots can communicate their status information to the central computer. In order to be able to track the states of multiple robots in a centralised multi-agent robotic system, it is essential to connect several robots, represented as nodes, to a single node representing the tracking algorithm. To allow those systems to have the least configuration effort, a connection and disconnection of robots should be possible in the system's run-time. Furthermore, to allow multi-robot systems to work secure and dependable in dynamic environments and allow interaction with humans, it is also essential to detect malfunction or connection losses in the communication. Therefore it is vital to create and test a concept which includes the dynamic connection handling and the detection of malfunction of robots. Another concern is the ROS2 communications network, where the possible amount of data to communicate in multi-agent-robot systems is unknown. Those identified issues led to the following research questions:

RQ1: How can nodes in a ROS2 network be connected and disconnected dynamically during run-time while detecting unexpected connection losses?

RQ2: How do the increase of connections to one node and the publishing frequency influence the performance of a ROS2 publisher-subscriber system regarding data age and update misses?

There exist several notions of "data age", e.g., data age delay in the AUTOSAR standard and other related works [15], [16]. According to the context of our work, data age refers to the time a state information of a robot needed from the creation to reach defined points in the network.

1.3 Initial Assumptions

The focus of this research work is fitted to the needs of the stakeholder's system. As the system itself is not available for measurements and tests during the duration of the thesis work, the work needs to be conducted via simulation. Therefore, the stakeholder provided a simulation framework using RVIZ and ROS2, which abstracts the existing system using limited versions of the algorithms. This simulation is used to learn about the system and orient the implementations conducted during this thesis work on the data types used in the existing system. The experimental setup used throughout the thesis is a simulation, and the computation is done on a single computer. Therefore the results may differ from an implementation in a real, physical system. The created prototypes during this thesis will serve as a proof of concept and be tested in an independent simulation. Furthermore the thesis should fit the existing system avoiding substantial changes in the communication interface, and preserving as much as possible the current architecture and its implementation.

2. Background

Even in a controlled environment considered in this thesis, several challenges need to be overcome to guarantee a working autonomous robot system, especially when multiple robots are working in the same area. Several subsystems need to be introduced and work together to tackle those challenges. This section will introduce the main parts of multi-robot systems and describe their solution for those challenges and how they are participating in making a multi-robot environment work.

2.1 Material delivery system

The current Volvo manufacturing line includes assembly kits with all materials at each workstation. Because storage space near the worker is restricted, items are stored in nearby storage racks. The more variations in a product's components, the more storage space variance. Limited storage restricts production variance capabilities and increases costs due to extending and utilising several assembly lines to meet component differences, and with a new material delivery system, Volvo wants to offer additional product variations without raising costs. The ATR (mobile robot) was developed to transfer assembly kits to the production line and allow more material diversity. The ATRs are equipped with low-cost near-range sensors guided by Generic Photogrammetry-based Sensor System (GPSS) (visual tracking) for cost efficiency. The visual tracking system uses a feed from cameras positioned on the ceiling to find robots and obstacles and transmits the data to a centralised system. The data received by the centralised system is processed by fleet control, trajectory tracking, path planning and obstacle avoidance algorithms and the results are transmitted back to the robots in the form of new instructions.

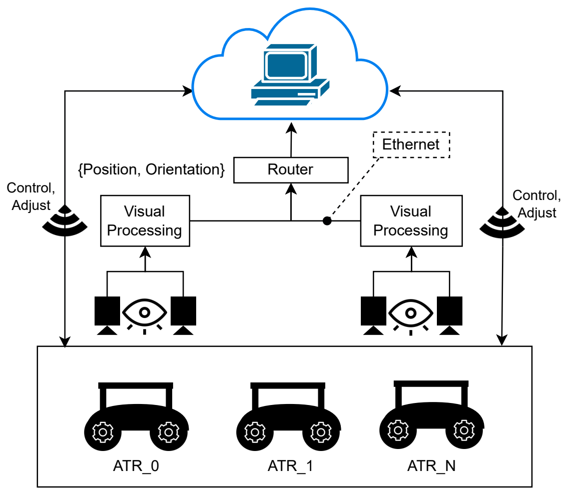


Figure 1: Centralised system architecture of the material delivery system [13]

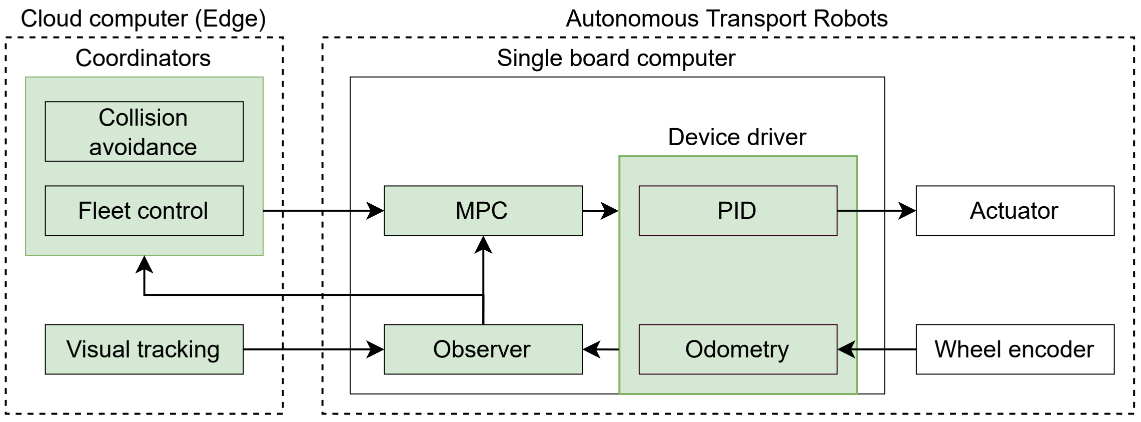


Figure 2: The architecture of the delivery system and its communication with other modules are diagrammed, the modules are divided into four modules: visual tracking, collision avoidance, fleet control and path following while green markers indicates ROS2 communication. [6]

2.2 Robot Operating System

Robot Operating System (ROS) is a middleware suite for robots [17] and rather than an operating system, ROS is composed of several frameworks enabling the creation of robot programmes with features such as hardware abstraction, low-level device control, message transfer between processes, and package management. Although ROS is not a Real-time Operating System (RTOS) [18] despite the significance of responsiveness and low latency for robot control. The absence of support for RTOS has been addressed via the development of Robot Operating System 2 (ROS2) [19]. A significant rewrite of the ROS was done to introduce industrial standard methods and functionalities while also adding support for RTOS programming and embedded hardware. In ROS, communication has three forms, topics, services and actions.

2.2.1 Publisher and subscriber via topics

Sending and receiving messages is the primary method through which ROS2 nodes communicate data. Nodes use channels to data between packages called topics that are used to transfer messages, and each topic in the ROS2 network has a unique name. If a node wants to distribute information, it must use a publisher to the same topic. A node that wishes to receive such information must use a subscriber to the same topic. Apart from its unique name, each topic contains a message type that defines the kind of message that will be sent in that topic, see figure 3. A publisher subscriber communication is asynchronous.

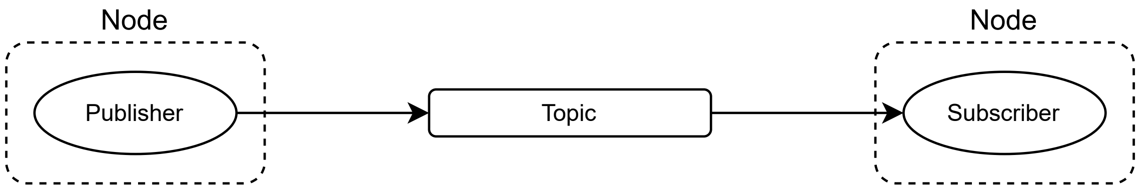


Figure 3: Communication through ROS2 topics

2.2.2 Services

Service client nodes deliver requests to service server nodes, giving back responses achieving synchronous communication. ROS services are made up of these two components: a service client and a service server, where the client, for example, sends a request to have something calculated by the server and get the response back, see figure 4.

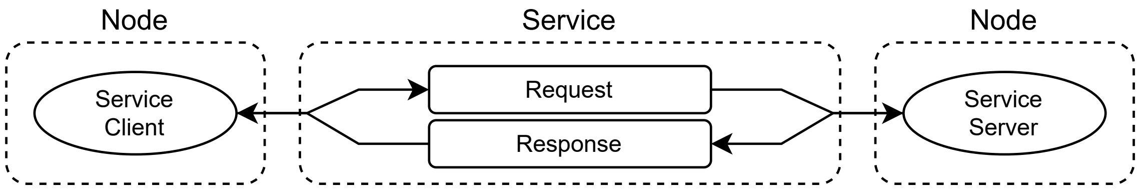


Figure 4: Illustration of ROS2 services work

2.2.3 Actions

Actions are hybrid since they provide a synchronous communication for the request, and an asynchronous communication for the continuous feedback. Actions utilise a goal to launch behaviour and send a result to create a continuous feedback system when the behaviour is complete.

2.2.4 Callbacks and executioners

There are five types of callbacks in ROS2: timer, subscription, service, client, and waitable, where timer events have precedence over communications. The callback is the minimal schedulable callback entity in ROS2 and is scheduled by the executor that controls the threading model used to process callbacks. The executor is a schedulable OS-level entity that runs on CPU cores as a thread and performs the callbacks provided, causing callback scheduling to be fundamentally different from traditional priority-based real-time task scheduling. The executor selects which callbacks are executed by polling the middleware layers for incoming messages, ROS Client Library (RCL) and other events, then invoking the corresponding callback routines until the node goes down. A wait-set, see figure 5, section 2.2.5, section 2.2.6, alerts the executor of available messages on the middleware layer, and each message is associated with a single binary flag per queue, further explained in [20], see figure 6. If the processing duration of the callbacks is less than the frequency of messages and events, the executor processes them according to the first-in, first-out (FIFO) principle. Messages and events will be queued at lower stack levels if specific callbacks need excessive processing time, which instructs the executor to process messages in a round-robin fashion rather than the previous FIFO order. By engaging with the communication middleware layer, an executor may change the ready state of non-timer callbacks in their respective queues. A change in the ready state happens when all queues are empty, and such a delayed change renders callback priority assignment useless, allowing chains to operate in a round-robin fashion. Additionally, ROS2 provides different types of executioners, such as single, multi and static single-threaded executors, depending on different applications and needs.

2.2.5 Proactor pattern

The proactor pattern describes how network systems should start, receive, demultiplex, dispatch, and handle events, whereas the proactor pattern objective is to enhance the performance of an application that receives and processes several events asynchronously. The proactor pattern entails waiting for an event before initiating the necessary procedure. Once the event begins to execute, more events may be triggered, and processed [21].

2.2.6 Reactor pattern

The reactor pattern awaits several events concurrently, and when a service request is received, it demultiplexes and delivers each request sequentially and synchronously. The reactor reduces component coupling by processing several client requests at once [22].

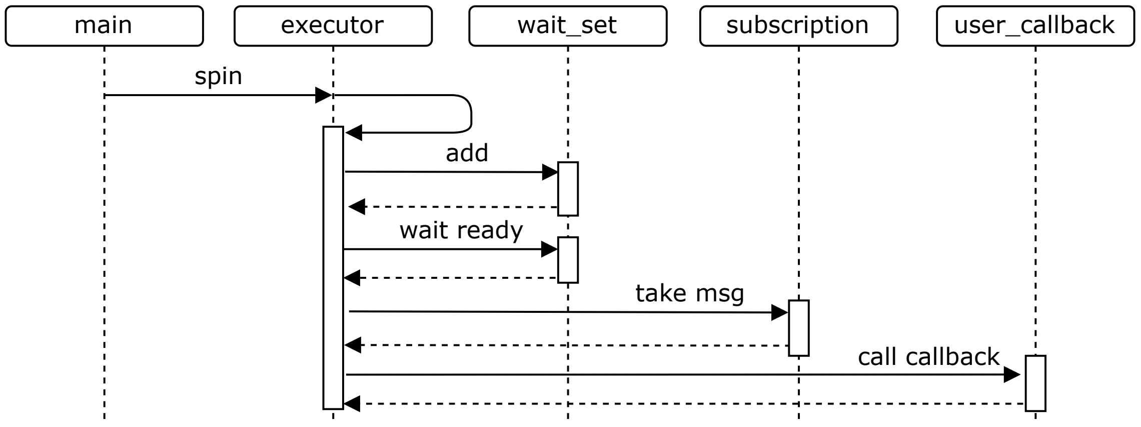


Figure 5: Depicts how the executor uses reactor pattern to provide proactor pattern when using wait-sets to add callbacks.

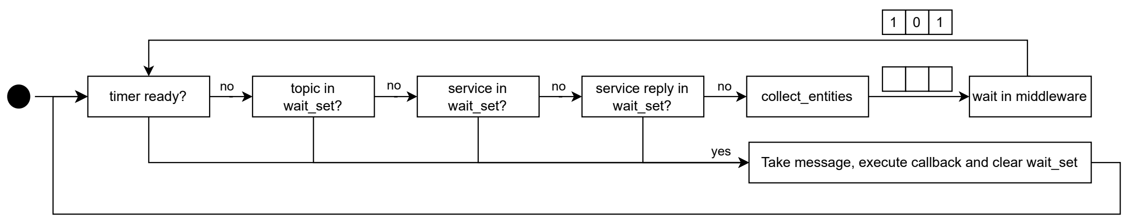


Figure 6: Executor scheduling algorithm

2.3 RViz

RViz [23] is a software application for 3D visualising robots, simulating sensors, and analyse algorithms. It allows the developer to see the robot's perception of its environment, whether real or simulated. RViz is used to visually represent a robot's state using simulated sensor data to construct an accurate representation of the robot's surroundings.

2.3.1 Coordinate frames

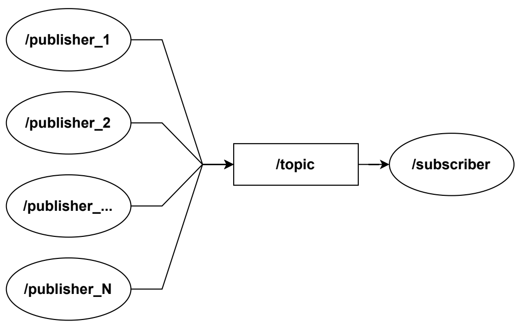
A robotic system comprises several 3D coordinate frames that vary over time, such as a head frame, a base frame, and a world frame. ROS2's TF2 transform library contains a mechanism for maintaining numerous coordinate frames over time. It preserves the associations between coordinate frames in a time-buffering tree structure and allows changing points and vectors between any two coordinate frames. Any node may utilise the TF2 libraries to listen for transformations such as the positions and orientations of all the points in RVIZ and then use the transforms to convert points across frames, providing the frames connected in the tree.

2.4 Nodes during run-time

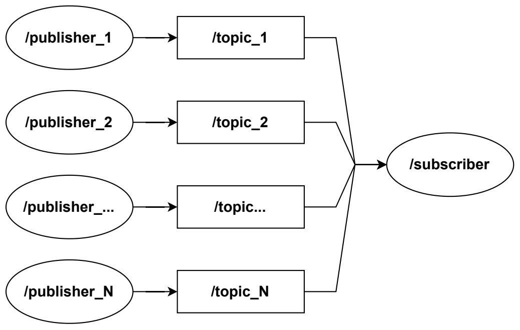
Generating new ROS2 nodes during run-time and establishing connections between subscribers is normally done using a static approach where the node, subscriber, and topic name are defined before compilation or execution. As a side effect of that static approach, the Quality of Service (QoS) and other configurations need to be defined for each node before compilation as well. The static approach is good for building a static amount of robots in a system, but other methods need to be used for scalability when attempting to establish a dynamic multi-robot system. A good practice is to reuse the same executable/package for a robot when launching multiple homogeneous robots, but nodes and topics with the same name within the same "namespace" [23, p. 22] cannot exist without communication collisions; therefore, a solution is to let each robot/node have a unique name generated during launch.

2.5 Architectures

Two different architectures of consideration during this thesis are a many-to-one, see figure 7a, and a one-to-one topic configuration, see figure 7b, as the one-to-one architecture is currently used in the material delivery system in section 2.1. But to expand on the definition of what makes a one-to-one and many-to-one architecture is, if the number of publishers that are publishing through one topic is considered a many-to-one architecture, while if each publisher has its topic, then it is considered to be a one-to-one architecture. In a many-to-one architecture, one subscriber is reading from one topic, where several nodes are publishing, while in the one-to-one architecture, the subscriber reads from several topics, where one publisher is publishing in each of them. Many different types of hybrid configurations could be used, for example, grouping publishers to specific topics, but evaluating those would exceed the possibilities of this thesis. A significant concern with the many-to-one configuration is that the subscriber cannot separate between publishers, and in the case of retransmissions due to packet loss during transmission over WiFi, the subscriber may end up in a deadlock that ends up halting the system. Having a one-to-one architecture may improve the separation between subscribers but undesired side effects, such as architecture complexity and processing overhead. However, these side effects may be resolved.



(a) Architecture Many-To-One



(b) Architecture One-to-One

Figure 7: Architectures of consideration

2.6 Real-time Operating System

A real-time system is subjected to real-time constraints [18], which indicates that the response must be assured within a specific time restriction, or the system must fulfil a set deadline. Real-time Operating System (RTOS) is implemented in contexts where a significant number of events, primarily external events, must be received and processed quickly or within a specific time period. This system is time-constrained and has a predetermined deadline, and processing must occur within the specified restrictions, or the system will fail.

Types of real-time systems that rely on timing limitations

The real-time system is of two types, soft and hard, and both have different use-cases.

Soft real-time system A soft real-time system may sometimes miss its deadline with an acceptable degree of probability. Missing a deadline has no catastrophic effects. The relevance of results generated by a soft real-time system slowly declines as tardiness increases, where tardiness measures how far behind a deadline a real-time system is in completing a given task.

Hard real-time system A hard real-time system can never miss its deadline, and the implications of missing the deadline might be catastrophic. The relevance of the results provided by a hard real-time system reduces rapidly and may become harmful as tardiness increases.

2.7 Network traffic

Network traffic is the amount of data travelling through a network at a particular moment, and network data is mainly contained in network packets, which supply the load in the network [24]. Measuring, controlling, and simulating network traffic are all based on the network of network traffic, and appropriate structuring of network traffic helps in assuring the network's QoS. The primary component of bandwidth measurement and management is network traffic, and it may be widely characterised according to its degree of bandwidth use, consumption during working hours, competitiveness for bandwidth, and slow response times.

2.8 Quality of Service

QoS is a collection of parameters that operate on a network to ensure that it can dependably execute high-priority applications and traffic when network capacity is restricted. QoS allows operators to prioritise specific high-performance applications while adjusting total network traffic. The operator may improve the performance of numerous applications on the local network and obtain insight into the network's packet loss [25], jitter [24, p. 476], throughput [26], availability [27], bit-rate [28] and transmission delay [29] by using QoS in, for example, a network of robots. The operator can design traffic on the local network and alter how packets are routed to the internet or other networks to prevent latency, ensuring that the operator meets the expected service quality for applications and achieves the desired performance. ROS2 offers a wide variety of QoS policies that allow the developer to tune communication between nodes. With the right set of QoS policies, ROS2 can be as reliable as Transmission Control Protocol (TCP) [30] or as best-effort as User Datagram Protocol (UDP) [31]. Unlike ROS, which primarily only supported TCP, ROS2 benefits from the flexibility of Data Distribution Service (DDS) transport, see section 2.10. QoS profiles can be specified for publishers, subscriptions, service servers, and clients, and applied independently for each instance. However, there is a risk of message delivery being prevented due to incompatibility.

2.9 Scheduling

Scheduling or process scheduling is a technique for allocating limited computational resources, generally processor time, bandwidth, and memory, to the multiple processes, threads, data flows, and applications required [32]. Scheduling is used to manage the system's load, ensure balanced resource allocation, and provide some prioritising based on predefined criteria. The scheduler's primary concern is how promptly it can finish a certain number of tasks (throughput), the time necessary to complete a task from the time of request or submission (latency), and the time required to complete a task or request (response time). Scheduling enables a computer system to handle all requests while maintaining a specified level of QoS.

2.10 Data Distribution Service

In distributed real-time systems, the Data Distribution Service (DDS) middleware is utilised to transmit data. DDS allows the exchange of data, events, actions, and other relevant communication between a message or data publisher and its subscribers [33]. Distributed computing systems often use this technique to ensure that data is sent and received across nodes quickly and efficiently. DDS reduces the requirement for network programming to manage communications since DDS automates the interaction of all linked nodes and applications. ROS2 supports several DDS implementations but not all vendor or implementation of DDS might not be best suited for all needs. There are several aspects to consider when selecting a middleware implementation, including license availability, and technical ones such as platform availability and computational load, neither of which will be evaluated in this thesis. The two DDS vendor of interest are the default vendor for ROS2 Foxy which is eProsima FastDDS [34] and the latest distribution's ROS2 Galactic default vendor Eclipse CycloneDDS [35]. Starting from ROS2 distribution Galactic and forward, Eclipse Cyclone DDS will be the default DDS vendor, but choosing a DDS vendor is usually as simple as changing the "RMW_IMPLEMENTATION" environment variable, if other vendors need to be evaluated.

2.11 Structure

Organising large applications on a robot system typically involves several interconnected nodes, each of which can have many parameters and dependencies. Designing the architecture needs to consider maintainability, modularity, reusability and performance. The architecture design needs to be done early on in the application phases to create a strong foundation upon which the application will be built. In ROS2, there does not exist a strict standard convention for how a project must be structured in order to function, but the ROS2 community has created a form of standard over the years to simplify collaborations and reusability but are not in any form a must-follow standard.

2.11.1 Launch hierarchy

A launch file in a ROS2 system contains all the needed information to start the execution of a ROS2 system or node. Launch files should be organised top-down to maximise reusability by grouping relevant packages and settings into separate launch files farther down the hierarchy. Grouping many launch files in the same folder allow identical robots to be swapped without modifying the launch files, and even switching from physical to simulated robots could be achieved with only minor adjustments. The hierarchy ranges from high level to low level, see figure 8, where the high-level launch files usually are the main launch file for the whole system. The main launch file contains instructions for launching everything from OS-specific instructions and simulation related packages to medium level launch files that contain instructions for launching low-level launch files. The medium level launch files usually refer to specific packages or subsystems of the application. In contrast, the low-level launch files can contain settings and instructions for hardware components such as cameras and other hardware packages/nodes, but a low-level launch file might not be needed depending on the developer's style and application.

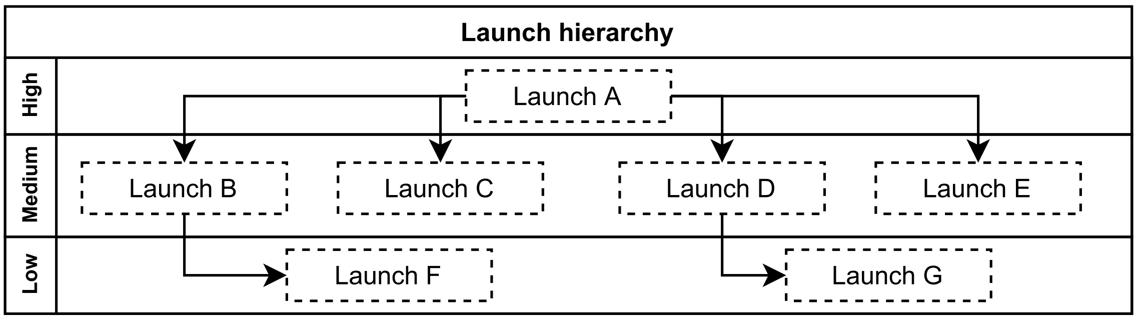


Figure 8: Depicts loosely coupled launch scripts in which are weakly associated with each other in a hierarchical structure ranging from high to low.

2.11.2 Package structure

Similar to the launch file structure, packages are organised in a top-down structure to maximise reusability by grouping relevant nodes in packages farther down in the hierarchy, ranging from high level to low level, see figure 9. The overall aim of the top-down architecture with packages is to be an independent unit and have the ability to replace or rewrite a package without breaking the whole application. Grouping the nodes into packages reduces the number of dependencies between other subsystems and avoids circular dependencies between other packages.

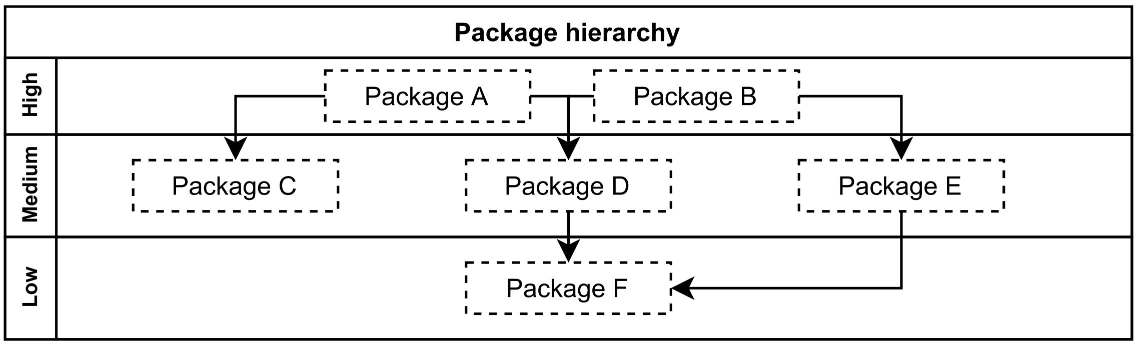


Figure 9: Depicts loosely coupled packages in which are weakly associated with each other in a hierarchical structure ranging from high to low.

A typical base structure for a robot application is to place custom message interfaces that are not included in the standard interface libraries into a separate package without any other dependencies than the standard interface libraries. These custom messages will extend the standard libraries. Having a separate library for custom message interfaces enables loosely dependent packages to reuse custom interfaces created and can be changed globally without the need to change the same interface inside each package, see "custom_msgs_pkg" figure 10. Simulating a robot will need files describing the robot mechanics, visual representation and simulation configurations for simulation tools such as RViz and Gazebo/Ignite. A common practice when simulating an application, depending on the complexity, is to create a separate package containing everything related to the simulation, both for modularity and the fact that simulation is mainly used during development, showcasing proof of concept and will not usually be shipped with the final product. The "description_pkg", see figure 10, contains the simulation related files and depending on the developer(s) of the application, it is not uncommon to find a launch directory within the package contrary to the "rule" of keeping all launch files in one directory as described in section 2.11.1. Including a launch directory within the simulation package can be viewed as proper separation, as no other package should be dependent on the simulation package, only on other packages and making it simple to remove the simulation package during deployment by making minor changes to the main launch file that calls the simulation launch file. The "bringup_pkg", see figure 10 contains all launch files and configurations for launching the application with or without exception for the simulation, see section 2.11.1 for more information. Having a modular package structure can lead to several packages being dependent on similar libraries, increasing build time as each package has to load the same library to reduce the building time of the application when the majority of packages are dependent on similar libraries, a common method is to create a meta-package that only contains a "CMakeLists.txt" and "package.xml" to load every common library only once.

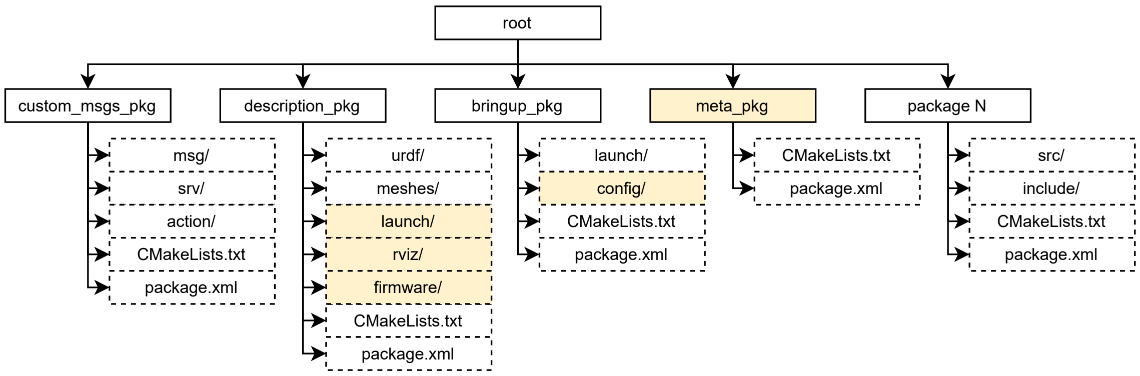


Figure 10: Common package hierarchy for larger ROS2 C++ applications, yellow indicates optional naming convention depending on application.

2.12 Temporal consistency of data

The absolute and relative consistency of the data is vital in timing-critical systems with updating data. In a system where, e.g. position data needs to be processed, this data is valid for a specific point in time. The communication delay of this data might already cause the data not to be valid anymore. Therefore absolute and relative consistency is defined as described by Audsley et al. [36]. Absolute data consistency is the condition that the absolute age of the data while processing is not allowed to be older than a specific threshold. Relative data consistency defines a set of data where all data points are to be created at a similar time frame.

3. Related Work

This thesis is divided into two main parts, where frameworks and ideas can be taken from related research work. The first goal of the thesis is to gain scalability for the multi-agent robot system using ROS2 by implementing the dynamic connection handling. As this implementation is a particular case, no use case-specific solution was provided in the current research. Nevertheless, research has been conducted about the usability of ROS2 for multi-agent robot systems, and several frameworks have been proposed. Some of which are presented in the first part of this section. The second part of the literature review will be about the investigation of the two different system architectures for communicating in a publisher-subscriber manner in a ROS2 system, including the execution of nodes as well as the communication in the underlaying DDS.

3.1 Multi-agent robot systems using ROS2

First of all, there have been several thesis works conducted in creating the stakeholder's system, and the used algorithms, such as the path following algorithm developed by Khalif and Jutvik [6] as well as an algorithm for ensuring constant distance between robots by Palsson and Svärling[13] and an algorithm for collision avoidance by Carlsson and Malmquist [14]. As the algorithms for the distributed system are already existing, the thesis aims to preserve the created system architecture and algorithms and implement improvements towards the connection handling of the robots. The usage of ROS2 in the context of multi-agent robot systems is relatively new, and there have only been a few actual physical implementations of systems. Nevertheless, the usage and performance of ROS2 in this context has seen a rising interest. First of all, there have been created frameworks, and toolboxes with architectures for the development and deployment of multi-agent systems by Dehnavi et al. [37], Testa et al. [38] as well as McCord et al. [39]. This thesis work is already based on a given simulation and a designed system with a given model of the system, so those toolboxes are not needed, but approaches especially for visualisation are very helpful. Different system architectures for multi-agent systems using ROS are proposed by Supachai et al. [40] for the augmented environment as well as by Guiseppe et al. [41] in the context of autonomous surface vehicles and Samyeul and Park [42] in a manufacturing system. Barcis et al. [43] are delivering a proof of concept for robot collaboration and interaction in ROS2. Those investigated architectures and structures are more related to the overall system design, which is already given for the system under investigation. Furthermore, none of the found papers addresses the issue of the dynamic connection handling in the context of ROS2 systems.

3.2 Analysis of ROS2 execution, communication and DDS

Since the release of ROS2, there has been quite extensive interest and research towards the analysis of ROS2 execution and performance, which might be quite relevant for the thesis. One of the tasks of this thesis is to evaluate the difference in the two possible communication architectures for connecting the robots to a tracker node using publisher-subscriber functionality in ROS2. To analyse such differences, multiple aspects need to be analysed and regarded, such as the execution of ROS2 nodes, the communication in ROS2 and the underlaying DDS. Therefore relevant work is stated in the following section. Casini et al. [44] are describing the execution model for ROS2 nodes. By analysing the ROS2 source code, the behaviour of the execution of nodes is defined and described. Furthermore, a theoretical analysis of the task execution and processing of information for ROS2 processing chains has been proposed. The paper is quite relevant for this thesis, as the described execution of ROS2 nodes helps to describe the results from the timing measurements. In difference to this thesis, the provided analysis for task execution is related to design constraints and only for a simple processing chain not containing multiple nodes communicating to a single node. Furthermore, the actual scheduling of an underlying computing system is not included, which is especially important for testing the system at the CPU utilisation limit. There has been extensive research to analyse systems regarding the stated response time. Response time analysis for architectures using a ROS network are proposed by Blaß and Casini [45] as well as Tang et al. [46]. Theoretical response time analysis for investigating and improving the existing system proposed by the cited works would be desirable. However, no holistic method is proposed to analyse a distributed multi-agent robotic system. Despite all, the stated works can be used in this thesis to gain further information about needed parameters to simulate the system and other ideas to improve the system in future work. Based on the response time analysis, Tang et al. [46] are proposing priority assignment for processing chains, which is opening the first approach to improving robotic systems regarding their end-to-end delays. Further work regarding task scheduling and priority assignment as well as synchronisation of tasks using ROS has been conducted by Choi, and Xiang [47] as well as Ding et al. [48] and Saito et al. [49]. Those proposed approaches of synchronisation and scheduling can be abstracted and used in this thesis to create concepts of improving the processing of information and controlling the system's latency, resulting in an improved response time. Zihang and Atsuhi [50] are proposing a performance analysis framework for ROS2 measuring callback execution time. Most of those stated research papers propose theoretical approaches for analysing system behaviour under simplified conditions. The stated behaviour of the ROS execution might be helpful to analyse the outcomes of the timing measurements but can, first-hand, not be used to predict the system behaviour holistically. Towards execution of task under real-time constraints in a ROS networks, research in exploring the capabilities of ROS2 has been conducted by Yang and Azumi [51] as well as Maruyama et al. [52]. Park and Raimarius [53] and Blass [54] are continuing those investigation in the real-time capabilities of ROS and performing studies in the context of multi-agent robot systems. Outcomes from those studies can be taken in this thesis and applied to the ROS network, introducing real-time capabilities and simulating their effect on the response time in the system. Furthermore, Puck et al. [55] are proposing ROS2 real-time control architecture in time-synchronized networks and investigating real-time capabilities. Distribution and synchronisation in computation for multi-agent robot systems towards real-time control of robots are addressed by the work of Puck and Keller [56]. Real-time communication inside ROS networks is offering further potential in improving the end-to-end delay of the system. Gutiéerrez [57] has researched communication for real-time robotics, where the results, especially for the real-time capabilities of communication in a ROS system, can be used to create improvements for the system in this thesis. These capabilities for real-time underline the goal of fast and reliable working systems. Our research aims not to implement real-time features, but those approaches might be used for future work to consider improvements for the execution of the communication and the connection handling. Another vital part of investigating is the communication and the DDS of ROS2 systems. Erös et al. [58] are investigating architectures for communication in ROS2 networks for control in automation systems. They are proposing different methods for many-to-one connection, but neither a performance evaluation nor the context of multi-agent robot systems is given. Ren and Katsuya [59] are investigating optimisation possibilities for inter-node communication allowing multiple DDS implementations and dynamic binding in one system. Those approaches might be interesting in the context of the thesis, but the thesis goal is to work within the current published commercial versions of available DDS implementations. Kim et al. [60] are analysing trade-offs to be made in cases where ROS2 is used with specific security standards and implementations in the DDS. Kronauer et al. [61] are researching end to end latency for distributed ROS2 systems using standard QoS settings and different DDS. Guidelines are proposed for distributed ROS2 architectures to reduce the latency overhead. In comparison to this work, in our thesis, connection handling and scalability in terms of nodes is the main focus, so the results are not directly applicable. Yuya and Shinpei [62] are exploring the potential and constraints of DDS and ROS2. Limitations regarded are the overhead of transforming data to DDS messages. Zhaolin Chen [63] is investigating network performance of ROS2 systems and investigating QoS and security constraints. Investigations are provided until a network of up to five nodes which is significantly lower than our thesis includes. Furthermore, more advanced QoS attributes are used, where this thesis wants to provide a first evaluation of performance using default settings. For future improvements, this paper might be refined in the context of this thesis created system. Several theoretical approaches for analysing the behaviour of execution and processing chains in ROS2 are provided in the conducted research. Those approaches are pretty restricted in their applicability to the case of this thesis, as simplifications and requirements are made for creating those theoretical analysis approaches. Nevertheless, the related work gives a good foundation and overview for analysing the results of the implementation and measurements in this thesis and can be applied for deriving possible improvements and areas of investigation for future work.

4. Method

This section presents the two suitable research methods: system development and platform-based design. A choice for one research method is made and discussed, followed by an introduction to the application of the chosen research method during the thesis work as well as the scientific contribution. This section ends by stating ethical considerations to be made for this thesis work.

4.1 System development research methodology

The system development research methodology described by Nunamaker and Chen [64] is a multi-dimensional as well as multi-methodical approach applicable for research in engineering and technical science. The system development research method follows the theory of system development and other complementary methods. The overall research contribution results from systems development, experimentation, observation and performance testing. These methods are needed to investigate the different aspects of the research questions. It furthermore makes this research methodology applicable in areas where research questions are hard to postulate and the direction can not be determined in the early stages of the project. The mentioned research strategies lead to the research methodology consisting and connecting the four defined and so-called research strategies shown in figure 11, where the system development is standing in the centre of this methodology. The other three research strategies are theory building, experimentation and observation.

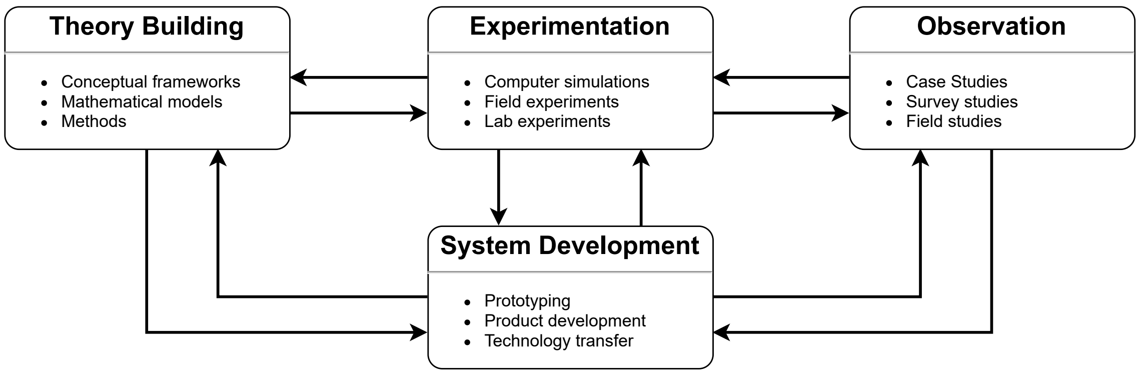


Figure 11: Overview over the process of the system development research [64]

The **Theory building** stage includes the search and brainstorming for ideas to do a theoretical construction of a conceptual framework.

The **experimentation** stage includes laboratory and field experiments together with experimental simulations. Experiments are designed based on theories, and the results may be used to improve the system and refine theories.

The **Observation** stage includes qualitative observation of the system. It is used to formulate a hypothesis that can be investigated in experiments or generalised for later investigations.

The **Systems development** is the application of the five steps of concept design, constructing the architecture, prototyping, product development and technology transfer. A more detailed description can be found in subsection 4.4.

The proposed research process is iterative, making it possible to backtrack on other directions focused on in the early stages of the project if more practical approaches for solving specific problems are viable [64].

4.2 Platform based design

Another approach to embedded systems development is the platform-based design approach described by Sagiovanni-Vincentelli, and Martin [65]. It has similarities with the system design research methodology in that both methodologies describe an iterative process for systems development. The platform-based design aims to address the satisfaction of specific constraints and the system design verification as the main issues of embedded systems development. The proposed approach consists of a meet-in-the-middle approach, where the development process is separated from the architecture implementation process. The first step in this approach is to describe the system at the highest level of abstraction without any implementation details. The iterative meet-in-the-middle approach nature is to revise the abstracted system model and the implementation. An overview over the described design flow of this methodology is presented in figure 12.

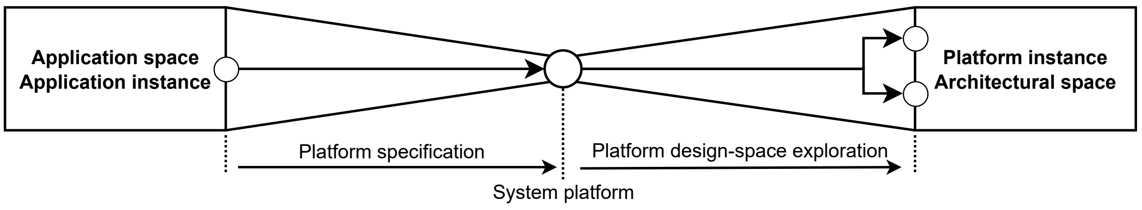


Figure 12: Overview over the process of the platform based design [65]. The figure shows the decoupling of the application development process (left triangle) and the architecture implementation process (right triangle) by the system platform (middle).

4.3 Discussion

In theory, both presented methodologies are suited with respect to the wanted developmental research to create a prototype ROS2 network and perform experiments to evaluate the performance over an increasing amount of nodes. This goal of deriving solutions based on observations and experiments of the first system versions supports the choice of the systems development research method. One difference between the two presented methodologies is in the initialisation phase. An abstracted model is already assumed in the initialisation phase of the platform-based development. In contrast, the system development research methodology follows several iterations to derive the system model. Indeed an abstracted model of the original system was created prior to the first steps of the thesis to identify and target specific bottlenecks of the existing system individually. However, this thesis's goal is to create solutions for a stated issue and mainly, the focus on the evaluation of those solutions will lead to an evolution of the final system model. This difference makes the system development research methodology more suitable for this thesis. Another significant difference between these two approaches is the focus of the methodologies. Platform-based development focuses more on the design issues and implementation of embedded systems than on the actual research and investigations, e.g. sub-components. The system implementation as a whole is making the most significant research contribution. In its multi-methodological approach, the system development research method includes investigations and observations of system components and a scientific derivation and exploration of the system. This fact also supports the selection of the System development research methodology for this thesis, as a scientific exploration of the system components focuses on possible optimisations and aims to create guidelines for possible improvements. With this, the uncertainty of the final system design and the outcome of this thesis is given.

4.4 Application of the research method

The process for the actual system development as a part of the systems development research methodology is consisting of five steps which are shown in figure 13. The scientific contribution and application as well as execution of each of the five steps in this thesis can be described following:

Construct a conceptual framework In this step, the research questions are stated, and the scientific purpose of the project is defined. The system functionalities and requirements must be investigated, and the system building process must be clarified. Furthermore, relevant literature needs to be studied to create approaches and ideas for solutions to meet the goal of this work.

Develop a system architecture This step will create a unique approach for the dynamic connection handling. Furthermore, the functionalities of sub-systems and system components are defined, as well as their interrelationships.

Analyse and design the system An appropriate system architecture is to be selected from the developed system architectures and found alternatives in the literature study.

Build (Prototype) System In this step, the actual simulatory implementation of the architecture is to be made. All the described functionalities and inter-modular relationships need to be implemented.

Observe and Evaluate system With this step, the performance of the system will be evaluated, and feedback will be given on the quality of the system. The evaluation includes proof of concept, and evaluation of the performance.

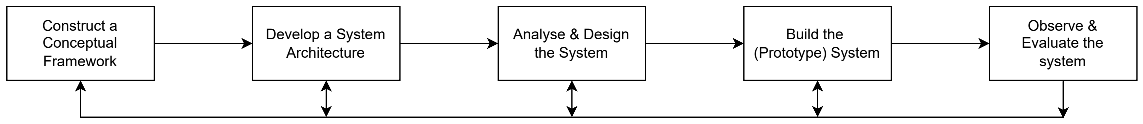

Construct a Conceptual Framework
Develop a System Architecture
Analyse & Design the System
Build the (Prototype) System
Observe & Evaluate the system

Figure 13: System development research process application [64]

In the iterative nature of the chosen research methodology, each step might be conducted more than once with the knowledge and information gained from previously conducted iterations. Applied to the case of this thesis work, the following steps are resulting from the scientific contribution, where steps 2,3,4 and 5 might be conducted iteratively:

1. System Analysis and Abstraction: This step aims to preserve privacy for the stakeholder's system and make the thesis more applicable in more general cases. The analysis gives first information about limitations and bottlenecks in the current system.
2. Framework Design: In this step, hypothesis for possible improvements will be created and refined with a literature review. This hypothesis will be used to design experiments that prove or disprove those. The experiments will be designed to be able to perform simulatory.
3. Building Simulations: The simulation platform will be set up regarding the designed experiments. The goal is to build the simulation to be used further after the end of the thesis.
4. Performing Experiments / Simulations: The designed prototype will be tested and evaluation will be performed.
5. Creating Improvements and creating guidelines: The possible improvements will be analysed and noted. Guidelines will be created to apply in the legacy system.

4.5 Ethical Considerations

In the case of this thesis, ethical considerations need to be held regarding the stakeholder data. Sensitive and private data might be used in the first step of the thesis to create an abstracted system and define the system. For the final version of the thesis, it is necessary to clarify that none of the sensitive information will appear in this thesis, and the first drafts will be shared with stakeholders for approval to counteract accidental data sharing.

5. Proposed Framework and Implementation

By applying the systems development research method, an iterative process is performed to derive and evaluate solutions for the dynamic connection handling. Each iteration is performed with a specific implementation goal and collected information for system improvements under one specific aspect. Moreover, to make the findings of this thesis applicable and related to the stakeholder's system, the evaluation will be performed based on requirements and system architectures supported by the existing system. All upcoming design choices are made to support the goal of applying the findings in the stakeholder's system. As the original system is not accessible and a whole system simulation cannot be set up during the project, a simplified version of the system is built with the most critical parts of the robot system. A system simulation is designed to investigate system behaviour and performance and implement prototypes for dynamic connection handling. Based on the goal of the thesis and the stated research questions, the implementation of this thesis consists of five iterations. Different evaluations are planned and performed by applying the chosen research method to answer the research questions and to fill the goal of deriving information for building prototype solutions for the system. The evaluations help to iteratively derive and improve the system while answering parts of the research questions. The following subsections detail the actual implementation for each of the five iterations, with motivation derived from the goal of each step. This section is divided into five subsections, where each subsection contains one of the iterations, respectively. The division in each iteration is essential as the findings and outcomes are dependent on previous iterations. The results and discussion for each iteration can be found in the related results and discussion section so that the reader can more easily follow along. The five iterations consist of:

1. **Excerpt from the ATR system:** where a simulation is created focusing on the chosen focus of this thesis important part of the ATR system.
2. **Dynamic connection handling:** where the algorithm for the dynamic connection handling is implemented and tested.
3. **Dynamic launch and ROS2 update:** where an update of the ROS2 version is done, and considerations for the launch are made.
4. **Timing measurements:** where possibilities for measurements of the created system are implemented and timing measurements performed.
5. **Absolute data consistency and malfunction detection:** where a malfunction detection based on data history and absolute data consistency is proposed.

5.1 Part 1: Excerpt from the ATR system

The first implementation work in this thesis is done with the goal of creating a simulation environment which can be used in further work for the evaluation and testing of approaches for the dynamic connection handling. Therefore the given system architecture needs to be simplified and abstracted to the problem to be investigated. The simulation should include the specific parts of the system and covers defined scenarios for different states of the robots. It is necessary to complete the implementation and simulation of all components to proceed with future iterations of the thesis. The goal of the simulation is to launch multiple robots and let them communicate their state information periodically using the two different configurable communication architectures. The tracker module should receive the state information, which collects the state information and creates an array of the robot states called a state list. RVIZ should be used to visualise the robots to visualise the system. This section begins with a discussion of the implementation tools used, followed by an overview of the system's implementation. Each simulation's components and message are described in the following sections, followed by the launch description and the visualisation.

5.1.1 Implementation Tools

All implementations during this thesis are done using the ROS2 environment running on a Linux system. The source code is written and organised using Visual Studio Code. The current version of ROS2, which is used in the existing system, is Foxy Fitzroy. All QoS settings for communication are chosen to be the default settings. For the implementation of the ROS2 nodes, the programming language C++ is chosen. The implementation of the launch script is done in the programming language Python. As a simulation environment, RVIZ, which is an inbuilt tool of ROS2 is used. For visualising nodes in a network, ROS rqt node graph is used. All the named tools are used in all implementation iterations, respectively. Project management and versioning are done with GitHub and Zenhub.

5.1.2 System overview and Structure

As described, the first simulation of the system is abstracted and reduced to the investigated problem specifically. An overview of the simulation is given in figure 14. The following listing summarises the functionality of the main part of the under investigation system.

1. Robots: The robots represent the ATRs. Each robot will be launched with a unique id and periodically sends its status information like position and orientation to the tracker. More information can be found in subsection 5.1.4.
2. ATR Tracker: The ATR tracker collects the status information of all nodes and appends this information to an array. More information can be found in subsection 5.1.5
3. Visualisation: The visualisation is for showing the robots in a simulated environment. Therefore, the state list must be transformed into marker arrays and published in global and local coordinate systems. More information can be found in subsection 5.1.6.

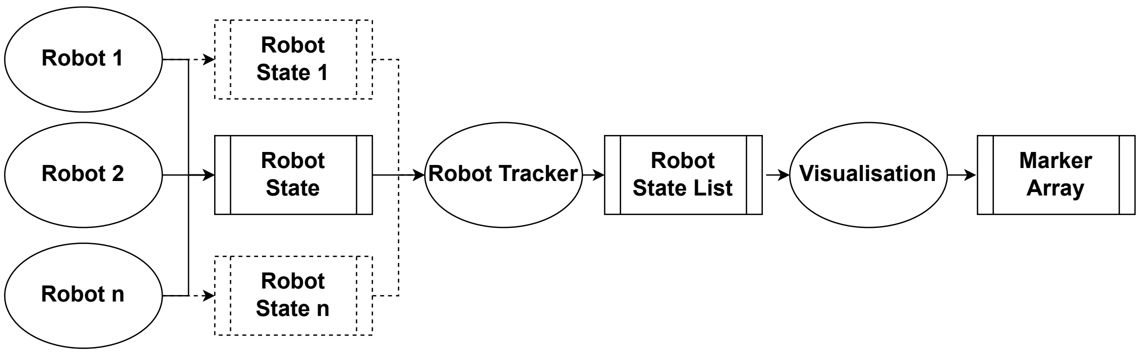


Figure 14: System overview of the main simulation containing the robots, two configurable communication architectures, ATR tracker and visualisation node (Architecture 0 (A0) full and Architecture 1 (A1) dashed line).

Marked by the rectangular boxes in the figure 14 are the messages and message types sent in the simulation. The following three message types are part of the simulation:

1. Robot State containing information about a robot. A detailed description can be found in section 5.1.3.
2. Robot State list containing a list of information from several robots.
3. Marker Array standard datatype and an array of markers for visualisation.

Two distinct system topologies are described to transmit robot state messages; either each robot has its unique topic, or all robots publish on the same topic and is represented in the overview by the dashed boxes figure 14. The creation of the ROS system is done by following the package structure described in the section 2.11.2. The following ROS2 packages are created:

1. atr_bot: containing the source code for the code running on each robot.
2. atr_bringup: containing the system's launch script and configuration files.

3. atr_interfaces: containing all message types used in the system. More information about the message types can be found in subsection 5.1.3.
4. atr_tracker: containing the source code for the tracker node.
5. atr_visualisation: containing the source code for the visualisation node and the configuration files for the RVIZ simulation.

This chosen modularity is helping for future distribution of the software, as specific packages and nodes can be compiled as a group or individually and then deployed on different computing devices.

5.1.3 Communication and Message Types

This section will describe the network implemented communication architectures and message types before detailing the exact functionality and algorithms for each network's participating nodes. All communication in this created simulation follows the publisher-subscriber approach described in the background section. In the system overview in figure 14 there are presented three different kinds of messages which need to be created and sent in the network. The first message type is the robot state message, which contains information about the state of an individual robot. The message type is designed and used in the original ATR system and integrated into the created simulation in this thesis. The state message contains the following information:

1. ATR id: individual id of the robot sending the message.
2. Pose: continuous and discrete pose of the robot
3. Velocity: linear and angular velocity data of the robot
4. acceleration: linear and angular acceleration data of the robot
5. pose source: of the types odometry, optometry and fused
6. Pose goal: The pose which the robot should reach in the goal
7. overall status: status of the robot, e.g. On Mission or blocked
8. mission status: Mission status, e.g. arrived
9. load status: e.g. loading, unloading
10. AtrStateSignals: Signals for a led signalling system
11. actuator: Information about the robots actuator
12. emergency stop: Indicates engaged emergency stops
13. battery status: charging status of the battery
14. collision status: detected collisions of the robot

The robot state list message type is an array of the described robot state message. The marker array is a standard message type and contains information for visualising a robot in RVIZ.

5.1.4 Robot Node

The robot node source code is written to represent each robot in the network. While creating the robot source code, the goal is to make it configurable and reusable. Multiple robots can be created with the same generated executable and the launch file using parameters. In the simulation, the robots will have the function to send their status information once they are initialised. The execution of the robot is oriented on the behaviour of the ATR robots in the legacy system. Parts of the source code for the execution of the robots can directly be taken from the provided simulation from Volvo. However, they will be simplified to static data. There is no further benefit in letting the robots move in this scenario. The execution of these nodes in this version has two main functions. Firstly, the initialisation at the startup of the node as described in algorithm 1 and secondly, the periodic timer callback function, which will send out the status information for each robot as described in algorithm 2. Each robot must publish its coordinate system about the world coordinate system in the tf data frame for visualisation purposes. During the initialisation of the robot nodes, each robot will initialise its position and store information about its unique id and the publisher period. A publisher will be created from the launch parameters given topic name using the robot state message type. If the auto_connect parameter is set, a timer will be created using the parameter publisher period, referring with the callback to the periodic callback function.

Algorithm 1: Initialisation of the ATR

```
Input: Id, Topic_Name, auto_connect, xpos, ypos, zpos, publisher_period
Store Input parameters
Create Publisher ($robot\_state\_message\_type$,$Topic\_Name$)
if $auto\_connect = TRUE$ then
    Initialise Timer($publisher\_period$,$timer\_callback\_function$)
end
```

The timer callback function is called each time the timer is triggered. The task of the timer callback function is to send the state of the robot by the use of the created publisher. Therefore the state message needs to be created and published with all the needed information. Each robot will send static data in this implementation, where the id increases its position in the x coordinate. Simulating all robots standing in a line helps visualise an increase of robots. Furthermore, a goal for each robot is added as being one step in the y-direction away from the robot. For the visualisation in RVIZ, it is also vital that each robot updates its coordinate system in the world coordinate system into the tf frame.

Algorithm 2: Timer callback function for periodical publishing of the robot state information

```
Input: Publisher_handler, xpos, ypos, zpos
xpos = xpos+id
goal_ypos = ypos+1
Create Message (xpos, ypos, zpos, goal)
Publish Message(Publisher_handler, Message)
```

5.1.5 ATR Tracker Node

The main task of the ATR tracker node is to collect the state information of each robot and create an array of the states and publish this array periodically. Therefore the tracker needs information about the number of connected robots and their ids. As in this first implementation, robots will have consecutive ids. The only information needed is the start id of the robots and the number of robots. As the designed system can use two communication architectures, the tracker needs information about the selected architecture to connect to one or individual topics for each robot. The tracker consists of three main functions:

1. initialisation: shown in algorithm 3 is initialising the publisher and needed subscribers as well as timers.
2. subscriber callback: shown in algorithm 4 is receiving the state information of the robots and adds them to the state list.
3. timer callback: shown in algorithm 5 is publishing the state list of the connected robots.

During the initialisation of the tracker shown in algorithm 3, the subscribers for the state publishers of the connected robots will be created and needs to be done regarding the chosen architecture. When the architecture wanted is the one-to-one, each robot publishes its topic. The topic names are created by adding the id at the end of the main topic name. In this case, the ATR tracker needs to create a subscriber for each robot. In the case of many-to-one communication, all robots are publishing on the same topic. Therefore the tracker only needs to initialise one subscriber. A subscriber in ROS2 can trigger a callback function, which will be called when new data arrives. This subscriber callback function is described in algorithm 4.

Algorithm 3: Initialisation of the tracker node

```
Input: num_of_atr, architecture, publisher_period, publisher_name,
       subscriber_base_name
Create Publisher(state_list_message_type, publisher_name)
if architecture = "one_to_one" then
    while id<num_of_atr do
        Create Subscriber(subscriber_base_name+id)
        Subscriber_Vector append(Subscriber, subscriber_callback)
        id++
    end
end
if architecture = "many_to_one" then
    Create Subscriber(subscriber_base_name, subscriber_callback)
end
Initialise Timer(publisher_period,timer_callback_function)
```

The task of the subscriber callback in algorithm 4 is to collect the state of the incoming message and update the status inside the status array. The callback will be executed every time a message is incoming from the calling publisher. The timer callback function in algorithm 5 is called from

Algorithm 4: Subscriber callback of the tracker node to collect received data

```
Input: message
get robot id (message)
take timestamp
store timestamp in reception_time_array(id)
store robot state in state_list(id)
```

the periodical timer and is publishing the collected states of the robot periodically.

Algorithm 5: Timer callback of the tracker node to publish the state list periodically

```
Data: State_List
take timestamp and store in msg header
while id<num_of_atr do
    publish atr goal in tf
    append atr state to msg id++
end
publisher_handler publish (msg)
```

5.1.6 Visualisation Node and visualisation

The visualisation node was previously established in the provided simulation of the system. It was afterwards integrated into the newly created simulation. Therefore the functionality will shortly be explained in this section, but further explanations are irrelevant. Visualisation is used to show robots in an environment. For the visualisation, RVIZ is used as an inbuilt ROS2 tool. The visualisation node was created to translate the robot state information to the needed parameters for visualisation in RVIZ.

5.1.7 Launch and execution

A launch package is created to launch and execute the nodes, where all the configuration and launch files are organised. The launch file is implemented in python. During the implementation, modularity is given in that manner. One main launch script is created, which executes launch scripts for the visualisation, tracker and robot launch individually. The parameters to pass to the nodes and visualisation are stored in an according yaml file. The launch procedure is presented in the algorithm 6. Each of the robot has its execution of the nodes inside the source code. This helps in future work to assign special executioners to the nodes individually.

Algorithm 6: Launch procedure of the simulation

```
Input: config.yaml
load launch description for simulation
load rqt graph
load tracker launch
while $idx < amount\_of\_robots$ do
    launch robot(config.yaml)
    idx++
end
```

5.2 Part 2: Dynamic connection handling

After implementing and testing the main excerpt of the ATR system, the dynamic connection handling can be implemented. The main simulation can be used and extended to the dynamic connection handling. This part of the thesis aims to establish and implement dynamic handling for connecting and disconnecting and connection tracking of robots to the tracker under run-time with the two different communication architectures. Furthermore, the simulation framework needs to be extended to provide possibilities to test the implementation and show the prototype of the algorithm working. The design and implementation of the dynamic connection handling will be explained in this section, together with implementing a simulation framework to test and evaluate the dynamic connection handling. The section will explain the design of the dynamic connection handling approach and then explain the designed algorithms inside the robot and tracker nodes. Furthermore, an approach for testing the connection handling will be presented, and the implementation explained. The section finishes with an explanation of the implemented tests for the dynamic connection handling.

5.2.1 Overview

For the dynamic connection handling, ROS2 services and client are used to signal the wake-up from the robots. The chosen connection approach follows three main steps:

1. The robot is launched and requires to connect to the network. The client inside the robot is calling the connection service in the tracker.
2. The service inside the tracker creates the required publishers and executes the required id handling. After the creation, an acknowledgement is sent back to the robots.
3. The robot receives the acknowledgement and creates the publisher and timer for publishing the state information periodically.

The same procedure is used for disconnection of the robots, but with the difference that the according publisher's subscribers and timers are deleted. The implementation of the connection handling is described more in detail in the following subsections. An overview of the implemented simulation framework can be found in figure 15. The overview shows that the services are added for the dynamic connection handling and services for controlling the robots for simulation and testing purposes.

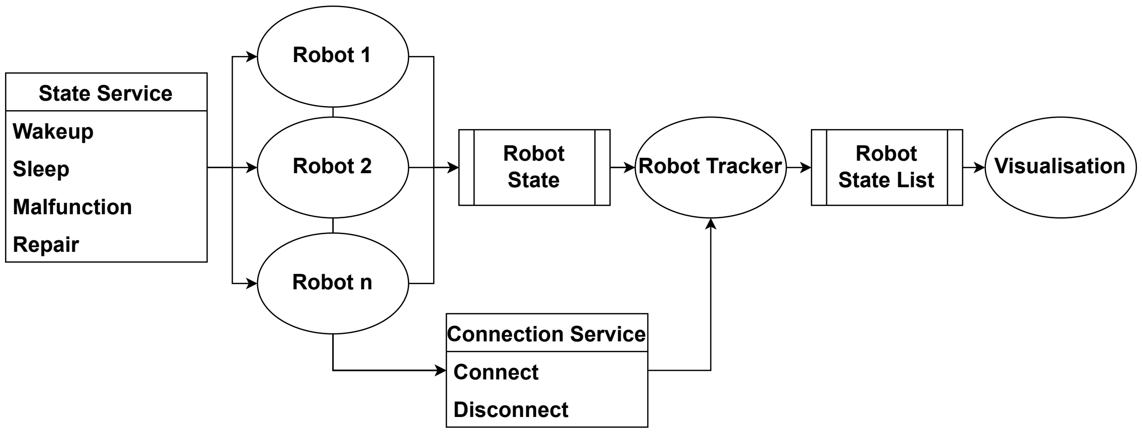


Figure 15: Framework for the connection handling

5.2.2 Connection handling from the tracker

The task of the connection handling is to enable new connections and disconnections of robots to the tracker under the system's run-time efficiently. It is essential to refer to the tracker's task in the whole system to define the needed actions and design an efficient and scalable implementation. The tracker's three main tasks are receiving information about the robot's states, creating an array of all available states, and tracking which robots are connected. Scalability in that context means that no fixed amount of maximum connected robots is assigned, and all the handling will be handled according to the number of robots. That implies that the tracker's implementation must allow robots to join the network at any moment, so the maximum number of robots cannot be fixed. C++ supports this implementation using vectors, and dynamic arrays, where elements can be added and removed under run time, leaving the only limitation of the maximum available amount of memory. The concept of using vectors and dynamic arrays is used to implement the connection handling in the tracker. The explanation of the implemented dynamic handling is separated into the connection and disconnection of subscribers, followed by the modification and tracking of the state list.

The first part of an incoming connection request is to establish a connection to the specific topic of the robot. Therefore it is vital to regard the chosen communication architecture. In a system with a many-to-one communication, where all robots publish the state message on the same topic, the implementation of the subscriber handler is relatively simple. A static subscriber to the main topic can be initialised when no robot is connected to the network and a member requests a connection. The subscribers need to do no more actions when further members join the network, as the tracker is already connected to the topic. Regarding the disconnect, if the last member leaves the network, the subscriber handler can be deleted to release the allocated memory and remove the processing time for scanning the topic. For this communication architecture, the only needed tracking for the subscription is, therefore, the number of robots. The handling of the one-to-one communication architecture is more complex. Each incoming request requires the creation of a subscriber since each robot publishes its state data on a specific topic. In the implementation, the names of the topics are derived from the robot's identifiers, making it easier to identify the topic and manage the variables. The easiest way to handle those connections is to initialise a vector of connection handlers. The corresponding subscriber will be created for each incoming request, and the handler will be appended to the vector. In the event of a disconnect request, the correct subscriber in the vector to be deleted must be determined. As the robot's id is not equal to the number of elements in the vector, active tracking must be implemented during the generation of the vector. A mapping of the handlers and id is made possible by creating a variable holding the highest id of the connected robot. With the concept of a lookup table, a new vector must be created that contains information for mapping the id to the correct element in the vector. The element in place of the robot's id corresponds to the element in the list array. Suppose robots with lower ids are not connected to the network. In that case, the vector needs to be filled with negative indicators, showing that the according id is not represented in the network. In the event of a disconnection request, the robot's id can now be mapped to the handler's position based on the handler vector. After removing the handler from the vector, all subsequent values in the lookup table for the handler vector must be lowered by one. For this reason, a third vector needs to be created, which follows the logic of the connection handler vector (appending the incoming value at a connection request) but holding the information of the id of the robots. Therefore, this vector includes the information about which order the robots are connected to the network.

Besides establishing and removing connections under run time, the main function of the tracker module needs to be executed. The periodic receipt and transmission of state information from all connected robots are arranged as a list. The implementation of vectors for establishing the connection and implementation of the state list can be done similarly. The state list will be created by appending the robot's state that wants to connect to the network at the end of the list. Suppose a new message from the same robot is incoming. In that case, the value in the vector can be overwritten using the ids from the lookup table. When a robot disconnects, the element belonging to this robot can be removed to release resources. As a connection request of the robot is in the network to be seen as a wake-up, the state information of the robot will already be sent at the connection or disconnection request, which then can be used to initialise the first or store the last robot's position at the time of connection or disconnection. Another addition for releasing computing resources is the removal of the timer calling the periodic state list publisher when the last robot disconnects. The concept for the implementation of the connection handler can be found in algorithm 7. In addition, a debug function is added to the tracker module, where on-demand the vectors can be printed.

5.2.3 Connection request from the robots

The connection request from the robot's point consists of sending out a request to the right service and waiting for the acknowledgement. Once the acknowledgement is received, the robot can create a publisher and send out its information periodically by initialising a timer. The procedure of the connection request from the robot can be found in algorithm 8. Implementing the dynamic connection handling is opening new opportunities for the initialisation and start of the system. Instead of establishing the connection from each robot in the initialisation, the connection service can be called if required. Without the dynamic connection handling, the robot checked for an auto-connect and then created a publisher and initialised the timer. Instead, the connection service needs to be called during the initialisation. The important thing is to add the root node to an executioner, which spins until the node gets the acknowledgement from the tracker. If not, the node would miss its response to the service, and system crashes are caused by the reliability settings of the service in ROS2. At the same time, in a launch script, the tracker needs to be launched first to respond to the connection requests of robots launched at the initialisation. By calling the service with different parameters, a connection or disconnect can be generated as an action. In the event of a disconnection, the periodic publishing timer and publisher handler can be deleted upon receipt of an acknowledgement.

Algorithm 7: connection handling service

```
Input: request
Data: max_connected_id, connection_map, reception_time_array, state_list,
      participants, id_array , subscriber_list
get robot id (request) get request id (request) get robot state (request)
if request_id = connect then
    if robot not connected then
        if robot_id > max_connected_id then
            cnt = max_connected_id while cnt < robot_id do
                append connection_map with -1
                cnt++
            end
            connection_map(id) = robot_id
        end
        else
            connection_map(id) = robot_id
        end
        take timestamp
        append timestamp to reception_time_array
        append robot state to state_list
        if system architecture = "many_to_one" and participants = -1 then
            create subscription
        end
        else if system architecture = "one_to_one" then
            create subscriber(robot id)
            append subscriber to subscriber_list
        end
        if participants = -1 then
            create a timer for publishing the state list
        end
        participants++
        append robot id to id_array
    end
    response acknowledge = 1
end
if request_id = 1 then
    if robot connected then
        if system architecture = "many_to_one" and participants = 0 then
            delete subscription
        end
        else if system architecture = "one_to_one" then
            delete subscriber(robot id) from subscriber_list
        end
        if participants = 0 then
            delete timer for publishing state list
        end
        remove robot id from connection_map
        remove robot from state_list
        remove robot from reception_time_array
        participants–
    end
    response acknowledge = -1
end
if debug = True then
    print max_connected_id, participants, id_array, subscriber_list, connection_map
end
```

Algorithm 8: connection request from the robots

```
Input: action_id
create state
if $action\_id = connect$ then
    call connection client (connect, state)
    wait for response
    if $response\ acknowledge = positive$ then
        create publisher
        create timer for periodic publishing
    end
end
else if $action\_id = disconnect$ then
    call connection client (disconnect, state)
    wait for response
    if $response\ acknowledge = negative$ then
        delete publisher
        delete timer for periodic publishing
    end
end
```

5.2.4 Test and Simulation functionalities

To utilise dynamic connection handling at run-time and to simulate the full functionality of the robots, four distinct actions are implemented in each robot's simulation service. Service is developed within the robot's code to target each robot individually for testing and simulation purposes. The service can be invoked to execute the specific functionality based on the input parameter. The following functionalities are implemented in the robot simulation service:

1. Wake up: simulates a wake up of the robot, where the state data is created, the connection client is called, and the publisher and timer are created.
2. Sleep: simulates a disconnection of the robot, where the disconnection client is called, and the publisher and the timer are deleted.
3. Malfunction: simulates a malfunction of the robot in that case. The timer for periodical publishing is deleted, and the publisher is deleted. The tracker still has the robot registered as active without calling the disconnect service. An error is occurring in the system.
4. Repair: simulated a repair action for the malfunction, where the timer and publisher are initialised.

The functionality of the simulation service is shown in algorithm 9. Under the system's run-time, the services can be used to target each robot individually for launching, adding and removing robots.

5.3 Part 3: Dynamic launch and ROS2 Update

After a successful implementation and test of the dynamic connection handling, the next question is how to improve the results seen in the evaluation of the previous step. At some iterations of the tests, the system crashed due to an error message presented in table 6. One of the reasons was estimated to be that the robots launched too close together in time. They influence the scheduling and execution of the connection handling from the operating systems side and the tracker crashes. Another hint found in forums[1] was that the latest release of ROS2, called galactic, and another DDS implementation called CycloneDDS have solved some problems and are more robust against the seen run time error. In this section, firstly, the problem of the launch behaviour will be solved by adding timers to the launch script, and then the update of the ROS2 version will be tested and change the DDS implementation.

[1]https://app.bountysource.com/issues/92392736-exception-in-handling-service-response-if-client-is-not-available

Algorithm 9: Service for simulating the four different actions of a robot

```
Input: request
if request = connect then
    call connection client(connect)
end
else if request = disconnect then
    call connection client(disconnect)
end
else if request = malfunction then
    delete timer for periodic publishing
    delete publisher
end
else if request = repair then
    create a timer for periodic publishing
    delete publisher
end
```

5.3.1 Dynamic launch

The recent launch of the robots is written so that all robots are released for connection requests simultaneously. In order to limit system load, timers can be introduced to the launch script so that the robots are launched without causing each other to halt. The implementation of the nodes itself is not changed in this step. The implementation is chosen to provide the opportunity to launch robots in groups while adding a variable delay between each group, which can be set via parameters.

5.3.2 ROS2 and DDS update

In a second step, the ROS2 version is updated. The previous running version of ROS2 in the system was Foxy Fitzroy. Galactic will replace this version. The update itself is not changing the implementation of the nodes themselves. The update means that the packages for ROS2 need to be replaced in the underlying operating system. Furthermore, the DDS is updated, where the previous used DDS implementation FastRTPS is replaced with CycloneDDS.

5.4 Part 4: Timing Measurements

The implementation of the dynamic connection framework in the context of the ATR system has in the previous steps shown success. It can be used in a further step to evaluate the two different communication architectures. Therefore, the framework added two more aspects: timestamp recording and a measurement node for conducting and logging the measurements.

5.4.1 Framework

The framework will be added the option to take and store timestamps at different positions. The figure 16 shows the overview of the created ROS2 network as well as the position of the timestamp. A further explanation can be found in the following subsections.

5.4.2 Timestamp recording and storage

The most important timestamps of the system, which can be seen in figure 16 are defined as the following:

T1 Timestamp one is taken when the robots create their status information and publish those in the state message. The timestamp will create a state message in each robot and can be used to give the age of the data.

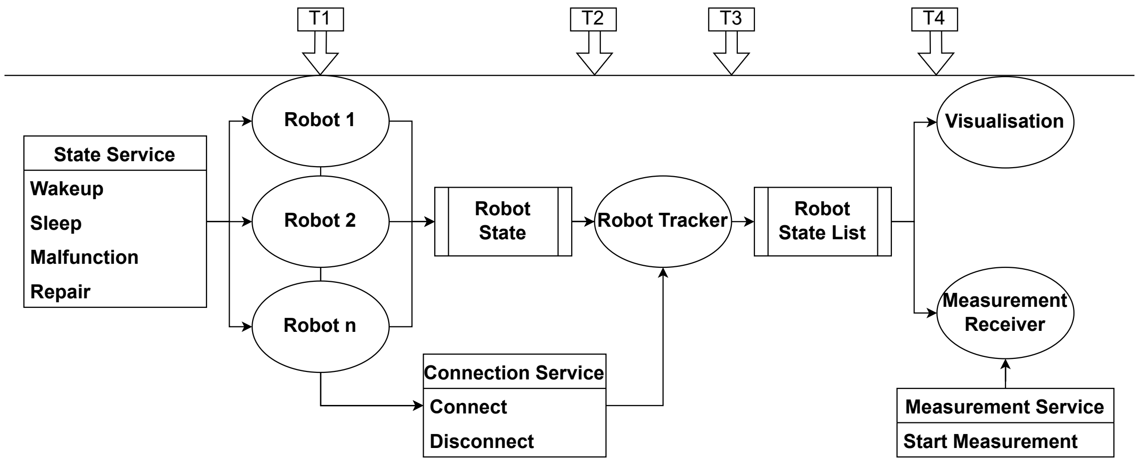


Figure 16: Framework for timestamps and measurements in the simulation

T2 Timestamp two is taken when the tracker node receives the state from each robot. Each robot's state messages are sent individually. This timestamp will be individual for each robot's state.

T3 Timestamp three is taken when the tracker node publishes the state list. This information is equivalent to the point in time of the periodical execution of the tracker state list creation and transfer.

T4 This timestamp is taken when the state list message is received at the network's end.

Messages will include timestamps to maintain the ability to trace the information about the timestamps when a piece of information reaches specific points in the network. From the implementation in 5.1 it could be seen that the state message from each robot already contains the information about the creation time (T1). The tracker implementation also shows that the time for receiving the message is already taken but just stored locally. A timestamp for each array element can be appended for both the creation and reception of the subsequent message. Furthermore, this list will contain the timestamp T3 in the header information. The last timestamp of the reception will be recorded at the measurement node specified in the following section, which also prints the timestamps for measurement purposes.

5.4.3 Measurement node

The measurement node is created and added to the network to collect the timestamp information and provide the opportunity to start and stop measurements and print the resulting timestamps to a log file. The node needs to connect to the state list messages and provide the opportunity to start a measurement. A service that can be called with the input of the number of state list messages to collect and print out is responsible. The concept of the implementation of the measurement node is presented in the algorithm 10.

5.5 Part 5: Absolute data consistency and malfunction detection

The last part of the thesis is about the connection and the detection of connection losses or malfunction and the data consistency. In order to detect malfunction or connection losses, the system must know which robots should be connected and which ones are disconnected. That information is provided through the implemented connection procedure. A malfunction in the system can be detected when the previous data was not updated. The same or older data is provided from the system in two or more iterations, or the messages were not received. Therefore absolute data consistency can be used to check the age of all available data inside the tracker. A checkup can be implemented in the timer callback for the periodical publishing of the data. In order to allow a different period, this implementation provides an extra timer which will trigger an

Algorithm 10: Measurement node

```
Input: Log_filename, measurement_length
Data: measurement
Function Initialisation:
    create subscriber to state list topic (topic callback)
Function Measurement Service(measurement_length):
    store measurement_length
    set measurement = TRUE
Function topic callback (msg):
    if measurement = TRUE then
        open Log file (Log_filename)
        read timestamps(msg) store timestamps in Log file
        cnt++
        if cnt = measurement_length then
            measurement = FALSE
            cnt = 0
        end
    end
```

extra callback. In this callback, the array holding the state information from each robot with the creation timestamps will be scanned and the age calculated. Suppose the age exceeds a particular threshold (which is set to be the publishing period of the robots). In that case, a warning is indicated to show that new data might be missing. Other options are possible besides the absolute data consistency for detecting possible malfunctions. A buffer could be implemented, storing the history of the sent state list inside the tracker. Each robot's data can be compared to the previous iterations to see if the data has been updated or not.

6. Evaluation and Results

This section is giving an overview about the results achieved by conducting the tests and experiment designed in the section 5.

6.1 Part 1: Excerpt from the ATR system

The first part of the implementation is about rebuilding the excerpt from the ATR system, which contains all crucial nodes and participants of the network, to simulate the two different system architectures and to evaluate the performance. In the first step, the system functionality should be represented in the current status but with additions which make further evaluation and improvement possible. Some tests need to be designed, implemented, and executed to ensure the correct implementation of the first simulation framework. These tests are presented in the following subsection, followed by the results from their execution.

6.1.1 Implementation test and evaluation design

The first simulation is built as a reduced version of the legacy system to the main root of the problem under investigation. The robot system will be launched with different parameters, and the correct execution will be checked with the inbuilt ros tools to show the first implementation working. ROS2 allows echoing all published messages and the frequency of publication inside a topic. Furthermore, the ROS2 rqt ros node graph shows all active nodes and topics. The simulation in RVIZ will visualise all the launched robots standing in a line. The system will be launched with a different number of robots and periods for publishing data from the robots and the tracker to test the implementation. Furthermore, the two architectures will be launched and tested. Even though the existing simulation does not cover the dynamic connection handling yet, a first stress test will be performed, where the number of connected robots will be maximised until malfunctions occur to see the upcoming errors. This information might help investigate future limitations of the simulation of a multi-robot system on a single computer. The designed test and evaluation for this iteration consists of 4 steps which are

1. Launch 10 robots with many-to-one communication Architecture 0 (A0)
2. Launch 10 robots with one-to-one communication Architecture 1 (A1)
3. Generate a system crash by launching too many robots with A0 communication architecture
4. Generate a system crash by launching too many robots with A1 communication architecture

The goal of the first two tests is to ensure the correct working wise of the implementation by measuring the communication frequencies and triggering of the robots and checking the correctness of the data by using the visualisation in RVIZ. The goal of steps 3 and 4 is to find out the current limitations of the simulation in the first step. The following table 1 shows the used hardware for simulating the system.

Table 1: Hardware setup for experiments part 1, 2 and 3

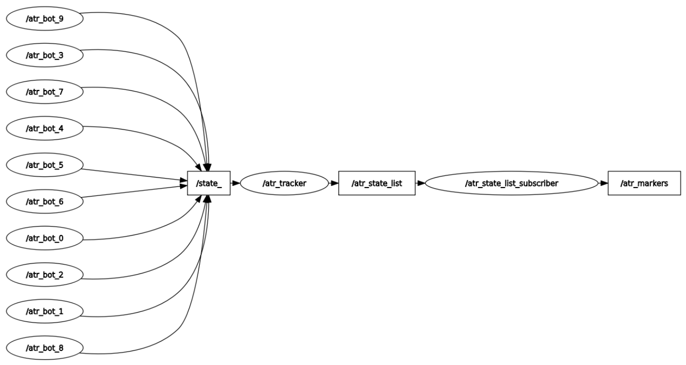
OS	Kernel	DE	CPU	GPU	RAM
Ubuntu 20.04.4 LTS x86_64	5.13.0-40-generic	GNOME	Intel Xeon E5-1660 v2 (12) @ 4.000GHz	NVIDIA Quadro K2000	1822MiB / 32033MiB

6.1.2 Results

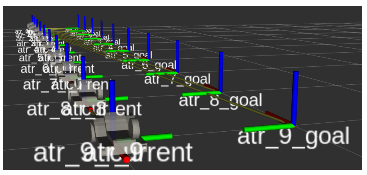
The execution of the described tests and evaluations are presented in this section. Firstly the outcome of the connection and launch will be shown, followed by the system stress test.

Architecture

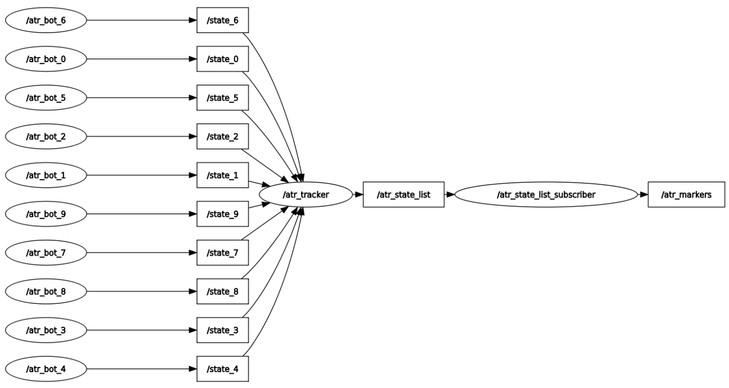
The figure 17a is showing the launch of the system with the two different architectures on the top and bottom accordingly. The upper half of the figure is visualising the output of the rqt node graph, which shows all launched nodes as ellipses and the connection with the publishers and subscribers through topics (boxes). The two different architectures can be seen with in figure 17. Figure 17a shows all the robots publishing into the same topic. Figure 17c shows all the robots publishing using individual topics for connecting to the tracker node. The lower part of the figure is showing the visualisation output of RVIZ. All launched robots can be seen to be lined up and publishing their coordinate system as well as their goal and goal coordinate system.



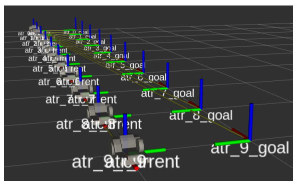
(a) Architecture Many-to-One



(b) RViz of Many-to-One architectures



(c) Architecture One-to-One



(d) RViz of One-to-One architectures

Figure 17: Architecture Many-to-One versus One-to-One

The following tables 2 and 3 are visualising the measurements conducted with the ROS2 frequency measurement option. It can be seen, that in A0 the frequency is around 100hz and the publishing of the state list is conducted with around 10 hz, which agrees with the defined parameters. Both values for A1 are around 10 hz as well.

Table 2: Architecture Many-to-One

Topic: atr_state_list	average_rate: 9.965	min: 0.093s	max: 0.107s	std_dev: 0.00301s
Topic: atr_state_	average_rate: 101.066	min: 0.001s	max: 0.058s	std_dev: 0.01571s

Table 3: Architecture One-to-One

Topic: atr_state_list	average_rate: 10.029	min: 0.097s	max: 0.102s	std_dev: 0.00165s
Topic: atr_state_N	average_rate: 10.002	min: 0.099s	max: 0.101s	std_dev: 0.00077s

System stress test

The second part of the evaluation is to cause a system crash by connecting more and more robots to the system and to see the resulting crash message. The same crash behaviour and error message was resulting for both architectures, which is shown in table 4.

Table 4: Maximum ATR error message

0	[RTPS_TRANSPORT_SHM[Error]	[Failed init_port fastrtps_port7613: open_and_lock_file failed]	[->Function [open_port_internal]
1	[RTPS_TRANSPORT_SHM[Error]	[Failed init_port fastrtps_port7613: open_and_lock_file failed]	[->Function [open_port_internal]
2	[RTPS_TRANSPORT_SHM[Error]	[Failed init_port fastrtps_port7613: open_and_lock_file failed]	[->Function [open_port_internal]

6.2 Part 2: Dynamic connection handling

The dynamic connection handling for the robots is added to the built excerpt from the ATR system. The implementation of the dynamic connection handling required a change in the approach of launching and initialising the robots. The functionalities and a right execution needs to be tested to deliver a proof of concept for the connection handling. This section will firstly present the designed implementation tests before the results are shown in the second part.

6.2.1 Implementation Test design

Different scenarios have to be tested to test the correct implementation of the connection handling. As a first step, the basic connectivity and the handling will be tested by launching the system with ten robots and connecting them in order to the system. This test will be executed for both architectures individually. In a second step, a sequence is created where robots are connected and disconnected more randomly. Firstly ten robots are launched and not connected to the tracker. In the next step, the robots are connected in random order. The third step is to disconnect all robots in another random order. After the disconnection, some robots will be connected and disconnected again. This test is repeated for both of the architecture. To be able to ensure a correct connection handling, the variables involved in the connection handling are visualised, as well as the state list shown for visualising the changing states of the robots. In the last step, the limits of the system will be tested in the sense of connected robots. The number of robots will be increased until the system is not able to connect more robots or crashes, and the error messages or warnings will be stored.

6.2.2 Results

The figure 18 and figure 19 are showing the visualisation of the ROS2 node graph and the RViz at specific points of the implementation test. In the figure 19a, 19b and 19c the launched and connected robots are shown. Furthermore it can be seen, that the connection and disconnection has been taken place. The test worked without triggering exceptions or errors, and the state list message was updating. In table 5, an excerpt from the logging of the connection handling variables

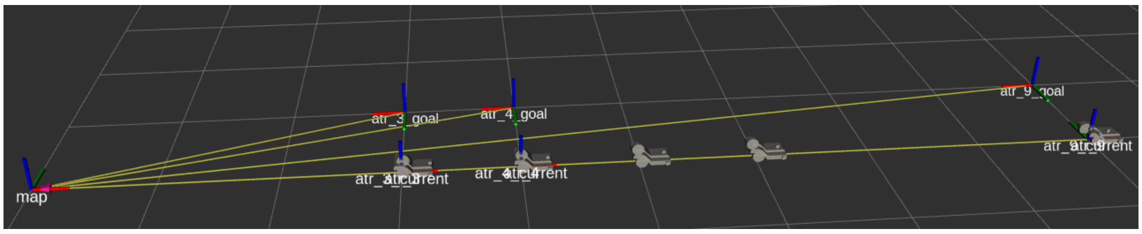
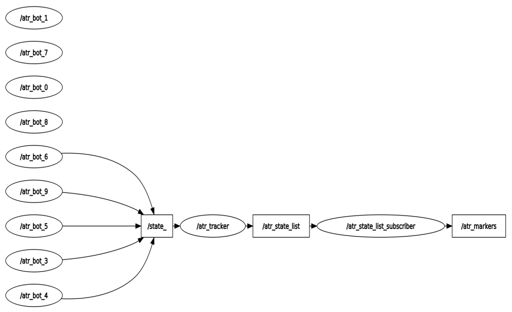
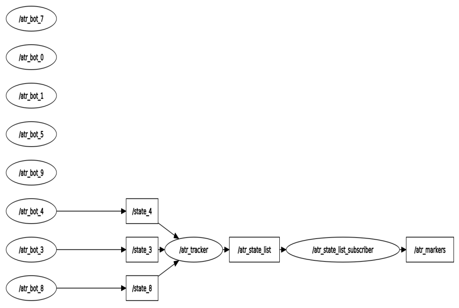


Figure 18: RViz of randomly connected ATR

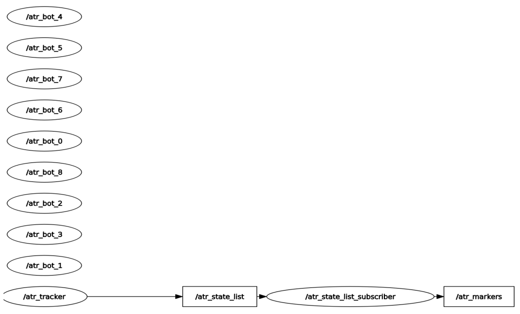
is shown. It can be seen, how robot 1 and 0 are starting and sending their connection requests to the tracker. The tracker node is processing the connection requests and connects the ATRs in the order of the incoming connection requests. The id array and map (which is called connected) are filled accordingly.



(a) Random connected ATRs using many-to-One architecture



(b) Random connected ATRs using one-to-one architecture



(c) Launched, but not connected ATR

Figure 19: ROS node graph of randomly connected and not connected robots

Table 5: Dynamic connection handling using debug

```
[INFO] [atr_bot-5]: process started with pid [27872]
[INFO] [atr_bot-6]: process started with pid [27875]
[atr_bot-6] [INFO] [1651131834.089264794] [atr_bot_1]: ATR_1 WAKEUP STARTED
[atr_bot-6] [INFO] [1651131834.094539094] [atr_bot_1]: SEND CONNECTION REQUEST
[atr_bot-5] [INFO] [1651131834.094904860] [atr_bot_0]: ATR_0 WAKEUP STARTED
[atr_tracker-4] [INFO] [1651131834.095277454] [atr_tracker]: Incoming request to connect atr_1
[atr_tracker-4] [INFO] [1651131834.095844315] [atr_tracker]: Return Connection with id 1
[atr_tracker-4] [INFO] [1651131834.095870937] [atr_tracker]: DEBUG CONNECTION HANDLER ATR_1
[atr_tracker-4] [INFO] [1651131834.095890053] [atr_tracker]: max_id: 1
[atr_tracker-4] [INFO] [1651131834.095907159] [atr_tracker]: participants: 0
[atr_tracker-4] [INFO] [1651131834.095925771] [atr_tracker]: connected: [ -1 0]
[atr_tracker-4] [INFO] [1651131834.095943243] [atr_tracker]: id_array: [ 1]
[atr_tracker-4] [INFO] [1651131834.095958909] [atr_tracker]: RETURN TO NODE
[atr_bot-6] [INFO] [1651131834.096340163] [atr_bot_1]: ACKNOWLEDGE: 1
[atr_bot-6] [INFO] [1651131834.096648920] [atr_bot_1]: START TIMER FOR PUBLISHING 1
[atr_bot-6] [INFO] [1651131834.096684671] [atr_bot_1]: TIMER CREATED 1
[atr_bot-5] [INFO] [1651131834.098660404] [atr_bot_0]: SEND CONNECTION REQUEST
[atr_bot-5] [INFO] [1651131834.099517453] [atr_bot_0]: ACKNOWLEDGE: 0
[atr_bot-5] [INFO] [1651131834.100008555] [atr_bot_0]: START TIMER FOR PUBLISHING 0
[atr_bot-5] [INFO] [1651131834.100044635] [atr_bot_0]: TIMER CREATED 0
[atr_tracker-4] [INFO] [1651131834.099140965] [atr_tracker]: Incoming request to connect atr_0
[atr_tracker-4] [INFO] [1651131834.099206390] [atr_tracker]: Return Connection with id 0
[atr_tracker-4] [INFO] [1651131834.099229724] [atr_tracker]: DEBUG CONNECTION HANDLER ATR_0
[atr_tracker-4] [INFO] [1651131834.099247550] [atr_tracker]: max_id: 1
[atr_tracker-4] [INFO] [1651131834.099265121] [atr_tracker]: participants: 1
[atr_tracker-4] [INFO] [1651131834.099288631] [atr_tracker]: connected: [ 1 0]
[atr_tracker-4] [INFO] [1651131834.099304967] [atr_tracker]: id_array: [ 1 0]
[atr_tracker-4] [INFO] [1651131834.099320148] [atr_tracker]: RETURN TO NODE
```

After the tests for the correct functionality of the connection process and handling, the test with the maximum amount of robots is executed. The results show for both architectures, that between 20 and 45 connected robots, the ATR tracker was crashing for both architectures. The reasons were the same two different errors for both architectures, which were appearing randomly and independent from each other. Excerpts from the error messages are presented in table 6.

Table 6: Error causing the tracker node to crash

[atr_tracker-4] terminate called after throwing an instance of 'rclcpp::exceptions::RCLError'
[atr_tracker-4] what(): failed to send response: client will not receive response
[ERROR] [atr_tracker-4]: process has died [pid 96684, exit code -6]
[ERROR] [atr_tracker-4]: process has died [pid 106292, exit code -11]

6.3 Part 3: Dynamic launch and ROS2 Update

The evaluation and implementation of the dynamic connection handling, was leading to the fact, that an update of ROS2 and the addition of delays to the launch might help to increase the maximum amount of robots, which could be connected to the network. The evaluation conducted with those updates are presented in this section, as well as the obtained results.

6.3.1 Evaluation

In a first step, the timers are added to the launch script. After the launch of each robot, a delay of 0.5 seconds will be added and the system tried to launch with the most robots connected to it repeated for both architectures. In a second step, the ROS2 version is updated from Foxy to galactic and the experiment repeated. In a third step, the DDS vendor is switched from FastRtps to CycloneDDS and the experiment repeated.

6.3.2 Results

After adding the delay to the launch script, a quite similar behaviour in some simulations the first error described in table 6 was seen at around 40 robots. In other simulations the system was launching and running, but at the same amount of robots, the service for connecting new robots was not unavailable, which was seen for both architectures. After the update of the ROS2 version from foxy and galactic, with the previously used fastrtps DDS, for A1 an average of 92 robots could get connected. For A0, an average of 114 robots were launched before the service was getting unavailable again, or the system crashed caused by the same error like in the previous step. With the ROS2 galactic and the cycloneDDS version, for both architectures around 180 robots were able to be connected before the service was unavailable again. The crash behaviour was no longer seen during the experiments.

6.4 Part 4: Timing Measurements

In section 5.4 the designed and implemented framework for taking measurements from the ATR system is explained. In this section the conducted evaluation will be explained and the results presented. This subsection is closing with a discussion of the results.

6.4.1 Evaluation

The implemented framework allows to take four different timestamps in the duration of the ATR system. The simulation is designed to allow different periods for the execution of the ATR tracker and the robots. To be able to investigate the behaviour, a simulation setup is created. The created simulation of the system is allowing three parameters as variables for the experiment setup. The first parameter is consisting of the architecture, which is A1 and A0. With the using of the two different architectures an evaluation of the behaviour and recommendation for an architecture is possible. The second parameter is the period for the robots to publish data and the tracker to publish the state list. To simplify the evaluation of the system, for bot timers the same period will be taken during the experiments. In the legacy system built at the stakeholders site, an update rate of the robots in the 10 ms area. As an evaluation of the system at the lower limit of possibilities is wanted, the timings are set to be 1, 2 and 5ms. The third parameter during the experiments are the amount of launched and connected robots. There the decision is made to simulate 1, 5, 10, 20, 50 and 75 robots in the system. For each of the possible combinations of parameters, the system will be launched and the measurement service in the measurement node called. The measurement is configured to take 100 samples, so store the timestamps of 100 state list updates. After the data collection, the timestamps are taken and plotted in the raw timestamps format to analyse the behaviour and find correlations and to start describing the behaviour. After this step the data will be processed to gain further information. For each measurement following values will be calculated:

1. Duration of the measurement in seconds
2. The duration between timestamp 1 and 2 which is the time the message needs from the creation in the robots until it is processed inside the tracker.
3. The duration between timestamp 2 and 3 which is the time the data is hold in the tracker before it is send out as a list.
4. The duration between timestamp 3 and 4 which is the time between the list is created and received in the measurement node.
5. The amount of update misses, which is meaning when robots are not updating their status information in a period of the tracker execution.

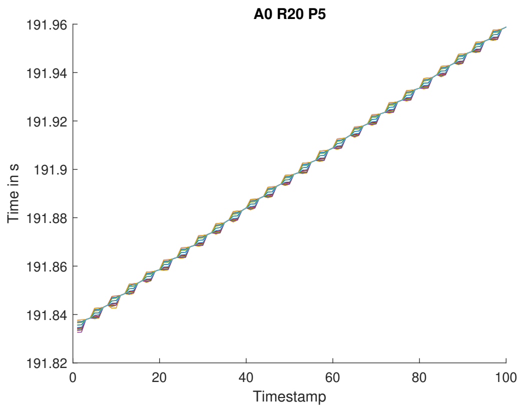
The results of the evaluation and data processing are presented in the next subsection.

6.4.2 Results

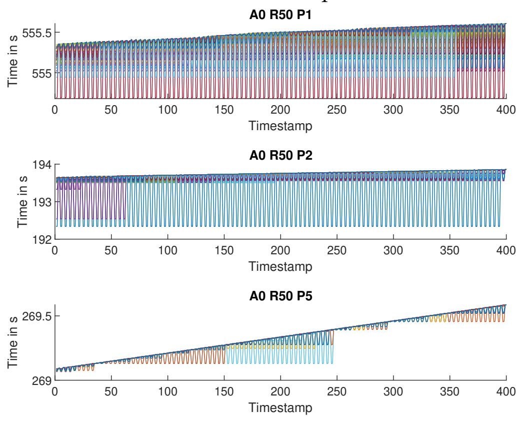
The achieved results from conducting the measurements and data processing explained in the previous subsection are presented in this subsection. After conducting the measurements, the data for each of the measurements is plotted. Figure 20a shows the data for the first 100 taken timestamps of the measurement for A0 with 20 robots and a 5 ms period, wherefrom every four timestamps are building up one round of execution, which means that each fourth timestamp is representing the same action in another iteration. On the x-axis, the number of timestamps is presented. The y axis shows the system time where the specific timestamp was created. The different coloured lines in the graph represent the different robots participating in the system. It can be seen that the robots are firstly spawned in a random order, and the messages are sent and received in the second timestamp. All incoming messages are synchronised in timestamp three, as they are collected in a list and sent out together.

The fourth timestamp represents the reception of the list. The figures 20b and 20c are showing the timestamps plotted for the A0 in subplot 20b and for A1 in subplot 20c created with launching 50 robots with the 1, 2 and 5 ms update periods in the different subplots presented below each other. In the graph, it is visible for A1 that some robots are not updating their data. That can be seen when the first two timestamps, one iteration later, are the same as the previous iteration. For a 1 ms update period, this is happening more often than for a 5 ms update period. In the plots for A0, another behaviour can be seen. The data seems to update, but the communication of the data between T1 and T2 is taking relatively long, especially for 1ms and 2 ms update rates. Furthermore, in both architectures, it is visible that with the 5ms update period, the graph is with the maximum point almost going linear up, while with a 1 ms update period, the behaviour becomes non-linear.

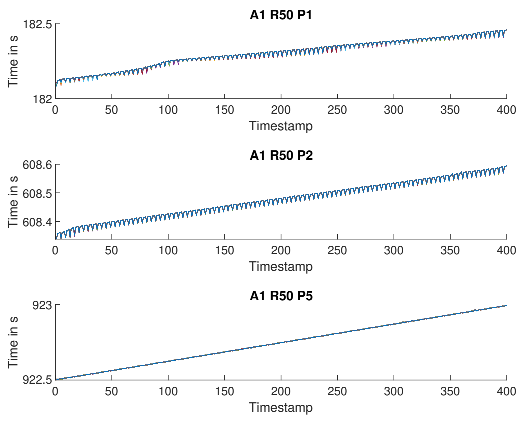
To get more information about the behaviour of the full data, the duration between the timestamps is calculated and averaged for each iteration of the experiment using the three parameters. The results of those calculations are presented in tables 7, 8, 9 and the overall data age in table 10. The data is presented for each of the three timing setting of 1, 2 and 5 ms periods in the columns. Each of the columns for the timing settings is presenting the average data age for the two possible architectures. The rows are thereby representing the amount of robots launched in the system.



(a) Plotted Timestamps for A0 with 20 robots and 5 ms period.



(b) Architecture 0



(c) Architecture 1

Figure 20: Plotted Timestamps for the two different architectures with 50 robots and over 1, 2 and 5 ms period. The different colours are representing the different robots participating in the system.

The following table 7 is presenting the average time needed from the creation of that data inside each robot (T1), until the reception of the data in the tracker node(T2).

Table 7: Duration T1T2 in ms

Timing	1ms		2ms		5ms	
Robots	A0	A1	A0	A1	A0	A1
1	0.178	0.225	0.245	0.236	0.241	0.223
5	0.176	0.144	0.183	0.181	0.219	0.225
10	0.192	0.208	0.198	0.205	0.179	0.210
20	0.368	0.965	0.306	0.301	0.185	0.201
50	10.383	23.428	1.928	19.586	0.449	0.666
75	24.206	61.956	9.450	32.523	0.616	1.762

The following table 8 is presenting the average time in ms the data spent inside the tracker node after reception (T2), until the data is send out as a list (T3).

Table 8: Duration T2T3 in ms

Timing	1ms		2ms		5ms	
Robots	A0	A1	A0	A1	A0	A1
1	0.195	0.850	0.687	0.922	3.423	3.771
5	0.438	0.375	0.917	1.108	3.022	2.880
10	0.456	0.439	0.968	0.800	2.932	3.256
20	1.074	0.285	0.827	0.857	2.428	2.850
50	151.147	0.516	85.853	0.359	7.099	2.104
75	907.164	3.225	273.382	0.777	493.034	1.616

The following table 9 is presenting the average time in ms the data needed from the sending of the list in the tracker node (T3) until the reception in the measurement node (T4).

Table 9: Duration T3T4 in ms

Timing	1ms		2ms		5ms	
Robots	A0	A1	A0	A1	A0	A1
1	0.156	0.153	0.202	0.183	0.179	0.197
5	0.436	0.266	0.282	0.282	0.334	0.314
10	0.356	0.363	0.385	0.408	0.438	0.437
20	0.708	0.832	0.787	0.748	0.829	0.992
50	2.963	2.743	2.236	2.080	1.864	2.043
75	11.374	9.703	4.047	4.318	3.413	3.325

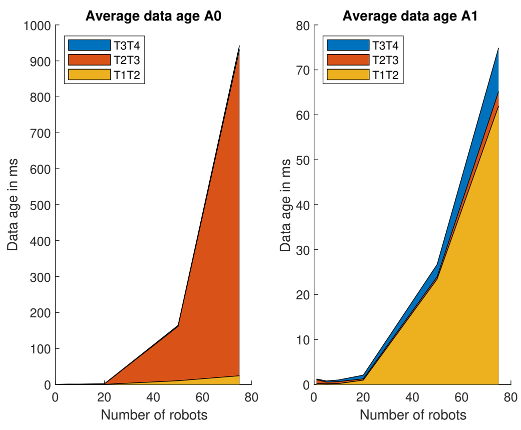
The following table 10 contains the average data age in ms after a whole iteration of communication of the data from the robots over the tracker to the measurement node.

Table 10: Duration T1T4 in ms

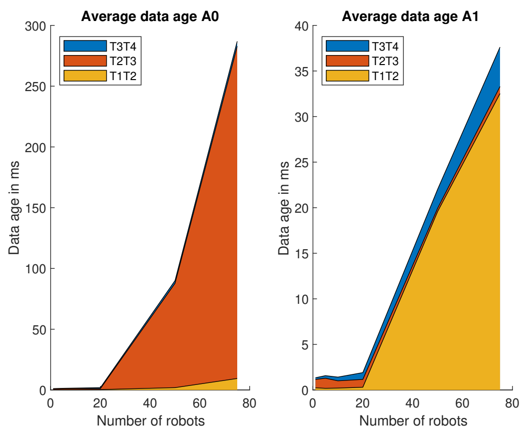
Timing	1ms		2ms		5ms	
Robots	A0	A1	A0	A1	A0	A1
1	0.529	1.228	1.134	1.341	3.843	4.191
5	1.050	0.785	1.382	1.571	3.575	3.419
10	1.004	1.011	1.551	1.413	3.549	3.903
20	2.150	2.081	1.920	1.906	3.441	4.044
50	164.494	26.688	90.017	22.025	9.413	4.812
75	942.743	74.884	286.879	37.618	497.063	6.703

For a better overview and analysis, the average data age and the proportion of the different durations are presented in figures 21a 21b and 21c for the different timing settings. Each plot is presenting therefore one architecture at a specific timing setting. On the x axis, the amount of launched robots can be found, while the y axis is describing the data ageing in ms. The duration between the sending of the data in the robots and the reception of the data inside the tracker is represented by the yellow area. The orange area is representing the time the data is spending inside the tracker node, before it is send out as a list. The blue area is representing the time which is needed to send the data as a list from the tracker node to the measurement node.

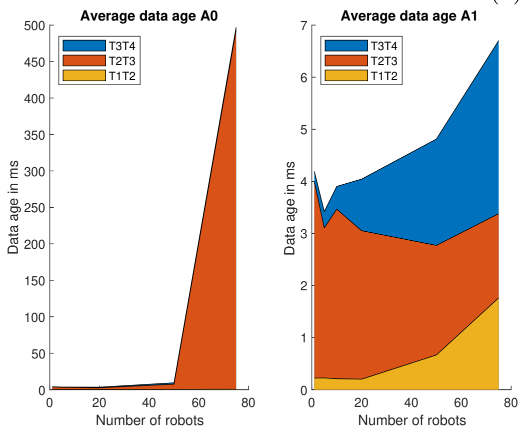
From the tables and the figures it can be seen, that after a threshold with a higher amount of robots, the average data age is increasing considerably, but much more for A0 than for A1. For the 5ms period, the average data age is starting to rise at a point of 50 robots, while the rise for 2 and 1 ms is seen around 20 robots. Furthermore, it can be seen, that in A1, the most significant part of the data ageing is happening between timestamp T2 and T3, while in A0 the most significant part of the ageing is happening between timestamps 1 and 2 and 3 and 4. This behaviour of the



(a) Timing 1ms



(b) Timing 2ms



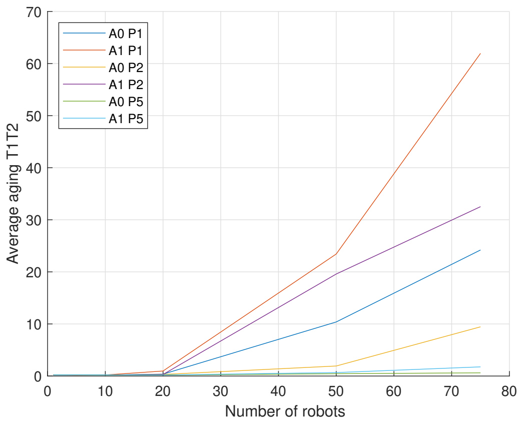
(c) Timing 5ms

Figure 21: Average data ageing in the two different architectures for the period of 1 ms, 2 ms and 5 ms. The different colours are representing the different duration between the 4 timestamps.

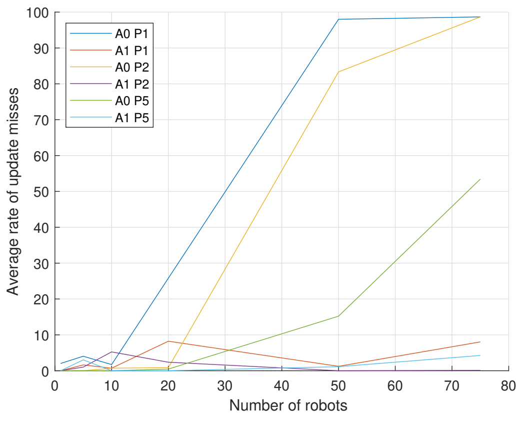
data ageing between timestamps 1 and 2 for the different parameters is also shown in figure 22a. In this figure the amount of launched robots is shown on the x axis and the average ageing of the data from T1 to T2 on the y axis. The different lines are presenting the different measurements. It can be seen, that the ageing for architecture is increasing with the amount of connected robots. For A1, the ageing is increasing more and to a higher level than for A0. Another part of the data analysis is the calculation of the update miss ratio. This value is describing how many percent of updates from robots are missed. On the x axis the amount of robots is presented, while the y axis shows the percentage of update misses. It can be seen, that for higher amount of robots the ratio is increasing, for A0. For A1 the update misses are around the same percentage. The calculated values are presented in figure 22a.

The last part of the data evaluation was the calculation of the length of the measurement. Therefore the timestamp from the first sending of the data list from the tracker is taken until the last sending of the status list from the tracker. Table 11 is showing the results. It can be seen, that for the high amount of robots with, the time is increasing for both architectures, while the time is increasing most for the 1ms period data. To the execution time to rise, a specific threshold need to be reached, which is lower for low periods. Furthermore the A1 is increasing more than the A0.

As the thesis aims to investigate the behaviour of the two communication architectures during an upscaling of a system in terms of connected robots, the most important observations regarding those aspects for each of the architectures are summarised in the following. Basically the findings can be summarised in the following way: In general, the architectures are behaving similarly until



(a) Average duration T1T2 in ms



(b) Update miss ratio

Figure 22: Average data ageing between T1T2 and update miss ratio. The different lines are representing the different architectures and timing intervals.

Table 11: Measurement duration in ms

Timing	1ms		2ms		5ms	
Robots	A0	A1	A0	A1	A0	A1
1	0.099	0.099	0.198	0.198	0.495	0.495
5	0.099	0.099	0.198	0.198	0.495	0.495
10	0.099	0.099	0.198	0.198	0.495	0.495
20	0.099	0.099	0.198	0.198	0.495	0.495
50	0.254	0.338	0.210	0.233	0.495	0.495
75	1.183	1.247	0.364	0.517	0.500	0.500

reaching a specific threshold of connected robots. The threshold is around 20 robots for 1 and 2 ms periods and around 50 robots 5 ms periods. The following findings are

1. Data ageing

   after the threshold of connected robots, the data age is rising for both architectures

   after the threshold of connected robots, the data age is rising much more for A0 than for A1

   after the threshold of connected robots, the biggest part of the ageing is happening between T2T3 for A0

   after the threshold of connected robots, the biggest part of the ageing is happening between T1T2 for A1

2. Update misses

   after the threshold of connected robots the update misses are rising for A0

   the update miss ratio is not increasing for A1

3. Measurement execution time

   after the threshold of connected robots the execution time is rising over the amount of robots

   after the threshold of connected robots the execution time is higher for A1 than for A0 when more robots are connected

6.5 Part 5: Absolute data consistency and malfunction detection

The evaluation of this section is based on two parts. A practical test and a evaluation based on data used from the timing measurements in section 6.4.

6.5.1 Planned evaluation

In a first part the watchdog solution for absolute data consistency is implemented in the tracker node and the system launched. Then ten nodes will be connected and some nodes disconnected using the malfunction service, which lets the node disconnecting without calling the disconnection service. In a second step, the difference between the absolute data consistency method is compared to the method using update misses detected by data history. The comparison is done with data from the timing measurements. To simulate the behaviour of the data consistency algorithm, the data was checked if at the timestamp 3 which is the sending of the state list the data was older than one update period.

6.5.2 Results

When the system was launched and the nodes disconnected with the malfunction client, the connection misses were indicated in the log output. The indication was repeating periodically, until the node was connected again. The results of the calculations run in Matlab can be found in table 12. It can be seen, that the data consistency method is indicating more update misses.

Table 12: Detected update misses using data history and absolute data age

Timing	1ms		2ms		5ms	
Architecture	A0	A1	A0	A1	A0	A1
Data history method	12727	836	11475	112	4730	387
Absolute data age method	13254	13715	11911	12764	4831	1548

7. Discussion and overall limitations

In this section, the results presented in section 6. are analysed and discussed. The outcomes of each implementation iteration are taken to conclude further implementations in the following iterations.

7.1 Part 1: Main Simulation

The proper functioning and implementation of the ATR system excerpt will be tested during the first part of the evaluation and robot launch. The two different system architectures can be seen working. The launch procedure is created dynamically so that the setting for the parameters can be switched without big configuration effort. Furthermore, the simulation provides the opportunity to launch each robot at an individual position, so the visual proof of launching the robots can be done with the visualisation. The goal, pose, and coordinate systems are communicated and visualised as expected. The frequencies in the measurement are as expected. With ten robots publishing their message with 10 Hz into one topic, a frequency around 100 Hz should result, which can be seen in figure 2. The division of the project into the different packages and the division of the launch script makes a future distribution of the code possible so that parts of the system can be executed on different computers. As a result of this step, a solid base for further implementation and evaluation of the dynamic connection handling is built. The results from the stress test are seen as errors in the underlying DDS and transportation layer. After investigating the error, a possible reason can be addressed that the assigned resources in terms of shared memory were insufficient to allow more publishers or subscribers to subscribe to the system, which is a logical reason for the system not allowing more robots to launch and connect.

7.2 Part 2: Dynamic connection handling

In this part of the thesis, a connection handling process for the dynamic launch and connection of robots to the ATR tracker is presented and implemented. The chosen approach is working with the ROS2 concept of services. Therefore, the tracker node provides a service function, which can be called by all the robots individually and with the passed parameters, the connection handling can be achieved. Theoretically, a connection handling could also be implemented using a publisher-subscriber hierarchy. The main difference between a service client connection handling over a publisher-subscriber approach is that calling the service allows sending a response and acknowledging or cancelling the connection. With a connection handling process using publisher subscribers in ROS2, either two different topics would need to be initialised for that purpose, or in an entirely created system, the feedback over a successful connection handling would be given at the end of the whole control loop. Furthermore, service is directed to the node calling it so that it could not happen that the acknowledge is received by another node that tries to connect simultaneously. The implementation of the connection service is done so that the nodes are informed whether their connection request is successful and then can start participating in the system by sending their status information periodically. Creating the publisher and the subscription in the ATR tracker under run-time requires a non-static execution of ROS2, as static executors are not executing new connections under run time. With the proposed procedure, the requirement is also full filled, that each robot node knows whether it is connected or not and does not accidentally starts publishing or participating in the system. The tests showed the connection working and its implementation working for all cases. The critical cases such as connecting and disconnecting in random order or connecting and disconnecting all robots were tested. The system stress test showed two different ROS2 systems were crashing. After some investigations about the error messages, it was found out that the first error in table 6 is caused when a node starts to call a service, but before it receives the response, the node is no longer existing. The service connection settings are reliable and can not be changed, so the service node crashes. In this case, this scenario means that it is not caused by the connection handling itself but could be caused by the Linux scheduler, which does not let the node execute longer as there are too many processes of launching nodes simultaneously. This problem should not let the ROS2 system crash, and it was reported as a bug in the Foxy version and written to be fixed in the latest version of ROS2 called Galactic. The second error message provided less information about the reason for the tracker node to crash, but it was also found that it might be related to the launch of too many nodes and requests simultaneously, and the Linux system is causing issues that let the ros execution crashing. It would need more investigation to fully understand the launch behaviour of many nodes at one computer. As this issue is not a primary issue for the created system and the connection handling was not the main reason causing the issue, more focus will be put into addressing that issue with an update of ROS2 and ensuring that not too many robots are launched simultaneously. In a real world system, the distribution of the robot nodes to their CPUs might already solve that issue even though many robots will not be launched simultaneously. An investigation might be fascinating in future work.

7.3 Part 3: Dynamic launch and ROS2 Update

The idea of delaying the robot launch and the ROS2 update came from the previous section's observations and was implemented and tested. The results show that updating the ROS2 version and changing the ROS2 vendor increased the maximum number of robots launched from 40 to over 180. Furthermore, the system limitation is caused by not being able to call the service anymore and not by any error that lets the tracker node crash. One outcome of this part is the launch behaviour. When all robots are launched on the same computer, the background operations might lead to a system crash while starting the system. This problem can be terminated by launching the robots in order with a specific time delay. Even though this result may vary in a real world system due to the distribution of computation, this change in launch behaviour significantly impacted the experimental outcomes. As a result of this section, a recommendation to update the ROS2 version and switch the DDS vendor from fastrtps to cyclonedds can be made. The combination of galactic and cyclone is in the simulation on a single computer working more solid than the combination of foxy and fastrtps. Further investigations of an effect of an update in the distributed system using a wireless network for communication would need to be investigated. However, the results from this section gave the first promising results, that an update can significantly improve the system's performance. Furthermore, an update of the system adds additional functionalities, which might be valuable in future work.

7.4 Part 4: Timing Measurements

Before discussing the findings from the preceding subsection, it is crucial to understand what factors affect the simulation results and what execution behaviour of ROS2 and Linux causes the results. Firstly it is essential to understand why the threshold exists and defines it. The logical explanation is the resource utilisation of the system running the simulation. In this simulation, unlike in a real world system, the execution of the various algorithms is not distributed to different processing units Timing, execution order, and schedules become increasingly crucial in high-utilisation areas. Slight deviations can explain the performance loss of the system in executions might already cause the scheduler of Linux to show a different behaviour, and misses and jitter in execution might occur more often. The behaviour is prompted by executing more robots as separate Linux processes. When processor utilisation is high, those queuing in the scheduler and the execution of the scheduler are responsible for the performance loss. This behaviour also explains that the threshold for 1ms and 2ms periods is around 20 while the threshold at 5ms is around 50 robots. It is also logically, as more scheduling activity is needed when 20 tasks are executed every 2 or 5 ms. At this system's limits, both the architectures are beginning to differ in their behaviour. To explain that behaviour, it is good to break the system down into the different observed phenomena. The first part is the update misses. The update miss ratio increases for A0 after the threshold, while it stays low for the A1. Therefore it is crucial to define what issues might cause update misses in the system. Update misses could happen in this specific system composition when the execution of the nodes concerning the tracker is jittering. A system's scheduler might, in certain instances, reverse the computing order of a node that executes periodically close the tracker, such that the robot computes after the tracker, or conversely. The robot is not executing during one period of the tracker's execution but is executing twice during another period. Both architectures exhibit behaviour that is about the same in terms of randomness and is dependent on the launch and execution of robots in relation to the tracker. A second reason for update misses, which only exists in A0, causes the actual raising in update misses for A0, and to clarify the raising update misses, it is necessary to comprehend the system's communication structure. In A0, the robots publish the message on the same topic. Each topic has a buffer that will be filled up when the receiving node cannot match the publishing speed. At the system's limit, the robots might execute so close together that the tracker receiving callback cannot match the speed for receiving the data. In that case, the buffer is filled up, and in the worst case, messages are dropped. As all robots are publishing on the same topic, a message drop is equivalent to an update miss, as the robot's data will not update concerning the previously published data. That happens more for more connected robots, especially when the scheduling cannot keep up. A1 features a topic and buffer for each robot so that a message drop does not equal an update miss since subsequent data from that robot is available. Next is the data age's behaviour when the robot threshold is reached, but to understand data ageing, it is essential to identify the different parts of the system that causes data ageing. The overall data age is divided into three parts, where the data is ageing. The first ageing occurs between creating the message in the robots and receiving the message in the tracker. The second part of ageing is the time between storing the message in the list and the periodic sending out of the list. The third part of ageing is the time between sending the message from the tracker and the reception in the measurement node. In general, the average data age influences the previously described update misses. Update misses are, in general, recognised in the second part of data ageing between T2T3, as the message was received long ago but never updated. So the latest data has been lying a long time inside the tracker node. The second factor is the data ageing during the communication, where the ageing is dependent on two factors. The first factor is the transportation of the message in the network and DDS. The second and, in this case, more important factor is the time it takes to read and process the incoming data.

In the collected data, it was seen that the total data age was rising with the high utilisation of the processor. It was also seen that the most significant impact on ageing in A0 was the time spent inside the tracker node. From the analysis, before and the update miss ration, this is explainable, as a lot more update misses are happening for this architecture precisely, and the update misses have a significant influence on the average age of the data. In the graphs in figure 20, it could also be seen that it is mostly the same robots which are not updating their data, which in the periodical case supports the theory of the dropped messages. As in a periodical case, the same robots are more likely that their message will be dropped. It can be seen in the graphs that the update miss ratio for A0 has reached a maximum of 98%, which would explain why the average age of the data is climbing so rapidly. Furthermore, it can be seen that the time between T1 and T2 has the most significant impact on the rise of the data age in A1. From the analysis, this is equal to a longer sending and computation of the message. For an understanding and explanation of the seen behaviour, it is vital to understand the procedure of receiving a message in ROS2, which is presented in section 2.2.4. A snapshot of the newest message in each topic is obtained throughout a node's execution. A new snapshot with new data will be created when all those messages and all timers are processed, meaning that later arriving messages are not processed and queued in the DDS under execution of the messages. If later arriving messages are not processed and queued also means that if A1 has many topics and, simultaneously, incoming messages, there is an extended processing time before the new data is taken. On the other hand, the processing time is low for A0 with only one topic, as each message will be fetched at once. In the high utilisation of the system, slow execution of the collecting of messages process will lead to an update miss for A0 and data ageing for A1 instead. On the other hand, processing the incoming data in groups is beside the ageing, achieving higher relative data consistency for robotic systems.

The overall differences in data age across architectures before the threshold and high utilisation depend more on the tracker execution than on the architectures. Another substantial impact was noticed in the measurement duration. As can be observed, the A1 measurement is more delayed relative to the A0. The timer executes less often when all messages are processed in one iteration, which is more computationally intensive. Overall, the architecture one seems to have a higher reaction time to data reception caused by the working wise of the ros execution. Especially with heavy computations during timer callbacks and more interfering timer callbacks, the reception time for the messages will be lower than in A1, which is also seen in table 7. The update misses caused by the message misses are at a certain point, causing the average age for data to increase much more than A1. In that cases, the A1 is showing its advantages and also allows a higher relative data consistency in the system by sacrificing reaction time towards new data and services. A1 is also more complex regarding resources, as the DDS needs more communication channels and more data queues. Other influences of e.g. quality of service settings for the communication like buffer size and reliability of communication were not evaluated in this measurements, but they might help to increase the performance in future evaluations.

7.5 Part 5: Absolute data consistency and malfunction detection

This part of the thesis is the most unfinished and will leave many opportunities for future work caused of the limited time frame of the thesis work. However, in a multi-agent robot system interacting in a dynamic environment and with humans, safety actions must be strictly defined. Furthermore, safety evaluations need to be done and the criticality of specific events evaluated. A connection miss or update misses might get detected in this system, showing the same behaviour. The simulated methods use absolute data consistency as a metric for update misses. The evaluation has shown that using the update period as a maximum data age causes more false detections of connection losses caused by jitter in execution and update misses. When the system reaches its operational limit, its execution jitter of the nodes and communication delay of messages increases which would lead to more false indications. The absolute data consistency was the more resource optimised solution in terms of stored data, but implementing a history of data could furthermore be utilised for synchronising data and achieving relative data consistency, which might be needed. As no definitions about the criticality of update misses are given, the two options for implementing a letting open for future work. In the implemented simulation, all real misses of connection using the absolute data consistency method utilised by a watchdog timer were detected, even though in some iterations, for short times connection misses were indicated of other connected robots which actually were not disconnected. Those false indications were not lasting more than one update period, which, dependent on a chosen criticality of the system, would not trigger safety actions. The effectivity of the buffer solution was not tested in the simulation. Actual measurements where the real execution of nodes is measured would need to confirm the results.

7.6 Overall Limitations

The whole thesis work is conducted via simulation on a single computer. Initially, it was planned to perform some of the experiments and evaluations on a distributed network, but the time was not allowed. The setup is causing some limitations to the thesis. Firstly, a dynamic connection handling is not only influencing the implementation of the tracker and the nodes. Furthermore, the control algorithms might need to be adapted in the whole system. Those adaptions could not be implemented or tested, as the legacy system was not available for tests. Secondly, the described factor of the centralisation and computing on a single computer is not mirroring the legacy system behaviour. The seen errors in the simulation might be different while testing the limits of a real distributed system. The measured values for the maximum of connected robots can be used to relate them to each other. Besides that relation, the values do not have any other meaning. The same appears for the timing measurements. With the chosen update periods, a real and distributed system might behave differently so that the thresholds for behaviour at a specific amount of connected robots might be different, and a real network between two nodes will influence even the system behaviour with the data ageing. Furthermore, the thesis only built an approach and tested for a specific excerpt from the system so that implementation in the legacy system is not given. All the simulated values might even differ at a different computer, but the system's overall behaviour at a computational limit will be more explainable and foreseeable with this thesis. The system implementation was done with default settings for the DDS, and even the message types were predefined by the system, meaning that dependencies on those values are not investigated and might influence an application.

8. Conclusions

In the thesis a dynamic connection handling approach is proposed, implemented and tested in a simulation. The simulation environment was created using ROS2 and RVIZ, building an excerpt from the legacy GPSS system. A performance test has been done regarding the amount of connected robots simulated on a single computer. After implementing and performing the test and evaluations, a final answer to the research questions can be given, as well as implications for the GPSS system be made.

8.1 Answer to research question 1

The research question was stated as follows:

RQ1: How can nodes in a ROS2 network be connected and disconnected dynamically during run-time while detecting unexpected connection losses?

Nodes can be connected and disconnected under run-time while communicating via client and service structure. While connecting through client and service structure, nodes may be connected and disconnected dynamically. The node that wants to connect or disconnect from the network initiates this process by calling a general service that performs a function within the tracker node. With a known communication architecture, the tracker can initialise or delete the specific subscriber and acknowledge the connection request. The connection handling inside the tracker can be initialised by vectors and dynamic look-up tables for the id handling. After the acknowledgement, the connecting node can initialise its publisher and start publishing its data. ROS2 allows the dynamic creation of publishers, subscribers and timers under run time using non-static executioners. The absolute data consistency in message age can detect connection losses and the internal buffer following the communication history. Execution jitters might also be recognised as connection losses, but multiple missed data updates in a row can indicate real connection losses.

8.2 Answer to research question 2

The research question was stated as follows:

RQ2: How do the increase of connections to one node and the publishing frequency influence the performance of a ROS2 publisher-subscriber system regarding data age and update misses?

The conducted simulations and measurements have shown, in general, that the data age is increasing over the number of connected robots for all publishing periods. The chosen communication approach furthermore influences the data age. In a many to one approach, where multiple nodes communicate through the same topic, the data ageing is bigger than for the one-to-one approach, where each communicating node has its own message. The most significant influence on the data ageing in the many to one approach is the update misses caused by the high utilisation and scheduling of the processor and operating system. That also answers the second part of the question regarding the update misses. The update miss ratio only increased significantly for the many to one communication approach over the number of connected robots. Before the seen raise and for the second architecture, no correlations for the number of robots to the update miss ratio and data age could be drawn from the results. A low update period was increasing the seen and described phenomena of the data ageing and update misses, so that the increase was seen with a lower amount of connected robots. Before the substantial rise, the data age depended on the execution of the robot nodes in relation to the tracker node and, therefore, within the update period and could theoretically be more significant with a higher period.

8.3 Implications for the ATR System

A connection handling approach was designed and implemented in a simulation during the thesis work. The designed approach was tested successfully in the simulation and can be taken and implemented in the legacy system. Furthermore, it was found that an update of the ROS2 version and DDS vendor might improve the system robustness in cases for more connected nodes. Furthermore, an update of the ROS2 version and the DDS vendor allows more opportunities for setting QoS and communication parameters, which could benefit the system performance. The evaluation of two different communication approaches for connecting the robots to the tracker has shown that one-to-one communication works better in cases where more robots are connected. The utilisation reaches a specific threshold depending on the connected nodes and the update frequency. The threshold would need to be tested for the legacy system individually, as the simulation did not include a processing distribution like it can be found in the legacy system. Furthermore, from the processing of the data and the execution of the tracker node, relative data consistency might be easier with the one-to-one communication approach as well, as the data is processed in groups and not one by one like in the other architecture. Before that threshold, there was no correlation seen between the number of connected robots to the data age. With a lower update frequency, the maximum data age is most likely rising but dependent on the execution of the nodes towards each other. For a lower system complexity, the many to one architecture might be chosen in those cases with lower update rates or less maximum amount of connected robots, as the performance is also not influenced negatively compared to the second architecture. A theory was derived based on the execution of the nodes in ROS2, stating that the response time for the connection service might be lower with the one-to-one architecture. This theory has to be validated and tested in the legacy system.

9. Future work

The thesis work sets the starting point for further investigations on scalability in multi-agent robotic systems and further analysis of the ATR system. During the implementations and evaluations, tasks for future continuation were seen, described in the following sections.

9.1 System implementation

First, the designed and tested connection handling approach has to be implemented and tested in the legacy system. The system following control algorithms might need to be adapted to react to the dynamic number of robots in the network. Therefore the tracker node might be used to register the connecting nodes in all other calculating nodes. Another aspect regarding the communication of the state messages of the robots might be real-time databases, which can be used to achieve relative data consistency.

9.2 Malfunction detection and safety actions

Regarding the malfunction detection, further definitions need to be made and an implementation realised as well as tested in a real environment. Safety analysis with the criticality of special events needs to be performed to decide about possible safety actions in case those events are happening. Those events include update misses and connection losses.

9.3 Timing measurements and systems modelling

Regarding the conducted timing measurements, the outcomes and stated theories for explaining the results need to be validated by further experiments and more data collection. Therefore in the first step, more data for the different timings and especially dependencies on the underlying scheduler load and CPU utilisation would be helpful. Furthermore, the measurements need to be repeated in a distributed architecture, which would give more results about a real world system, including a real network and the processor utilisations. Another possible measurement for validating the results and stated theories for explaining the results are measurements of real execution of the nodes, where not only the messages are tracked and timestamped, but every execution of each node will be measured. Further data about the two different architectures, especially with service response time, would be desirable to validate the executioner working wisely in ROS2 and give further recommendations for preferred system architectures. A theoretical analysis or modelling of the system is another possibility for more in-depth system analysis. The components and execution of the ROS2 communication and especially communication chains would be good to analyse with theoretical models. As no models or methods for analysing more complex processing chains, this would be a field for future research.

9.4 ROS versions and DDS vendors

The influence and differences between the different used ROS2 versions and DDS vendors might be another focus for future work. Therefore more tests can be performed in the legacy system or other simulations, investigating performance with more participating nodes and network-related measurements with the DDS.

9.5 QoS

The executed ROS2 update during the thesis leads to the fact that way more settings regarding the QoS in the communication and computation of callbacks are possible. The thesis uses the default settings for QoS. Settings like the buffer size might influence the seen behaviour regarding the data ageing, and possible influences might be investigated. Furthermore, prioritisation of callbacks is possible with ROS2 galactic, which might help decouple the connection handling execution from the processing of the incoming state messages.

References

[1] M. Ford, ‘Rise of the robots : Technology and the threat of a jobless future,’ 2015.

[2] T. Arai and J. Ota, ‘Dwarf intelligence—a large object carried by seven dwarves,’ *Robotics and Autonomous Systems*, vol. 18, no. 1-2, pp. 149–155, 1996.

[3] ——, ‘Let us work together-task planning of multiple mobile robots,’ in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS’96*, IEEE, vol. 1, 1996, pp. 298–303.

[4] M. J. Matarić, ‘From local interactions to collective intelligence,’ in *The biology and technology of intelligent autonomous agents*, Springer, 1995, pp. 275–295.

[5] D. Rus, B. Donald and J. Jennings, ‘Moving furniture with teams of autonomous robots,’ in *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, vol. 1, 1995, 235–242 vol.1.

[6] A. Khalif and L. Jutvik, ‘Path following for atrs using embedded nonlinear model predictive control,’ 2021.

[7] X. Xu and C. Wang, ‘Indoor navigation based on global and local path planning for wheeled mobile robot in airport,’ in *2018 13th World Congress on Intelligent Control and Automation (WCICA)*, 2018, pp. 1002–1007. DOI: 10.1109/WCICA.2018.8630407.

[8] M. A. Gomaa, O. De Silva, G. K. Mann, R. G. Gosine and R. Hengeveld, ‘Ros based real-time motion control for robotic visual arts exhibit using decawave local positioning system,’ in *2020 American Control Conference (ACC)*, 2020, pp. 653–658. DOI: 10.23919/ACC45564.2020.9148022.

[9] D. Bozhinoski, I. Malavolta, A. Bucchiarone and A. Marconi, ‘Sustainable safety in mobile multi-robot systems via collective adaptation,’ in *2015 IEEE 9th International Conference on Self-Adaptive and Self-Organizing Systems*, 2015, pp. 172–173. DOI: 10.1109/SASO.2015.31.

[10] F. Yili, X. Hongyan, W. Shuguo, L. Jianguo, X. He and L. Han, ‘Mobile robot control based on fuzzy behavior and robot safety body in unknown environment,’ in *2005 International Symposium on Computational Intelligence in Robotics and Automation*, 2005, pp. 547–552. DOI: 10.1109/CIRA.2005.1554334.

[11] F. Schilling, E. Soria and D. Floreano, ‘On the scalability of vision-based drone swarms in the presence of occlusions,’ *IEEE Access*, vol. 10, pp. 28 133–28 146, 2022. DOI: 10.1109/ACCESS.2022.3158758.

[12] Y. Aydin, G. K. Kurt, E. Ozdemir and H. Yanikomeroglu, ‘Authentication and handover challenges and methods for drone swarms,’ *IEEE Journal of Radio Frequency Identification*, pp. 1–1, 2022. DOI: 10.1109/JRFID.2022.3158392.

[13] K. Pålsson and E. Svärling, ‘Nonlinear model predictive control for constant distance between autonomous transport robots,’ 2020.

[14] D. Carlsson and E. Malmquist, ‘Develop an anti-collision system for a fleet of atrs in a distributed real time environment using ros2/dds,’ 2021.

[15] *AUTOSAR Techincal Overview, Release 4.1, Rev.2, Ver.1.1.0.* http://autosar.org.

[16] S. Mubeen, T. Nolte, M. Sjödin, J. Lundbäck and K.-L. Lundbäck, ‘Supporting timing analysis of vehicular embedded systems through the refinement of timing constraints,’ *Softw. Syst. Model.*, vol. 18, no. 1, pp. 39–69, Feb. 2019, ISSN: 1619-1366. DOI: 10.1007/s10270-017-0579-8. [Online]. Available: https://doi.org/10.1007/s10270-017-0579-8.

[17] *Robot operating system*, https://www.ros.org/blog/why-ros/, Accessed: 2022-01-23.

[18] J. A. Stankovic and R. Rajkumar, *Real-time operating systems - real-time systems.* [Online]. Available: https://link.springer.com/article/10.1023/B:TIME.0000045319.20260.73.

[19] O. Robotics, *Ros2 documentation*, Accessed: 2021-12-06. [Online]. Available: https://docs.ros.org/en/galactic/index.html.

[20] D. Casini, T. Blaß, I. Lütkebohle and B. Brandenburg, 'Response-time analysis of ros 2 processing chains under reservation-based scheduling,' in *31st Euromicro Conference on Real-Time Systems*, Schloss Dagstuhl, 2019, pp. 1–23.

[21] D. Schmidt, M. Stal, H. Rohnert and F. Buschmann, *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2.* Jan. 2000.

[22] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M. Stal, *Pattern-oriented software architecture, a system of patterns, ser. pattern-oriented software architecture*, 2013.

[23] M. QUIGLEY, *Programming Robots with Ros.* SHROFF Publishers &amp; DISTR, 2016.

[24] D. E. Comer, *Computer networks and internets.* Academic Internet Publishers, 2007, vol. 5.

[25] *Wireless reliability: Rethinking 802.11 packet loss.* [Online]. Available: `https://ieeexplore.ieee.org/abstract/document/4594875/`.

[26] G. M.
bibinitperiod J. Z.
bibinitperiod K.-W. S. & and B. Slimane, *Fundamentals of mobile data networks*, 2016.

[27] F. P.
bibinitperiod M. Hawkins, *High availability: Design, techniques, and processes*, 2001.

[28] P. C. Gupta, *Data communications and computer networks*, 2011.

[29] K. Kurose J.F; Ross, *Computer networking: A top-down approach*, 2013.

[30] *Transmission Control Protocol*, Sep. 1981. [Online]. Available: `https://www.rfc-editor.org/info/rfc793`.

[31] *User Datagram Protocol*, RFC 768, Aug. 1980. DOI: `10.17487/RFC0768`. [Online]. Available: `https://www.rfc-editor.org/info/rfc768`.

[32] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, 'Scheduling: Introduction, multi-level feedback queue, proportional-share scheduling, multiprocessor scheduling,' in *Operating systems: Three easy pieces.* Arpaci-Dusseau Books, LLC, 2018.

[33] G. Pardo-Castellote, 'Omg data-distribution service: Architectural overview,' in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, IEEE, 2003, pp. 200–206.

[34] *Eprosima fast dds documentation*, https://fast-dds.docs.eprosima.com/en/latest/, Accessed 10-05-2022.

[35] *Cyclone dds*, https://projects.eclipse.org/projects/iot.cyclonedds, Accessed 10-05-2022.

[36] N. C. Audsley, A. Burns, M. F. Richardson and A. J. Wellings, 'Absolute and relative temporal constraints in hard real-time databases,' in *Fourth Euromicro workshop on Real-Time Systems*, IEEE, 1992, pp. 148–153.

[37] S. Dehnavi, M. Koedam, A. Nelson, D. Goswami and K. Goossens, 'Compros: A composable ros2 based architecture for real-time embedded robotic development,' in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 6449–6455. DOI: `10.1109/IROS51168.2021.9636590`.

[38] A. Testa, A. Camisa and G. Notarstefano, 'Choirbot: A ros 2 toolbox for cooperative robotics,' *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2714–2720, 2021. DOI: `10.1109/LRA.2021.3061366`.

[39] C. McCord, J. P. Queralta, T. N. Gia and T. Westerlund, 'Distributed progressive formation control for multi-agent systems: 2d and 3d deployment of uavs in ros/gazebo with rotors,' in *2019 European Conference on Mobile Robots (ECMR)*, 2019, pp. 1–6. DOI: `10.1109/ECMR.2019.8870934`.

[40] S. Vorapojpisut, M. Lhongpol, R. Amornlikitsin and T. Phuapaiboon, 'A robot augmented environment based on ros multi-agent structure,' in *2019 4th International Conference on Control, Robotics and Cybernetics (CRC)*, 2019, pp. 52–56. DOI: `10.1109/CRC.2019.00020`.

[41] G. Conte, D. Scaradozzi, L. Sorbi, L. Panebianco and D. Mannocchi, 'Ros multi-agent structure for autonomous surface vehicles,' in *OCEANS 2015 - Genova*, 2015, pp. 1–6. DOI: `10.1109/OCEANS-Genova.2015.7271543`.

[42] S. Noh and J. Park, 'System design for automation in multi-agent-based manufacturing systems,' in *2020 20th International Conference on Control, Automation and Systems (ICCAS)*, 2020, pp. 986–990. DOI: `10.23919/ICCAS50221.2020.9268357`.

[43] A. Barciś, M. Barciś and C. Bettstetter, 'Robots that sync and swarm: A proof of concept in ros 2,' in *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, 2019, pp. 98–104. DOI: `10.1109/MRS.2019.8901095`.

[44] D. Casini, T. Blaß, I. Lütkebohle and B. Brandenburg, 'Response-time analysis of ros 2 processing chains under reservation-based scheduling,' in *31st Euromicro Conference on Real-Time Systems*, Schloss Dagstuhl, 2019, pp. 1–23.

[45] T. Blaß, D. Casini, S. Bozhko and B. B. Brandenburg, 'A ros 2 response-time analysis exploiting starvation freedom and execution-time variance,' in *2021 IEEE Real-Time Systems Symposium (RTSS)*, 2021, pp. 41–53. DOI: `10.1109/RTSS52674.2021.00016`.

[46] Y. Tang, Z. Feng, N. Guan *et al.*, 'Response time analysis and priority assignment of processing chains on ros2 executors,' in *2020 IEEE Real-Time Systems Symposium (RTSS)*, 2020, pp. 231–243. DOI: `10.1109/RTSS49844.2020.00030`.

[47] H. Choi, Y. Xiang and H. Kim, 'Picas: New design of priority-driven chain-aware scheduling for ros2,' in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021, pp. 251–263. DOI: `10.1109/RTAS52030.2021.00028`.

[48] N. Ding, X. Meng, Y. Zhao and D. Wu, 'Ros task scheduling algorithm in multi-core system,' in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, 2019, pp. 516–519. DOI: `10.1109/ICPADS47876.2019.00078`.

[49] Y. Saito, T. Azumi, S. Kato and N. Nishio, 'Priority and synchronization support for ros,' in *2016 IEEE 4th International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA)*, 2016, pp. 77–82. DOI: `10.1109/CPSNA.2016.24`.

[50] Z. Li, A. Hasegawa and T. Azumi, 'Autoware-perf: A tracing and performance analysis framework for ros 2 applications,' *Journal of Systems Architecture*, vol. 123, p. 102 341, 2022, ISSN: 1383-7621. DOI: `https://doi.org/10.1016/j.sysarc.2021.102341`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S1383762121002344`.

[51] Y. Yang and T. Azumi, 'Exploring real-time executor on ros 2,' in *2020 IEEE International Conference on Embedded Software and Systems (ICESS)*, 2020, pp. 1–8. DOI: `10.1109/ICESS49830.2020.9301530`.

[52] Y. Maruyama, S. Kato and T. Azumi, 'Exploring the performance of ros2,' in *2016 International Conference on Embedded Software (EMSOFT)*, 2016, pp. 1–10. DOI: `10.1145/2968478.2968502`.

[53] J. Park, R. Delgado and B. W. Choi, 'Real-time characteristics of ros 2.0 in multiagent robot systems: An empirical study,' *IEEE Access*, vol. 8, pp. 154 637–154 651, 2020. DOI: `10.1109/ACCESS.2020.3018122`.

[54] T. Blass, 'Real-time execution management in the ros 2 framework,' *Ph.D. dissertation*, 2022.

[55] L. Puck, P. Keller, T. Schnell *et al.*, 'Performance evaluation of real-time ros2 robotic control in a time-synchronized distributed network,' in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, 2021, pp. 1670–1676. DOI: `10.1109/CASE49439.2021.9551447`.

[56] L. Puck, P. Keller, T. Schnell *et al.*, 'Distributed and synchronized setup towards real-time robotic control using ros2 on linux,' Oct. 2020. DOI: `10.1109/CASE48305.2020.9217010`.

[57] C. S. V. Gutiéerrez, 'Towards a distributed and real-time framework for robots: Evaluation of ros 2.0 communications for real-time robotic applications,' *Tech. Rep*, 2018.

[58] E. Erős, M. Dahl, K. Bengtsson, A. Hanna and P. Falkman, 'A ros2 based communication architecture for control in collaborative and intelligent automation systems,' *Procedia Manufacturing*, vol. 38, pp. 349–357, 2019, 29th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM 2019), June 24-28, 2019, Limerick, Ireland, Beyond Industry 4.0: Industrial Advances, Engineering Education and Intelligent Manufacturing, ISSN: 2351-9789. DOI: `https://doi.org/10.1016/j.promfg.2020.01.045`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S2351978920300469`.

[59] R. Morita and K. Matsubara, 'Dynamic binding a proper dds implementation for optimizing inter-node communication in ros2,' in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2018, pp. 246–247. DOI: `10.1109/RTCSA.2018.00043`.

[60] J. Kim, J. Smereka, C. Cheung, S. Nepal and M. Grobler, *Security and performance considerations in ros 2: A balancing act*, Sep. 2018.

[61] T. Kronauer, J. Pohlmann, M. Matthé, T. Smejkal and G. Fettweis, 'Latency analysis of ros2 multi-node systems,' in *2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, 2021, pp. 1–7. DOI: `10.1109/MFI52462.2021.9591166`.

[62] Y. Maruyama, S. Kato and T. Azumi, 'Exploring the performance of ros2,' in *Proceedings of the 13th International Conference on Embedded Software*, ser. EMSOFT '16, Pittsburgh, Pennsylvania: Association for Computing Machinery, 2016, ISBN: 9781450344852. DOI: `10.1145/2968478.2968502`. [Online]. Available: `https://doi.org/10.1145/2968478.2968502`.

[63] Z. Chen, 'Performance analysis of ros 2 networks using variable quality of service and security constraints for autonomous systems,' Naval Postgraduate School Monterey United States, Tech. Rep., 2019.

[64] J. F. N. Jr., M. Chen and T. D. Purdin, 'Systems development in information systems research,' *Journal of Management Information Systems*, vol. 7, no. 3, pp. 89–106, 1990.

[65] A. Sangiovanni-Vincentelli and G. Martin, 'Platform-based design and software design methodology for embedded systems,' *IEEE Design Test of Computers*, vol. 18, no. 6, pp. 23–33, 2001. DOI: `10.1109/54.970421`.