Decentralised Multi-robot manufacturing simulation

Master Thesis

Marek Raška

Aalborg University
Department of Electronic Systems
Fredrik Bajers Vej 7B
DK-9220 Aalborg

Preface

This Master Thesis has been completed as a part of obtaining master thesis title done in coordination with study curriculum at Aalborg University with framework theme focusing on swarm robotics used in manufacturing. This thesis is the culmination of work achieved in Decentralised Multi-robot manufacturing simulation.

October 23, 2023

Marek Raška
<mraka21@student.aau.dk>

Acronyms and abbreviations

Acronym	**Definition**
ROS2	Robot Operating System 2
CNN	Convolutional Neural Network
DNN	Deep Neural Network
EKF	Extended Kalman Filter
GUI	Graphical User Interface
IMU	Inertial Measurement Unit
SLAM	Simultaneous Localisation And Mapping
SOTA	State Of The Art
URDF	Unified Robot Description Format
Open-RMF	Open Robotics Middleware Framwork
PoI	Points of Interest
LTS	Long term support
API	application programming interface

Contents

1 - Introduction

In recent years, the manufacturing industry has been part of a significant shift towards automation and robotics, driven by the need to enhance productivity, flexibility, and cost-effectiveness compared to simple robot manufacturing. Traditional centralized manufacturing systems, where a single controller manages and coordinates the activities of all robots and systems, have limitations in terms of scalability, adaptability, and fault tolerance. To address these challenges, the concept of decentralized multi-robot manufacturing has emerged as a promising approach. This thesis explores the potential of decentralized multi-robot manufacturing through simulation, with the aim of optimizing production processes and achieving greater efficiency in the factory of the future.

The essence of decentralized multi-robot manufacturing lies in the distribution of decision-making and control among a network of autonomous robotic agents. Each robot possesses its own sensing, reasoning, and acting capabilities, allowing it to perform tasks independently or collaboratively with other robots. This paradigm shift empowers manufacturing systems to operate in a more flexible, scalable, and fault-tolerant manner compared to the centralized approach.

In a centralized manufacturing system, a single controller is responsible for managing and coordinating the activities of all robots. While this approach provides a high level of control and coordination, it can become a bottleneck as the number of robots increases. The centralized controller must process and respond to all the information from the robots, leading to potential delays and decreased overall system performance. Additionally, a single point of failure in the centralized controller can bring the entire system to a halt and all the robots with it. This lack of scalability limits the potential of centralized manufacturing systems to adapt to changing production requirements or recover from failures quickly.

In contrast, decentralized multi-robot manufacturing systems distribute decision-making and control among multiple autonomous robots. Each robot can make decisions based on local information and collaborate with other robots to perform tasks efficiently. This decentralized approach enables a higher degree of scalability, as the addition or removal of robots does not heavily impact the overall system performance. Furthermore, by removing the reliance on a single controller, the system becomes more robust, as failures in individual robots do not necessarily

Figure 1.1: BMW logistics robots in colaboration with NVIDIA [1].

disrupt the entire system. Decentralization also allows for greater adaptability, as robots can dynamically allocate tasks and adjust their behavior based on changing conditions. That is at least in theory as dynamically changing behavior is still challenging to program even in realm of machine learned models.

The primary objective of this master's thesis is to develop a comprehensive simulation framework for decentralized multi-robot manufacturing systems and compare it with the centralized approach. By using the power of simulation, this thesis aims to explore and evaluate the advantages and disadvantages of both approaches in terms of control, scalability, adaptability, and fault tolerance. The simulation environment will provide a platform for modeling and analyzing the interactions between multiple robots, their coordination algorithms, and resource management techniques.

Through this research, this thesis seeks to show valuable insights into the capabilities and limitations of decentralized multi-robot manufacturing systems in comparison to the centralized approach. By identifying the trade-offs and potential benefits of each approach, we aim to contribute to the decision-making process for manufacturers considering the implementation of collaborative automation systems. Ultimately, through the simulation-based approach, we hope to unlock the potential of decentralized multi-robot manufacturing and pave the way for more efficient, flexible, and resilient factories of the future. Thesis and projects such as this are made in hope that this becomes a standard tool that will help people make better robotic systems. In the future this swarm simulation workbench could help

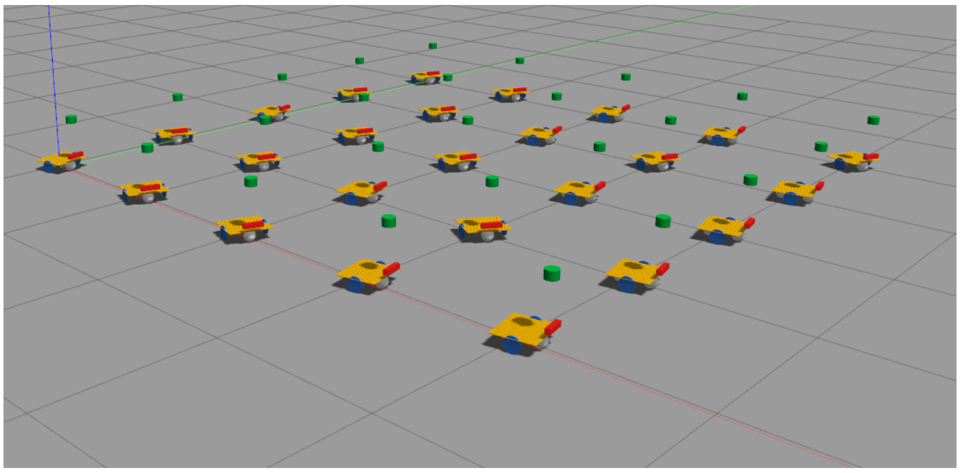


Figure 1.2: Simulated robot swarm in Gazebo simulator.

more fields is not limited to only research in manufacturing, but any decentralised robotics research such as intelligence emergence.

How can robotic swarm operate independently from each other and still achieve common goal in manufacturing process?

2 - Problem Analysis

To create simulation for multi robot manufacturing scenario I will need software on which to simulate this scenario, framework on which to operate these robots and control them. The simulation control should be also usable for real life applications. In this chapter I will focus on comparing available resources that can be used in achieving this simulation, from simulators, scenarios and worlds running on these simulators, maps made for them. This includes setting up important way-points for robots (be it to workstation or for general navigation), to visualising what tasks the robots do in the simulated world.

2.1 3D Robotics Simulators

This section compares simulation engines that are used for simulating robots from Gazebo and NVIDIA's Isaac to Unity. These are the most popular simulators and each will have their own section explaining their capabilities and features. At the end of this section they will be compared in their strengths and weaknesses to find which one is most suitable for the task of simulating environment of multi robot manufacturing for our given problem.

2.1.1 Gazebo

Gazebo is in the developers own words Advanced robot simulator for research, design, and development. As it is an open project with direct collaboration with ROS2 it has been designed to offer high-performance simulation capabilities. It supports distributed simulation, where computation is distributed across multiple servers, leading to improved performance. Additionally, Gazebo can automatically load and unload simulation assets based on spatial information, further enhancing performance. The ability to tune the simulation time step size allows running simulations in real-time, faster than real-time, or slower than real-time, providing flexibility in performance settings. This is very usefull feature as it makes gazebo work even on weaker machines.

Gazebo is cross-platform, currently supporting Linux and MacOS and Windows . It also seamlessly integrates with the cloud, enabling users to access, download, and upload simulation models and worlds on a cloud-hosted server at

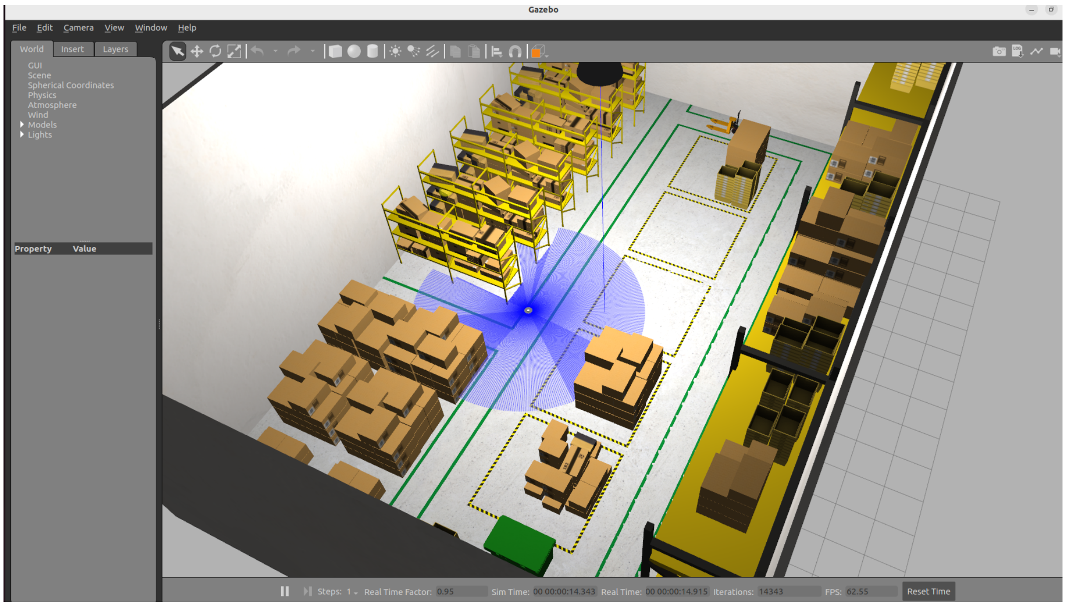


Figure 2.1: Gazebo environment showcase [2].

app.gazebosim.org, this might slow down first startup of simulation server a lot, because of models downloads. Local models have to be added to system gazebo_path Integration with ROS2 is possible via a ROS/Gazebo bridge since ROS Melodic, ensuring smooth communication and data exchange between the two systems.

Gazebo offers an array of sensors, including monocular cameras, depth cameras, LIDAR, IMU, contact sensors, altimeters, and magnetometers, all of which can be utilized within simulations. Its default physics engine, DART, offers highly precise and accurate physics simulations, surpassing game engines' capabilities. You can use your own physics engine and this is supported by gazebo from command line or C++ API.

Extensibility is a key feature of Gazebo, allowing its users to customize and enhance the simulation environment. The majority of Gazebo libraries offer a plugin interface, enabling the use of custom code at runtime. This allows for the integration of additional rendering engines and GUI libraries. Gazebo also provides a plugin simulation systems mechanism, allowing the loading of custom systems that can interact directly with the simulation, enabling introspection and modification of the simulation on the fly. Usability of custom systems differs from version to version, so strict version control is required.

Gazebo also offers convenient command-line interfaces through the "gz" command-line tool, which provides tools for topic introspection, message introspection, launch management, and logging. For a more visual experience, a graphical in-

terface based on QtQuick is available, allowing users to visualize the simulation environment and access plugins for topic visualization, message delivery, and simulation world control and statistics. Gazebo also provides a web interface, which enables users to discover new simulation assets, manage their assets, participate in simulation competitions, and run simulations on cloud resources.
In summary, Gazebo offers a robust and flexible simulation environment that prioritizes performance. [2]

2.1.2 NVIDIA Isaac ROS

Isaac ROS provides packages (GEMs) and complete pipelines (NITROS) for image processing and computer vision, highly optimized for NVIDIA GPUs and Jetson platforms. This means Isaac has a lot of prepared professional packages with high fidelity ready to use. Downside is that this is only usable on the latest NVIDIA GPUs.
The flexibility of modular packages empowers anyone developing on Isaac to select precisely the components they need for integration into their applications. This modular approach facilitates seamless replacement of entire pipelines or individual algorithms, offering an efficient and adaptable solution for simulation.
Isaac boasts an extensive collection of Perception AI Packages, encompassing cutting-edge DNN-based algorithms crucial for achieving high-performance perception. These packages are designed to harness the power of hardware acceleration, contributing to expedited development processes.
The latest Humble ROS 2 release enhances performance by harnessing hardware accelerators on compute platforms. NITROS, an NVIDIA implementation of type adaptation and negotiation, comprises hardware-accelerated modules known as GEMs within the Isaac ROS ecosystem. Developers now have access to the source code of NITROS, granting them the freedom to customize and extend its functionalities according to their specific requirements.
Isaac also offers the Stereo Visual Odometry (SLAM) GEM, delivering highly accurate real-time stereo camera visual odometry. This functionality is seamlessly integrated into existing applications, ensuring precise localization capabilities.
3D Scene Reconstruction with nvblox in its preview stage, utilizes RGB-D data to create a dense 3D representation of a robot's surroundings. This advanced feature aids in the detection of unforeseen obstacles, crucial for real-time decision-making and safe navigation. This is basically RVIZ for gazebo, but with better GUI and features.
DNN Inference GEM presents a collection of ROS 2 packages that enable developers to leverage NVIDIA's extensive library of inference models, whether from NGC or custom creations. These packages facilitate fine-tuning and optimization of pre-trained models using the NVIDIA TAO Toolkit. The deployment of these

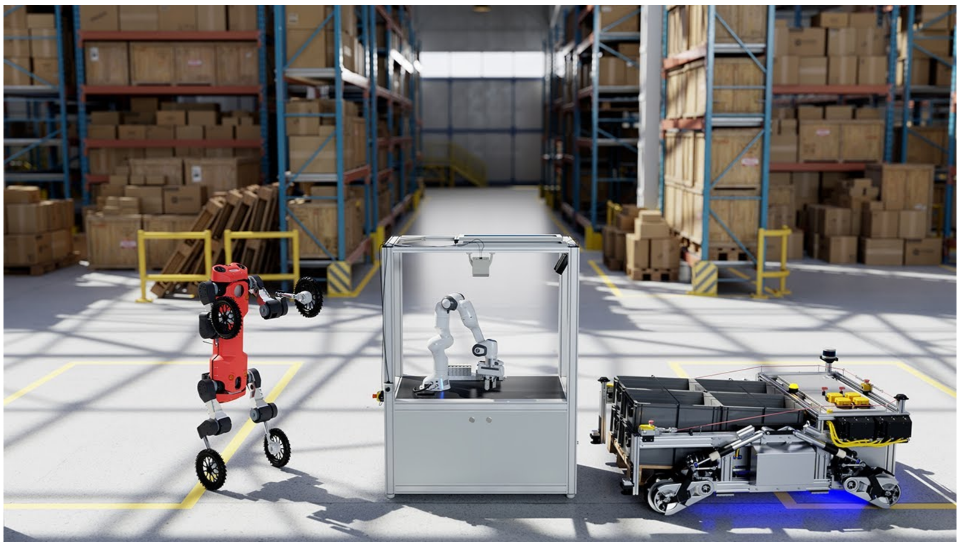


Figure 2.2: NVIDIA Isaac simulation environment showcare [3].

optimized models is achieved through TensorRT or Triton, NVIDIA's inference server. Optimal inference performance is ensured through TensorRT, while Triton offers flexibility for unsupported models. This presents easy to use machine learning connection between Isaac simulation and real life collected data.
In conclusion, Isaac is comprehensive platform for high-fidelity simulation, with a wide array of tools and capabilities designed to empower robotics applications with SOTA (State of the art) AI-driven functionalities.[3]

2.1.3 Unity

Unity, a robust and versatile game development engine, emerges as a pivotal tool for crafting interactive 3D simulations, including those appropriate to the field of robotics.
Unity excels in the realm of real-time graphics and physics simulation, enabling the creation of visually stunning and physically precise environments for simulations. This attribute is pivotal for achieving a high degree of realism in simulated scenarios.
Unity boasts an extensive ecosystem comprising a plethora of assets, plugins, and tools accessible through the Unity Asset Store. This invaluable resource empowers users with pre-built models, environments, and scripts, expediting the development process and enhancing the quality of simulations. As a game engine Unity has an edge in pre-build resources such as models and worlds, when it comes to their quantity and quality.
Unity's compatibility with multiple platforms, encompassing desktop, mobile,

Figure 2.3: Unity Robotics Demo Showcase [5].

and web, offers significant convenience when deploying simulations across diverse devices. This compared to other robotics simulators opens the possibilities for simulation on different devices other than high-end servers and desktops.
Unity uses C# as its primary scripting language, which provides a familiar and accessible programming environment for developers. While Unity lacks native integration with the Robot Operating System (ROS), it is worth noting that community-driven packages like ROS (ROS Sharp) have emerged to bridge the gap between Unity and ROS.These packages facilitate seamless communication between Unity-based simulations and ROS-based robotic systems, allowing for the exchange of data and commands, a crucial consideration for robotics-oriented applications.[4]

2.1.4 comparison of the simulators

Gazebo, NVIDIA Isaac ROS and Unity all have their strong points, such as extensive model library for gazebo and Unity, high fidelity AI driven simulation for Isaac. So lets focus on their drawbacks, which every single one of these simulators have a few. For NVIDIA Isaac it is its hardware requirements. It is limited to only newer NVIDIA CUDA GPUs and its OS requirements are ironically outdated: Ubuntu 18.04 even though ROS2 Humble is recommended to run on Ubuntu 22.04. Unity has other problems as there is only fan support for robotics programming in that simulation environment as can be seen in ROS and is by no means official ROS supported bridge between python or C++ versions of ROS. That leaves us with Gazebo simulator. It is made from under Open robotics umbrella, which also has ROS itself and Open-RMF (Open Robot Middleware Framwork). Open-RMF could be though of as a centralised competition to this Master Thesis as it is framework for simulation and control of multiple robots thought central server. Gazebo also has a large user community, extensive resources and relatively low computer spec requirement. This concludes comparison between simulators, with Gazebo being chosen for this project.

2.2 Environment

Once Simulator is selected a creation of environment that robot will be performing in must take place. In Gazebo this environment is defined by the .world file, where various objects, models and properties can be added for the simulated world. This world can be made with inbuilt tools like Gazebo GUI, Gazebo Model Editor or defined directly using an XML format in the file. All the physical properties of the objects can be determined there, such are mass, friction, collision shapes etc.. This is for collision model purposes.
There can be collision model and visible model that differs from it. Visible model is usually defined as .sdf file or .model file. If the global Gazebo library does not contain model that you would want or if there is a need for more specific model they can be imported locally by sourcing gazebo_path to model system path. For creation of such models - custom models, Blender or CAD software can be used. Objects can be static structures like building or terrain that can be created by the gazebo building editor as can be seen on the figure 2.4. They can be also items enhancing the environment such as furniture, manufacturing station or any other number of things. They can be also dynamic models that react to robots or can be transported by them such as boxes, containers or other cargo items.

Visuals of the environment can be enhanced by lighting and visual effects that gazebo provides, so you can achieve the visual feel of the simulated scene or its dynamics such as day and night cycle. Gazebo is not focused on photo-realistic visuals that can be found in NVIDIA Isaac simulations, but it can still have realistic looking appearance with proper texturing and lighting combinations.

2.2.1 Spawning robots into the environment

Adding functional robot into environment requires a model specification file, which for ROS2 is either URDF (Unified Robot Description Format) or XACRO (XML Macros) file that defines robots. URDF (Unified Robot Description Format) is an XML-based file format used to describe the structure and visual properties of robots in the context of robot modeling and simulation. URDF files are commonly used in conjunction with ROS2 (Robot Operating System 2) for describing the kinematics, dynamics, and geometry of robots.
In URDF you can describe hierarchical structure of the robot, including links,

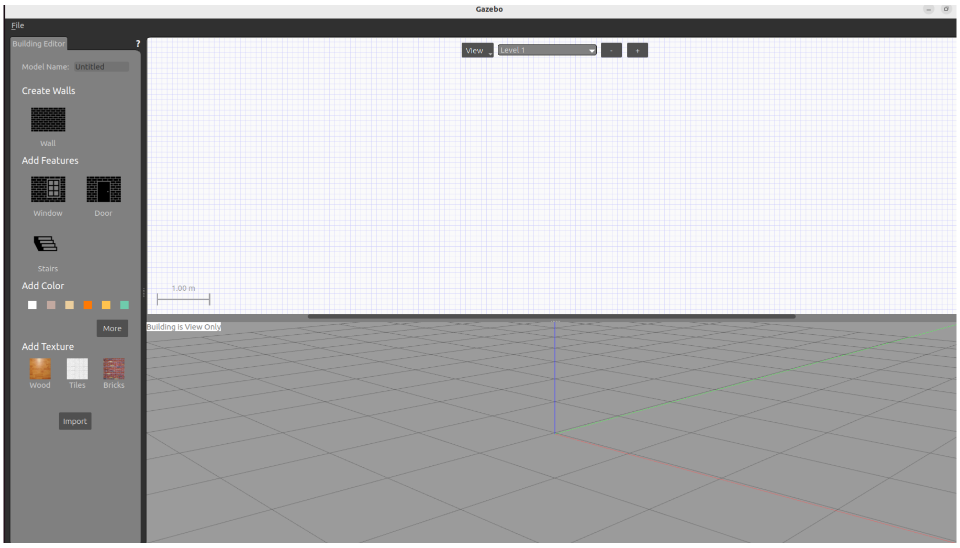


Figure 2.4: Gazebo building editor.

joints, and their relationships. Links represent physical components of the robot, such as bodies or end-effectors, while joints define the connections between links and specify their motion characteristics. You can also specify visual and collision properties of robot links. Visual elements define the appearance of the robot, such as meshes or geometries, while collision elements define simplified representations used for collision detection.
URDF can also define properties for simulator such as physical properties for the robot's links and joints, such as mass, inertia, and friction coefficients. This helps the physics based simulations in Gazebo, NVIDIA Isaac or even UNITY engine.
Final thing we can define in URDF file is sensors, such as cameras, lasers, or force/torque sensors, within the robot model. Sensor definitions allow simulation of sensor data generation and integration with control algorithms.

URDF is strict format that can be edited or generated, but cannot be changed parameticaly like XACRO. XACRO is an extension of URDF that allows for more modular and reusable robot descriptions. This can be thought of as using the same classes over again in programming with different parameters having different values. You can generate URDF files from XACRO files which can be then used for robot definition in simulation. XACRO even has its own python library for this purpose. Key features of XACRO is use of variables, which can store values or text strings. Variables provide a way to reuse values across different parts of the robot description, making it easier to maintain consistency and facilitate modifications. Xacro supports the inclusion of other Xacro files within a main Xacro file. This allows the composition of complex robot descriptions by modular-

izing different components and combining them in a single file. Includes promote code reusability and make it easier to manage large and complex robot models. This sort of nesting is useful on bigger projects that have lot of changing components from robotic arms, sensors or even types of mobility. Changes in mobility for robotic platform can be 2WD (two-wheeled drive), 4WD (4-wheeled drive) or legged propulsion, which provide mobility by locomotion.
Xacro files are pre-processed before generating the final URDF files. The Xacro preprocessor parses the Xacro syntax, resolves macros and variables, and outputs the resulting URDF file. This process simplifies the maintenance and generation of URDF files, especially for robots with repetitive structures or parameter variations. This can be done in CLI (Command-Line Interface) or in program using above mentioned XACRO library. Using Xacro alongside URDF enhances the readability, reusability, and modularity of robot descriptions, making it a popular choice within the ROS community.

When we have our desired description file written we can spawn the robot directly into running simulator with ROS2 command

```
ros2 run gazebo_ros spawn_entity.py -entity <robot_name> -file <
    path_to_robot_model> -x <x_position> -y <y_position> -z <z_position> -
    Y <yaw_angle>
```

This command works in gazebo simulator, but the structure for other simulators should be similar to this CLI script for gazebo. spawn_entity is ROS2 standard library package that spawns robots described by the URDF file into simulation. This can be used with namespaces and grouping in .launch to spawn homogeneous robotic swarm. [6] [7]

2.3 Robotic control

Control refers to the process of regulating the behavior and actions of a robot to achieve desired tasks and objectives. There are two primary approaches to robotic control: centralized control and decentralized control. Each approach has its own advantages and disadvantages, depending on the specific application and system requirements.

2.3.1 Comparison between centralised and decentralised approach

Centralized control, also known as monolithic control, involves having a single controller or decision-making entity that coordinates and governs the actions of the entire robot system. This central controller typically receives sensor data from various robot components, processes it, and generates control commands to drive the robot's actuators.

Advantages of Centralized Control:
Global System View: Comprehensive decision-making based on complete system information.
Optimal Coordination: Efficient coordination of robot components for synchronized actions.
Complex Task Execution: Execution of intricate tasks involving multiple robot components.

Disadvantages of Centralized Control:
Single Point of Failure: Controller malfunction can lead to system breakdown.
Communication Overhead: Data collection and command distribution may introduce latency.
Scalability: Challenges with large and distributed robotic systems.

Decentralized control distributes decision-making across multiple controllers or modules in the robot system.
Advantages of Decentralized control:
Modularity and Fault Isolation: Modular design enhances fault isolation. Be it physical defect or programming bug
Parallel Processing: Enables faster decision-making through concurrent operation.
Scalability: Easily accommodates new components and distributed controllers.

Disadvantages of Decentralized control:
Lack of Global View: May lack a complete system-wide view for complex tasks.
Inter-robot Communication: Effective coordination among robots is crucial.
Consistency and Synchronization: Maintaining coordination can be complex for multiple robots.

Choosing the Control Approach:
If this Thesis was not focused on Decentralised swarm control the choice between centralized control and decentralized control would depend on the specific requirements of the robotic system. Considerations such as system complexity, fault tolerance, scalability, and the level of coordination required for the tasks at hand play a significant role in determining the most suitable control approach.
It's worth noting that hybrid control approaches also exist, which combine elements of centralized and decentralized control to leverage the strengths of both approaches. These hybrid approaches aim to strike a balance between system-wide coordination and modular autonomy. Hybrid-control algorithms run into a problem of proper information bridging such as processing local data on either central system, local or both. As such they have the highest complexity of all control algorithms.

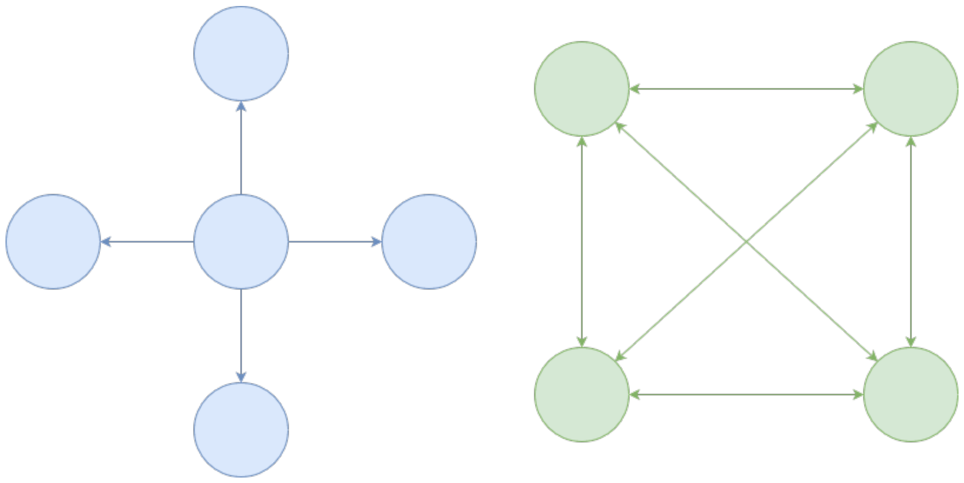


Figure 2.5: Diagram of centralised and decentralised control.

Ultimately, the control approach selected based on goals

2.3.2 Global and local goals

In robot control systems, the concepts of global and local goals are pivotal for shaping a robot's behavior and navigation strategy. Let's delineate the distinctions between these two approaches:
A global goal represents the ultimate destination or objective that the robot aims to achieve. It is typically defined in a coordinate system relative to the environment or a specific reference frame, most of the time a map. Global goals are usually set by an external agent, such as a human operator or a higher-level planning system. Examples of global goals include reaching a specific location, navigating to a target object, or following a predefined path.
When pursuing a global goal, the robot takes into account the overall task and plans its actions accordingly. It involves long-term decision-making and navigation strategies to reach the desired destination. Global goal planning often considers factors such as obstacle avoidance, path optimization, and global map information. The robot continuously updates its path based on the changing environment and optimizes its trajectory to achieve the global goal efficiently. Local Goal

A local goal, on the other hand, represents an intermediate or short-term objective within the robot's immediate vicinity. It is determined based on the current state of the robot and its surrounding environment. Local goals are typically computed in real-time using sensor feedback and local perception algorithms. They provide short-term targets that help the robot navigate and make decisions in its local

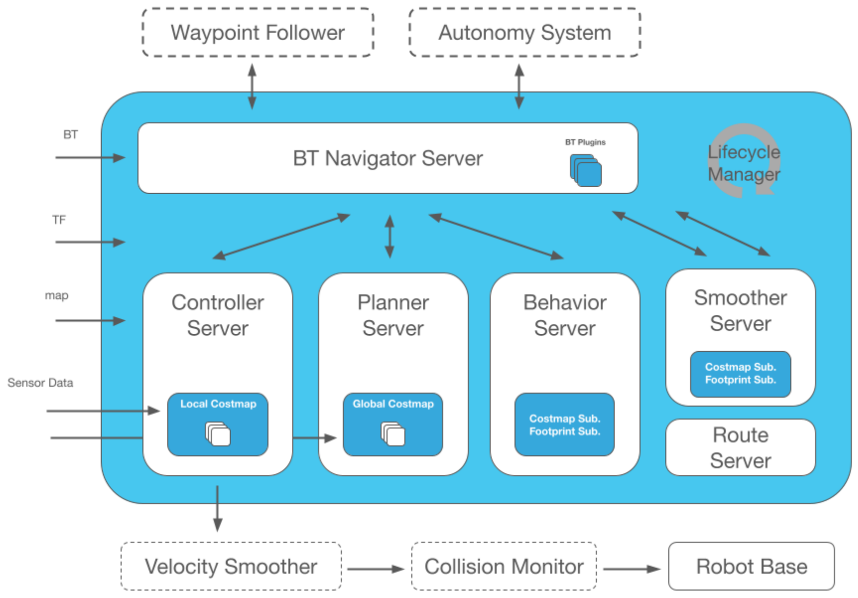


Figure 2.6: Navigation2 architecture showing global and local costmaps.

vicinity.
Local goals are often generated based on information such as obstacle detection, proximity to obstacles, or local landmarks. The robot uses these goals to plan immediate actions and adjust its trajectory to avoid obstacles or reach specific waypoints. Local goal planning focuses on short-term navigation, reactive behaviors, and obstacle avoidance to ensure safe and efficient movement in the immediate surroundings.
Integration and Importance
Global and local goals are not mutually exclusive but rather complementary in robot control systems. They work together to guide the robot towards achieving its ultimate objective while ensuring real-time adaptability to the immediate environment. The integration of global and local goals allows robots to navigate dynamically, respond to changing conditions, and handle unexpected obstacles or detours.
The global goal provides a high-level objective and guides the overall navigation strategy, while local goals help the robot react to local conditions and make immediate adjustments to its trajectory. By combining both approaches, the robot can navigate complex environments, handle dynamic obstacles, and efficiently reach its destination while maintaining local safety and adaptability.
In summary, the global goal sets the long-term objective for the robot, while local goals guide its short-term actions and adapt to immediate surroundings. The integration of both global and local goals enables robots to achieve efficient, safe, and adaptive navigation in a variety of environments and tasks. [8]

2.3.3 Navigation strategy

There are few strategies that can be used for Decentralised swarm robotics, such as Decentralized coordination:
is a strategy used in robotics and multi-agent systems where multiple individual agents or robots work together to achieve a common goal without relying on a centralized controller to make decisions for the entire group. Instead, each agent has some level of autonomy and makes local decisions based on its own perception of the environment and interactions with neighboring agents. This approach offers several advantages and is often inspired by principles observed in nature, such as in ant colonies, flocking birds, and social insect behavior.[9]

Decentralized coordination strategies can take various forms, including:

Swarm Intelligence inspired by behaviors observed in social insects like ants and bees, swarm robotics uses local interactions between robots to achieve group objectives. Algorithms like ant colony optimization and particle swarm optimization are commonly used for this purpose.

Distributed algorithms, such as consensus algorithms and distributed control, allow agents to reach agreements or make decisions collectively without relying on a central authority.

Multi-Agent Reinforcement Learning in complex environments, agents can use reinforcement learning techniques to independently learn and adapt their behaviors, with rewards based on the collective performance of the group.

Communication Protocols agents can communicate with each other through decentralized communication protocols, exchanging information about their state and intentions. These protocols enable cooperation without central control.

2.4 Solution

As can be seen from previous sections and subsections, there are many ways how to solve decentralised swarm simulation with different methods of controlling it. It is then important to limit focus of this thesis to one solution. It is also important to realise that there maybe issues with combining technologies for a novel solution As can be seen from previous text there are many ways to simulate robot swarm

and establishing communication between each of the robots or agents in the swarm. Therefore, it must be made clear in the early stages which setup can be used as it is limited by hardware and time that can be used on this thesis. To limit the scope of this thesis, it will be focused on decentralised swarm workbench of heterogeneous robots with connection to navigation stack that can perform navigation between PoI (Points of Interest). This will limit the hardware demand on the simulating desktop, so the limiting factor can be number of robots in the swarm instead of their complexity.
With the system determined and all the robots of the swarm being the same with same algorithm running on them with just different goals it should be possible to see the throughput of this made up manufacturing system.
For the swarm workbench to work properly, a setup of environment, maps of environment, robot models, their parametric files and such must be prepared for default operation. They should be interchangeable, but should be set up as a default workbench for any user to use. Robots should also have default algorithm with goals established beforehand so they can be used "out-of-the-box".

With these considerations in mind, the needed tasks wanted from the system can be explored. These tasks and delimitation are described in the following chapter.

3 - Requirements and Delimitations

As described in last section 2.4 a system must be developed to work as a robot swarm workbench with a navigational algorithm for all agents of the swarm. In order to determine whether this system can be used for a given task a set of requirements must be set. Also, because of the limited scope of this Thesis a set of delimitations must be set. These are described in this chapter.

3.1 Requirements

A satisfactory prototype workbench and algorithm must meet a certain set of predefined requirements. These requirements should be testable and measurable according to applicable metrics. They also should be comparable to their commercial counterparts, which are intended for a similar use case. The requirements of this Thesis can be conveniently divided into several parts, corresponding to different sub tasks, as seen below.

3.1.1 Workbench

Main component of the solution is working robot swarm workbench, a way to spawn several robots and allocate to them proper controls. As such, the following requirements should be met:

- Workbench should be able to spawn at least 20 robots at the same time.
- All robots running should have their namespace (Grouping of nodes) separated from every other robot to insure decentralisation.
- All robots should have their own RVIZ and nav2 stack running at the same time.
- The environment and robot models should be changeable.

3.1.2 Algorithm

To reach the goals set for the robots in the swarm, it is important that the algorithm satisfies the following requirements:

- Robots should reach their goal 100% of a time irrespective of a time it took to reach their goal.
- Robots should avoid static and dynamic obstacles 100% of the time. Collisions should not happen with either other robots or the environment.
- After receiving their goals, robots should move autonomously, without help from the user.

These requirements, are for an ideal version of the solution. The following section describes the delimitation of these requirements.

3.2 Delimitation

In section 2.4 a general solution has been given and as such, requirements could be drawn in section 3.1. Due, to resources and hours for the thesis being limited and do not allow for the setup of an ideal version of that generalised solution. Therefore, the scope of the thesis must be reduced according to the following criteria. This delimitation is based on the simulators, computers, time available, as well as the focus of the initial thesis proposal.

General scope

For scoping the thesis, most of the modules that work out of the box can be used to save time and focus on making a working swarm robotics package. Additionally, the system can have a already available models of robots. In the case of the implemented solution, these models are linorobot2 and turtlebot3 robots respectively. Development should be prioritise scalability in swarm size, but with all parts of solution working. The scope of this thesis is limited making one self-made algorithm for control of the swarm.

Hardware

For simulation a laptop will be used with generation of NVIDIA GPU older then 2018 models.

Robot Workbench

Workbench must be scale-able with number or robots it can spawn up to a limit, this limit can should not exceed 100 fully running robots. It also should be able to change most of the variables that user could want, such are .world file for environment and .sdf/.model/.urdf file for the robot.

Map

Map will be distributed centrally to every robot through the simulation, as if a central authority was giving a map (not fully decentralised)

Workstations

Workstations will be represented by points on the map and will not have their own model or forms of communication.

One plane limit

there will not be any multi-floor solutions for either spawning nor controlling robots

Algorithm

Will have a spatial limitation in a sense that at least 2 robots with can pass around each other without running into their inflation (safe) zones. Also there will be topology limitation in a sense that environment must have place where robots can take a holding patern while waiting for free space on workstations.

Randomly set goals

As this is not the focus of the thesis, workstation goal selection will be randomised for testing and will not be selected by any system.
This delimitation with the solution presented in 2.4 leads to a final problem formulation that guided the design and implementation of the rest of this thesis:

How can a Decentralised Robotic Swarm Simulation Workbench work with a given control algorithm?

4 - Design and Implementation

As described in the last chapter 3, the setup of the system must be designed according to this limited scope. This section will describe what different tools and packages were used in order to achieve functioning swarm workbench with control algorithm connected to every agent of the swarm.

4.1 Structure

As described in the introduction of this thesis, the robots will run on decentralised structure. As there is only one simulation server, there should be big difference in the load compared to how it would be in real life robot scenario. As in that scenario every robot would take on its own workload, but because this is the only point of computing and it has to simulate numerous robots running at the same time, all running their own nav2 stack, RVIZ and Simple_commander. This means this simulation has additive load for the simulation PC. Which means every additional robot simulated takes as much memory and processing time as the last one. this can be very noticeable with larger number of robots in the swarm. This structure can be seen in figure 4.1 and more detailed working version can be seen on 4.2.

4.2 Hardware and software

Simulating Gazebo is laptop with 32gb of RAM and NVIDIA 970m GPU. OS is Ubuntu 22.04 Jammy running ROS Humble.

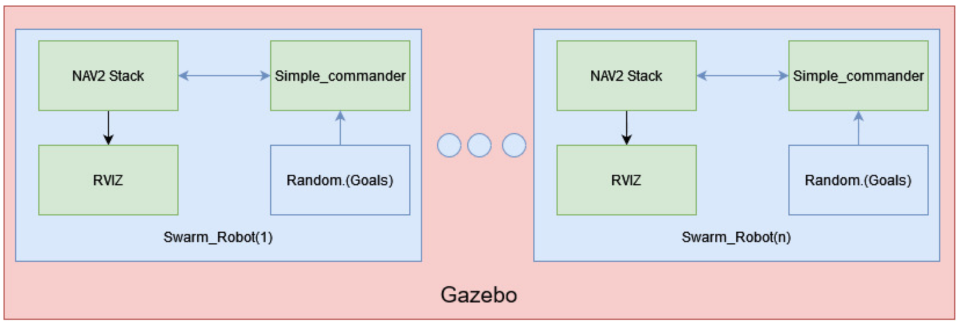
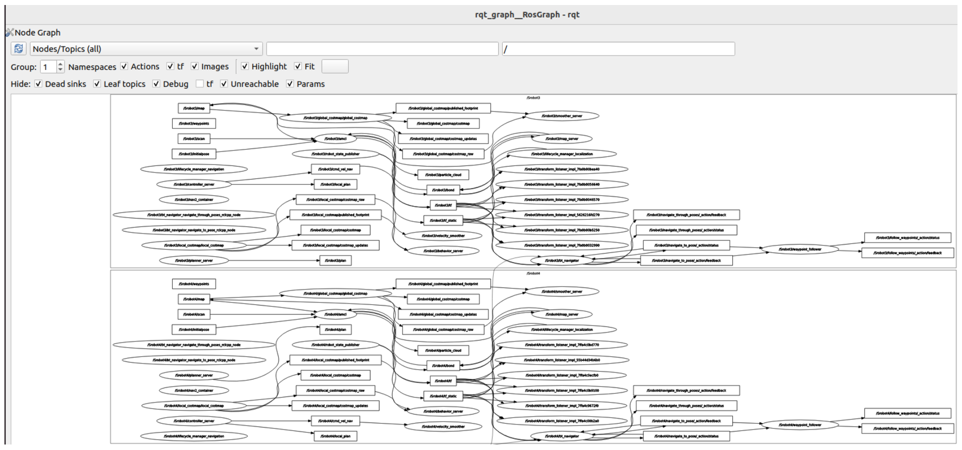


Figure 4.1: General schematic of the swarm workbench, color is to show different grouping of nodes.


rqt_graph__RosGraph - rqt

Node Graph

Nodes/Topics (all)

/

Group: 1 ✓ Namespaces ✓ Actions ✓ tf ✓ Images ✓ Highlight ✓ Fit

Hide: ✓ Dead sinks ✓ Leaf topics ✓ Debug ☐ tf ✓ Unreachable ✓ Params

Figure 4.2: Swarm Workbench seen by using RQT_graph.

ROS Humble is the latest LTS (Long term supported) version of ROS2 - is an open-source, flexible, and modular framework for building robot software. It is a significant advancement of the original ROS and is designed to meet the evolving needs of the robotics community. ROS2 provides a powerful set of tools and libraries for developing robotic applications and enables collaboration, consistency, and flexibility in the development process. It offers improved communication capabilities, including support for Publish/Subscribe and Request/Response communication patterns. This enables robust and efficient data exchange between different components of a robotic system.

4.3 Linorobot2 and Turtlebot3

Linorobot2 is an open-source project focused on creating a low-cost, educational, and versatile robotic platform that combines hardware and software components for robotic enthusiasts, hobbyists, and educators. The project emphasizes ease of use, affordability, and adaptability for various applications. And as such it found its way to AAU (Aalborg University), where it serves as test bench for student projects. This thesis will only be using .URDF and .Xacro files of the project along with its navigation packages.
TurtleBot3 is another open-source robot platform, but it's designed for a different set of use cases compared to Linorobot2. This thesis will be using its .URDF and .sdf files for simulating this robot, along with its nav2 implementation of multi-robot spawning.

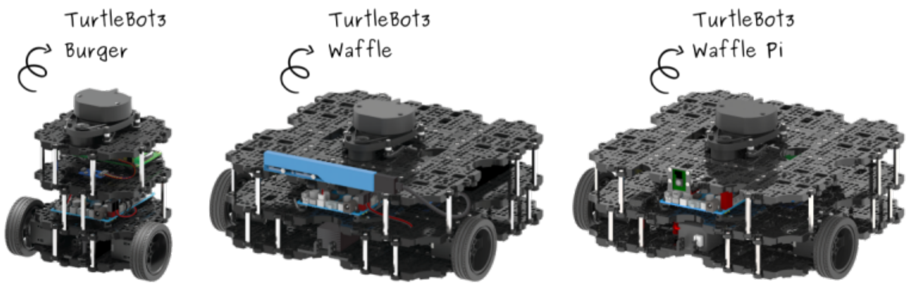


Figure 4.3: Turtlebot3 series of robots used in this project as models alongside linorobots2[10].

4.4 Multi-robot simulation

Goal of running multiple robots in one simulation is to have all their transforms, sensors and nodes separated by namespaces which can be seen in 4.2. Available .launch files in linorobot2 and turtlebot3 are not made for handling more than 2 robots at a time in simulation and thus have to be modified. Not just their robot spawning behaviors, but also their node allocation methods have to be changed.

4.5 Changing the launch files

there are 2 launch files spawning multiple robots in this project and both have different approaches. One is from box-bot project, minimal simulation project, that was changed to suit linorobot2 and given number of robots from terminal.
Second aproach was done with Turtlebot3. This aprroach was taken from nav2_bringup multi_tb3_launch.py. As both of these approaches were found lacking their structure had to change, for linorobot2 it was with grouping namespaces for each robot and all the nodes that belonged to that particular robot. this approach was problematic with spawning of RVIZ as the parameter file was not defined for multiple robots and had to be remade. Linorobot2 also has a separate file for launching nav2 stack which gives it bigger capacity for number of robots, up to 100.
Turtlebot3 compared to linorobot2 had multirobot support before, but only for 2. YAML file generator for parameter files had to be added to launch file along with dynamic creation of desired number of robots. Turtlebot3 has automatic nav2 and RVIZ startup, but as a result can handle much less robots at the same time, up to 15 only, but this is limitation of the hardware only.

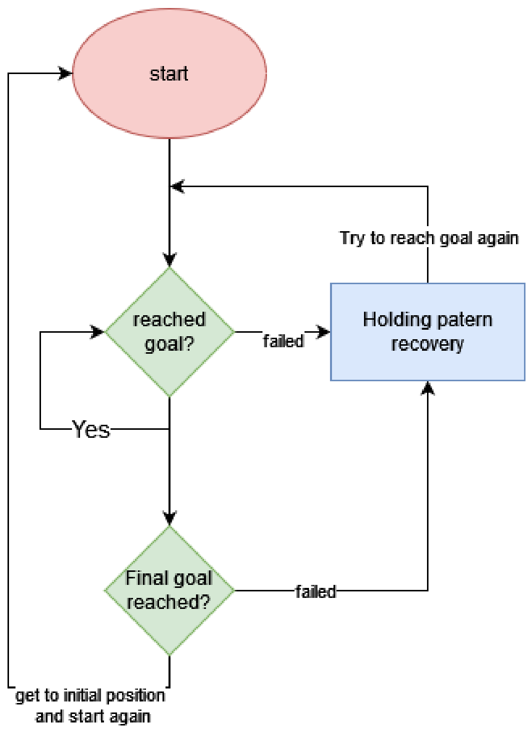
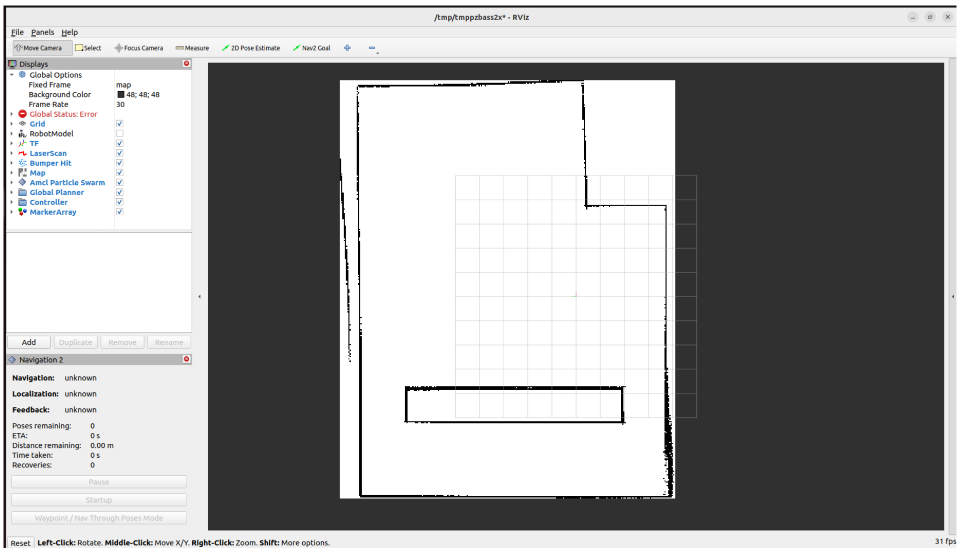


Figure 4.4: Recovery tactic for when goal position is occupied.

4.6 Navigation and Simple Commander

Navigation2 node is started up either at the same time as spawning or joined into the namespace from separate launch file. This does not mean that th nav2 stack is activated. It is only activated if autostart value was set to true, or it was start-upped in RVIZ. After nav2 node is started up, Simple commander script can be run. Problem is that this has to be done in separate terminals or run on different threads as each BasicNavigator() function is terminal blocking and wont allow another BasicNavigator() function to run. This means that in the latest version of the project, simple commander scripts have to be manually started up in each terminal.
In the Simple commander scripts there is space to make or call your own goals and should be changed for each environment. The Simple commander API even handles the holding pattern portion of the script, because when goal fails thanks to another robot already occupying the position, it start up the holding pattern and tries to reach that goal again.



Global Options	
Fixed Frame	map
Background Color	■ 48; 48; 48
Frame Rate	30
Global Status: Error	
Grid	✓
RobotModel	☐
TF	✓
LaserScan	✓
Bumper Hit	✓
Map	✓
Amcl Particle Swarm	✓
Global Planner	✓
Controller	✓
MarkerArray	✓

Navigation:	unknown
Localization:	unknown
Feedback:	unknown
Poses remaining:	0
ETA:	0 s
Distance remaining:	0.00 m
Time taken:	0 s
Recoveries:	0

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:** Zoom. **Shift:** More options.

Figure 4.5: RVIZ with navigation stack started up, without Pose estimate.

5 - Testing and Creation guide

To test all the systems and modules put into the same .launch file a series of test were done. After these test there will be guide how to create your own environment for swarm control, along with adding your own models.

5.1 Tests

First test was with all default setting, which are configured as such. World file for gazebo environment is slim_blockage.world, urdf and model is set to turtlebot3_waffle, number of robots is 5 and their names are Srobot1 to Srobot5 (Swarm robot), navigation stack is autostarted along side with RVIZ as seen in figure 5.1.
 this test was performed to show how default behavior works. On given hardware is after start of the launch script, all RVIZ consoles open and after them gazebo with all spawned robots.

5.1.1 Spawning more than default

Spawning more than default number of robots yields a considerable wait time for gazebo simulation to actually spawn the robot entity. This can be seen in figure 5.2. There is no given order in which the robots spawn as they are all assigned to different threads on CPU and it just depends on the core load at the time, when it come to spawn entity execution
 after a minute even 10 robots are no problem spawning with autostarted nav2 and RVIZ as can be seen in figure 5.3 . without them it is even possible to spawn up to a 100 robots, it is in coded limit as my hardware stopped temporarily working after trying 200 and shutting down from overheating. This coded in limitation can allways be coded out, but it should stay as safety.

5.1.2 Using Simple_commander

moving the robots through the poses can be done either from RVIZ manualy, or throught simple commander API script automaticaly, with holding pattern algorithm. this can be seen both in RVIZ and gazebo in figure 5.4. Simple_commander can only start when nav2 stack is active already as can be seen in RQT_graph in figure 5.5. It initialises the position thanks to recreating the spawning sequence for x and y positions of given namespace of the swarm robot. then it just gives the

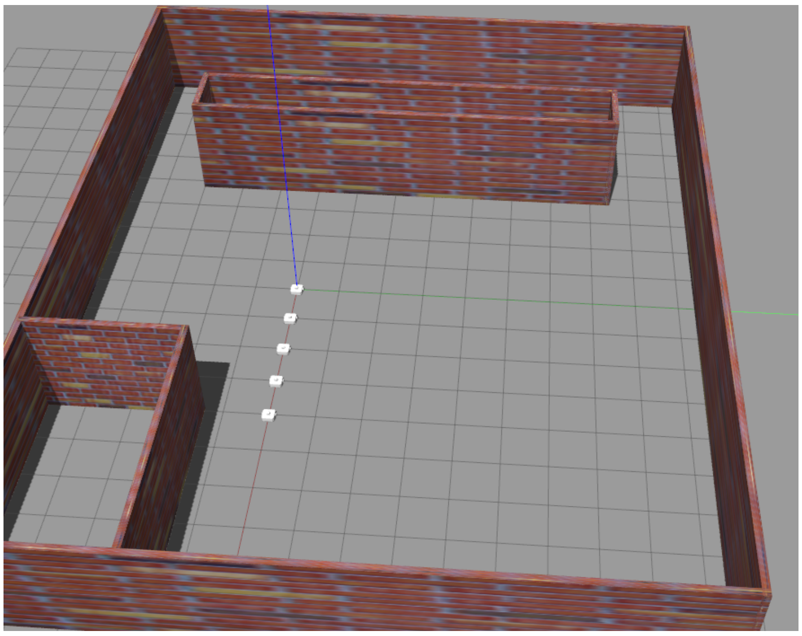
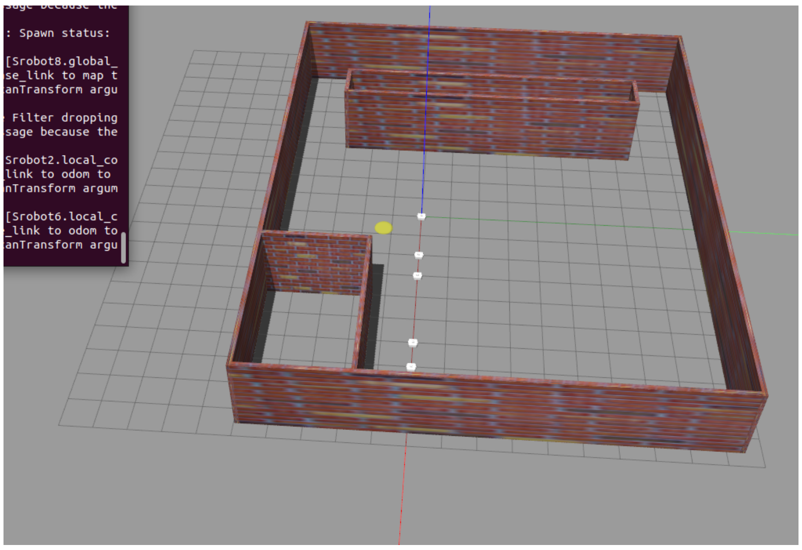


Figure 5.1: Default spawning method.



```
sage because the

: Spawn status:

[Srobot8.global_
se_link to map t
anTransform argu

 Filter dropping
sage because the

Srobot2.local_co
link to odom to
nTransform argum

[Srobot6.local_c
_link to odom to
anTransform argu
```

Figure 5.2: Problem with spawning more robots than simulation can handle on start-up.

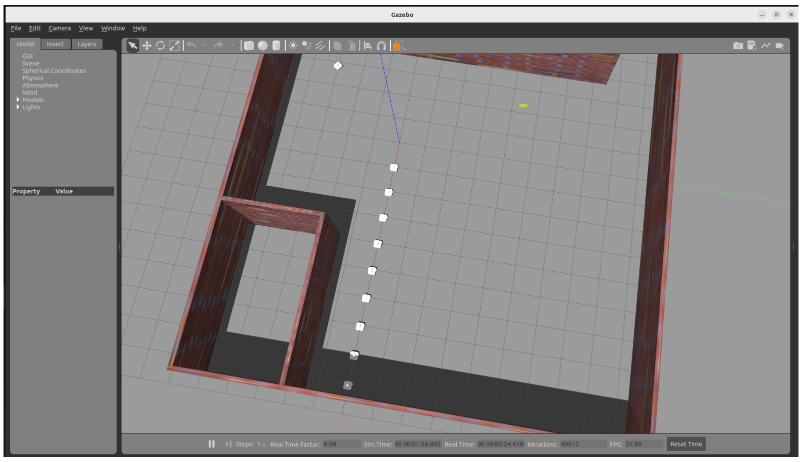
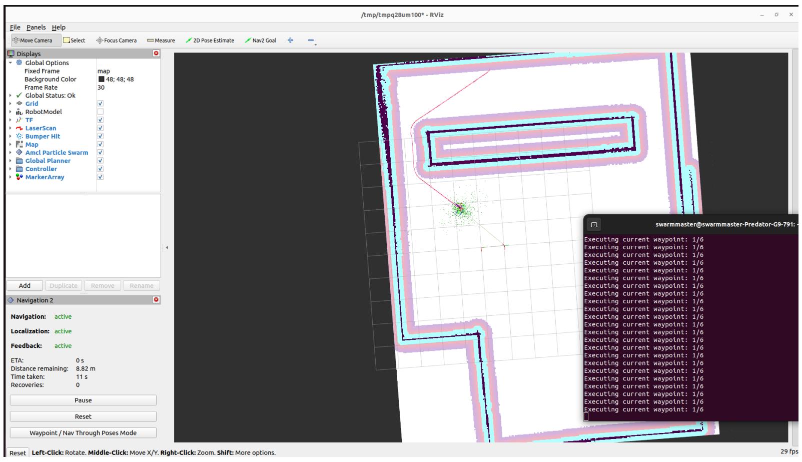


Figure 5.3: After re-running spawn entity automatically by the launch file, spawning more than default number of robots was complete.



```
Displays
Global Options
  Fixed Frame          map
  Background Color     48; 48; 48
  Frame Rate           30
Global Status: Ok
Grid                   ✓
RobotModel
TF                     ✓
LaserScan              ✓
Bumper Hit             ✓
Map                    ✓
Amcl Particle Swarm    ✓
Global Planner         ✓
Controller             ✓
MarkerArray            ✓

Navigation 2
Navigation:    active
Localization:  active
Feedback:      active

ETA:                 0 s
Distance remaining:  8.82 m
Time taken:          11 s
Recoveries:          0

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click: Zoom. Shift: More options.
```

```
swarmmaster@swarmmaster-Predator-G9-791:
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
Executing current waypoint: 1/6
```

Figure 5.4: Simple commander can be seen running in RVIZ and gazebo in this figure.

goal position in randomised order. This should be also doable by calling a service in the future for server given goal orders.

5.2 Creation guide

This workspace is build in ROS2 Humble, it is recommended to have full desktop installation on Ubuntu 22.04 with turtlebot3 and gazebo_ros packages. In order to create anything in swarm_workbench, you have to download the workspace with command shown bellow.

```
mkdir swarm_workbench
cd swarm_workbench
```

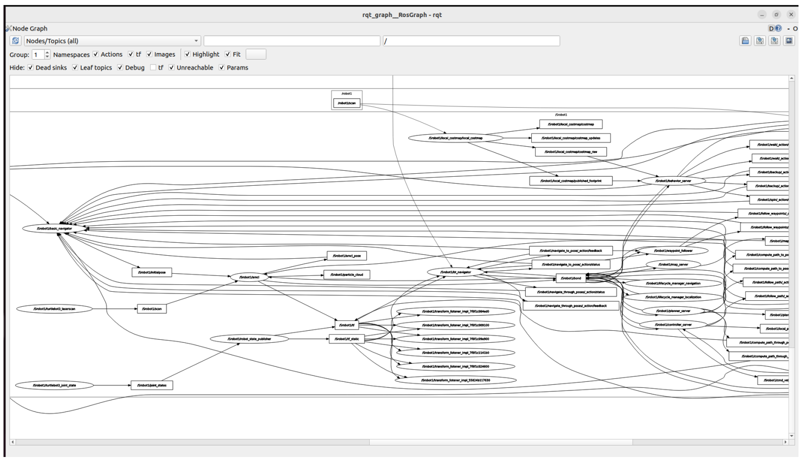

rqt_graph__RosGraph - rqt
Node Graph
Nodes/Topics (all)
Group: 1 Namespaces ✓ Actions ✓ tf ✓ Images ✓ Highlight ✓ Fit
Hide: ✓ Dead sinks ✓ Leaf topics ✓ Debug tf ✓ Unreachable ✓ Params

Figure 5.5: RQT_graph of simple commander API connections to navigation stack.

```
git clone https://github.com/TheMarksniper/swarm_workbench.git
```

after cloning the repository you will want to build with this command

```
colcon build --merge-install
```

if any problems arise there is small helper_commands.sh file that should have all the troubleshooting commands. They should be continuously added with more features. After successfully building with colcon you will have to source your new workspace

```
source install/setup.bash
```

and after that you can finally launch the multi-robot launcher. the default launch is like this.

```
ros2 launch swarm_gazebo new_multi_launch.py
```

if you want to change the number of robots, you can with added argument

```
ros2 launch swarm_gazebo new_multi_launch.py swarm_size:=10
```

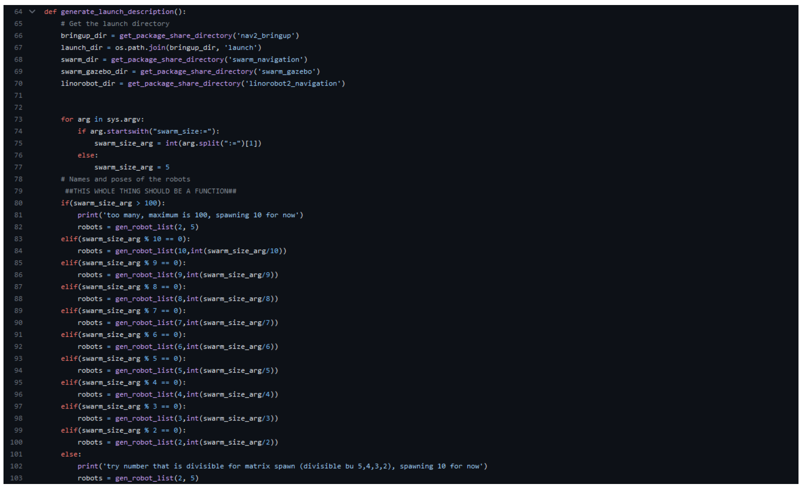
after the desired number of robots are spawned in gazebo and all RVIZ windows are working its time to run the python script files

```
python3 src/swarm/swarm_navigation/scripts/select_route.py Srobot1
```

this will launch the simple commander API script with randomised route selected for given robot. You will have to open a new terminal to run more than one simple commander script. Limitation as always should be just speed of your hardware.

5.2.1 How to change these default settings?

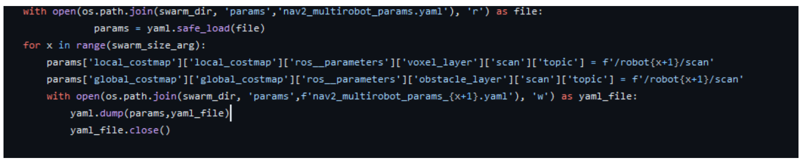
changing the default world can be found in



```python
def generate_launch_description():
    # Get the launch directory
    bringup_dir = get_package_share_directory('nav2_bringup')
    launch_dir = os.path.join(bringup_dir, 'launch')
    swarm_dir = get_package_share_directory('swarm_navigation')
    swarm_gazebo_dir = get_package_share_directory('swarm_gazebo')
    linorobot_dir = get_package_share_directory('linorobot2_navigation')


    for arg in sys.argv:
        if arg.startswith("swarm_size:="):
            swarm_size_arg = int(arg.split(":=")[1])
        else:
            swarm_size_arg = 5
    # Names and poses of the robots
     ##THIS WHOLE THING SHOULD BE A FUNCTION##
    if(swarm_size_arg > 100):
        print('too many, maximum is 100, spawning 10 for now')
        robots = gen_robot_list(2, 5)
    elif(swarm_size_arg % 10 == 0):
        robots = gen_robot_list(10,int(swarm_size_arg/10))
    elif(swarm_size_arg % 9 == 0):
        robots = gen_robot_list(9,int(swarm_size_arg/9))
    elif(swarm_size_arg % 8 == 0):
        robots = gen_robot_list(8,int(swarm_size_arg/8))
    elif(swarm_size_arg % 7 == 0):
        robots = gen_robot_list(7,int(swarm_size_arg/7))
    elif(swarm_size_arg % 6 == 0):
        robots = gen_robot_list(6,int(swarm_size_arg/6))
    elif(swarm_size_arg % 5 == 0):
        robots = gen_robot_list(5,int(swarm_size_arg/5))
    elif(swarm_size_arg % 4 == 0):
        robots = gen_robot_list(4,int(swarm_size_arg/4))
    elif(swarm_size_arg % 3 == 0):
        robots = gen_robot_list(3,int(swarm_size_arg/3))
    elif(swarm_size_arg % 2 == 0):
        robots = gen_robot_list(2,int(swarm_size_arg/2))
    else:
        print('try number that is divisible for matrix spawn (divisible bu 5,4,3,2), spawning 10 for now')
        robots = gen_robot_list(2, 5)
```

Figure 5.6: Code that generates positions for robots to spawn in.



```python
with open(os.path.join(swarm_dir, 'params','nav2_multirobot_params.yaml'), 'r') as file:
            params = yaml.safe_load(file)
for x in range(swarm_size_arg):
    params['local_costmap']['local_costmap']['ros__parameters']['voxel_layer']['scan']['topic'] = f'/robot{x+1}/scan'
    params['global_costmap']['global_costmap']['ros__parameters']['obstacle_layer']['scan']['topic'] = f'/robot{x+1}/scan'
    with open(os.path.join(swarm_dir, 'params',f'nav2_multirobot_params_{x+1}.yaml'), 'w') as yaml_file:
        yaml.dump(params,yaml_file)
        yaml_file.close()
```

Figure 5.7: Code that generates parametric files specific to each robot namespace.

```
nano src/swarm/swarm_gazebo/new_multi_launch.py
```

file under declare_world_cmd argument, of course you can always be calling the world:=your_path/world.world in the launch arguments, same with map:= file (declare_map_yaml_cmd).
Bigger problems happens if you want to change model of the simulated robot, as you have to change not only URDF file, but also sdf file and param file for RVIZ. It isnt a problem that cannot be solved with changing few values. you can see how to change behaviours of how robots spawn in figure 5.6 and how to make temporary yaml files in figure 5.7

6 - Discussion

6.1 Testing number of robots

This section compares the performance of the different tests described in the previous chapter. And the tests clearly show that a lot of computing power is required for simulation even of a few running robots in minimalist environment. In the test it was shown that number of robots spawn is not limited by ROS or gazebo, but pure computational strength.
testing was only done on Turtlebot3 as there was a problem with collisions on Linorobots2, when running simple commander API they had too little inflation for collisions and when they ran into each other they could not recover.

6.2 Simple commander

Simple commander API is extremely good tool to use in any application that already uses navigation stack. you can not only go toPose(), but also goThroughPoses() with many more functionalities that are not explored in this thesis, but will be in the future of this package.

7 - Conclusion

This Master thesis was driven purely by my interest in swarm robotics and as such is limited in its scope and performance. Still I would like to release the code and models as open source project in decentralised swarm robotics as it may one day in future help with other projects in this or other fields.

This Master thesis describes Decentralised robotic swarm simulation, as well as the performance of this setup. Chapter 5 shows the performance of the system and while dynamically spawning number of robots, as well as connecting them simultaneously to navigation stack and RVIZ. This can be seen taken further by adding simple commander API script and navigating with multiple robots at the same time. There is also a small guide for anyone who would want to contribute to the project with their own creation.

This project is something that is not yet done in ROS2 as the only indirect competitor is completely centralised system in Open-RMF, which is a professional project that completely neglects the decentralised approach. In the future this could be either addition to that project or become its own spin-off in the ROS2 ecosystem, same as simple commander was. This project deserves even more features and additions to it.

In conclusion to answer to the main problem stated in chapter 3. Yes a Decentralised robotic swarm simulation can work with control algorithm that is tapered to its needs and works well and can follow all goal without stopping.

Bibliography

[1] Hanns Huber. *BMW Group is making logistics robots faster and smarter*. www.press.bmwgroup.com, May 2020. URL: `https://www.press.bmwgroup.com/global/article/detail/T0308393EN/bmw-group-is-making-logistics-robots-faster-and-smarter?language=en`.

[2] Open Robotics. *Gazebo*. gazebosim.org. URL: `https://gazebosim.org/` (visited on 05/20/2023).

[3] NVIDIA. *Isaac ROS*. NVIDIA Developer, Oct. 2021. URL: `https://developer.nvidia.com/isaac-ros` (visited on 05/20/2023).

[4] Cameron Greene, Michael Pinol, Amanda Trang, and Jacob Platin. *Robotics Simulation in Unity*. Unity Blog, Nov. 2020. URL: `https://blog.unity.com/engine-platform/robotics-simulation-is-easy-as-1-2-3` (visited on 05/01/2023).

[5] Steve Crowe. *Unity Showcases ROS 2 Support with New AMR Demo*. The Robot Report, Aug. 2021. URL: `https://www.therobotreport.com/unity-showcases-ros-2-support-new-amr-demo/` (visited on 10/23/2023).

[6] Open-Robotics. *Xacro - ROS Wiki*. wiki.ros.org, Mar. 2022. URL: `http://wiki.ros.org/xacro`.

[7] Open-Robotics. *URDF- ROS Wiki*. wiki.ros.org, Mar. 2023. URL: `http://wiki.ros.org/urdf`.

[8] Open Navigation LLC. *Nav2 — Navigation 2 1.0.0 documentation*. navigation.ros.org, 2023. URL: `https://navigation.ros.org/`.

[9] David St-Onge, Vivek Shankar Varadharajan, Ivan Švogor, and Giovanni Beltrame. "From Design to Deployment: Decentralized Coordination of Heterogeneous Robotic Teams". In: *Frontiers in Robotics and AI* 7 (May 2020). DOI: `10.3389/frobt.2020.00051`. (Visited on 09/01/2023).

[10] Open-robotics. *TurtleBot3*. robots.ros.org. URL: `https://robots.ros.org/turtlebot3/`.