Cleaning Strategy for Service Robots in Presence of a Mobile Robot Fleet

MASTER THESIS

Meike Blöcher

Submitted on 1 June 2023

Friedrich-Alexander-Universität Erlangen-Nürnberg
Faculty of Engineering, Department Computer Science
Professorship for Open Source Software

Supervisor:
Prof. Dr. Dirk Riehle, M.B.A.

Friedrich-Alexander-Universität
Faculty of Engineering

Declaration of Originality

I confirm that the submitted thesis is original work and was written by me without further assistance. Appropriate credit has been given where reference has been made to the work of others. The thesis was not examined before, nor has it been published. The submitted electronic version of the thesis matches the printed version.

Erlangen, 1 June 2023

License

This work is licensed under the Creative Commons Attribution 4.0 International license (CC BY 4.0), see https://creativecommons.org/licenses/by/4.0/

Erlangen, 1 June 2023

Abstract

This thesis investigates obstacle avoidance strategies for cleaning robots in industrial environments with a present Automated Guided Vehicle (AGV) fleet and presents a comparative evaluation of these strategies. The study aims to enable safe and efficient navigation for cleaning robots by addressing the challenge of obstacle avoidance. A comprehensive system architecture is proposed based on a thorough literature review of obstacle avoidance techniques. The system is implemented and tested with Robot Operating System (ROS) and the simulation environment Gazebo. The implemented obstacle avoidance strategies are evaluated using three comparable metrics: cleaning time, reached coverage, and multi-coverage. The best-performing strategies depending on the user's preferences are the follow-reorder and the replan strategy. Overall, this research provides valuable insights into obstacle avoidance strategies for cleaning robots in industrial environments.

Contents

List of Figures

List of Tables

Acronyms

ACD Approximate Cellular Decomposition

AGV Automated Guided Vehicle

CCPP Complete Coverage Path Planning

CPC Collision Point Calculation

CPP Coverage Path Planning

DATMO Detection and Tracking of Moving Objects

OD Obstacle Detection

ROS Robot Operating System

SLAM Simultaneous Localisation and Mapping

TSP Traveling Salesman Problem

1 Introduction

The development of robots has revolutionised the way we live and work. In recent years, robots have been used in various industries to increase efficiency and productivity. One area where robots have made significant progress is in cleaning services. Cleaning robots have become progressively more popular in recent years due to their ability to automate the cleaning process and improve efficiency with minimal human intervention. Manual cleaning can be risky, exposing human operators to hazardous materials or environments.

This work's focus lies on cleaning robots in an environment of a working Automated Guided Vehicle (AGV) fleet. AGV fleets are becoming increasingly common in warehousing, logistics, and manufacturing industries. These vehicles transport goods, materials, and equipment within a facility, and with a tight schedule in transportation, the fleet should be disturbed minimally. With the increasing popularity of AGV fleets, automation in such an environment can provide significant benefits. These benefits include the uninterrupted working time of the fleet and human safety.

The specific use case is as follows:
A warehouse with a working AGV fleet, not being further specified, should be cleaned as efficiently as possible, while not interfering with the movement of the AGV fleet. This means the AGVs should not stop moving or change their paths for the robot to be able to clean the area. The cleaning solution provides multiple strategies for avoiding collisions with the AGV fleet, and the results achieved by the different strategies will be discussed. The primary objective of this research is the design and development of these strategies for a cleaning robot in a dynamic industrial environment in the presence of a separate mobile robot fleet to create a comprehensive cleaning solution.

The remainder of this thesis is organised as follows. A short overview of the related work is given in Chapter 2. In Chapter 3 the requirements necessary for the successful implementation of the strategies are presented. Chapter 4 details a comprehensive overview of the systems architecture. In Chapter 5, the system's design elements and implementation details are thoroughly discussed. Continuing with Chapter 6, the evaluation of the proposed requirements is executed and the strategies' performance results are discussed. Finally, in Chapter 7 a conclusion and outlook is given.

For better readability, the following work will display implemented nodes and modules by `typewriter font`.

2 Literature Review

One crucial part of robotic navigation is path planning, which describes the planning of a suitable route for the robot to drive. Typically the shortest path is desired. Yet for Coverage Path Planning (CPP), the path needs to be different. In this case, the task can be described as a specification of the Traveling Salesman Problem (TSP) (Bellmore and Nemhauser, 1968), in which the salesman has to visit every travel location given in an area with the shortest possible route. In the cleaning task, the travel locations are not points of interest but the entirety of locations in the given environment and the shortest, most efficient route is still desirable. The solution for this problem is given by Complete Coverage Path Planning (CCPP) algorithms.

Additionally, within the cleaning task in an industrial environment other requirements, such as collision avoidance with traffic from other vehicles or human workers also have to be met. This topic is very well-researched and numerous approaches have been developed of which only the most relevant algorithms are described in this work. Additionally, complete solutions for cleaning robots in various environments have been developed. A couple of interesting ones are also described in this chapter. First, this chapter introduces CCPP algorithms in Section 2.1, then some background on already existing cleaning strategy solutions and collision avoidance approaches is given in Section 2.2.

2.1 CCPP

In the field of CCPP, many algorithms have been developed (Jan et al., 2014; Luo and Yang, 2008; Oh et al., 2004; Viet et al., 2013; Yang and Luo, 2004). To give a good overview, three surveys are studied and the most interesting mentioned algorithms are explained. The first survey from Choset, 2001 summarises the different algorithms proposed until 2001 and categorises the algorithms into heuristic and cell decomposition approaches. These heuristic algorithms select random driving directions and continue to drive there until an object is detected. This behaviour is not mathematically complete and the resulting error would significantly impact large areas, like the one given in this research. Therefore, heuristic approaches are not relevant to this work. Also, time and energy are limiting factors in industrial settings, and with heuristic approaches, an optimisation in this regard is not possible. This leads the focus for the CCPP algorithm search to the cell decomposition approaches. The cellular decomposition approaches decompose the desired area into cells. This class of algorithms can be further distinguished into approximative and exact cellular decompositions.

In the Approximate Cellular Decomposition (ACD) algorithms, all cells are of the same size and once every cell is covered the area is considered clean. One example of ACD algorithms is the wavefront algorithm by Zelinsky et al., 1993. For this approach, a goal point needs to be set and the distance of each cell to the goal is calculated by propagating a wavefront. A path can be planned with the obtained distances. For simple navigation, the shortest path is chosen by following the cells with the smallest distance to the goal point. In complete coverage, the robot first visits all cells with the same furthest distance to the goal and only moves closer to the endpoint when that's the furthest point away. This solution produces paths with too many turns so an optimisation was introduced. This optimisation does not only calculate the distance to the goal but also takes the distance to obstacles into account. Another example is the spanning tree algorithm by Gabriely and Rimon, 2001. A further distinction of ACD approaches is Semi-ACD algorithms, which only discretise space partially. For example in Lumelsky et al., 1990 cells only have a fixed width and the planning is also done in a zig-zag manner.

In exact cellular decompositions, cells are no longer uniform but nonintersecting regions, which make up the whole environment. An example is the trapezoidal decomposition by Latombe, 2012, which parts the environment into trapezoidal regions that can be swept in a back-and-forth motion. These regions are ordered in a connectivity graph and a path is planned with this information. An improvement of this algorithm is the Boustrophedon algorithm developed by Choset and Pignon, 1998. In this decomposition algorithm, the cells are bigger and decomposed through connectivity information. Just as in the trapezoidal decomposition the boustrophedon also parts the region into cells that together cover the entire area. The cells are again ordered in an adjacency graph. In this approach, however, the environment does not have to be discretised and the cells can have different shapes. Additionally, the algorithm connects cells that can be cleaned in one motion, as fewer cells are better for the efficiency of the execution.

According to Choset, CCPP algorithms can also be classified into offline and online. The environment has to be known prior to planning for methods to be classified offline. Therefore, dynamic changes in the environment can not be integrated into the initial planning. Online algorithms on the other hand don't acquire full information about the surroundings beforehand but use sensors for information gathering while planning. The scenario in this work allows a combination of online and offline planning. As a map is most likely given through the present Automated Guided Vehicle (AGV) fleet manager and the environment itself does not change it could be classified offline. But with the AGVs moving, the cleaning robot needs a sensor for detecting the obstacles and adjusting the path. As online algorithms are more complex and need to get to know the environment first, the arising question is whether an offline algorithm is more efficient than an online algorithm by using the given information. In general, the usage of given information should have better results than not taking it into account.

In another survey by Galceran and Carreras, 2013, the most successful CCPP algorithms until 2013 are compared. They mention cellular decomposition methods as well as grid and graph-based algorithms. Apart from already introduced algorithms trapezoidal and boustrophedon, Galceran and Carreras also address the morse-based cellular decomposition, which was introduced by Acar et al., 2002. In the Morse-based approach, a decomposition similar to the boustrophedon decomposition is proposed. The area is sliced by a predefined method, e.g. circles or in a star pattern, and in the slices, critical points are searched by deriving the functions describing the connectivity in the given cells. These critical points are checked for degenerability. If they are nondegenerable, they are viewed as critical points and included in the path planning algorithm that is similar to a potential field algorithm, where these critical points attract or repel the cleaning robot.

The third survey by Bormann et al., 2018 compares the performance of six CPP algorithms, some already described by Galceran and Carreras, 2013 and Choset, 2001. These algorithms are classified as CPP and therefore are not focussing on reaching every point. Some of them may not be as relevant but most of them can be used for CCPP as well. The first algorithm they chooe is the classic Boustrophedon CCPP as already described earlier. For the evaluation with the other algorithms the optimised version by Huang, 2001 is implemented.

The second algorithm is the Grid-Based TSP CPP, where the area is parted into regular-shaped cells at all accessible locations. Each cell needs to be visited and the resulting path is obtained, by searching for the shortest travelling path either with an exact solution by the TSP solver Concorde, an approximate solution by a genetic algorithm or a nearest neighbour search.

The third approach is a Neural Network based algorithm developed by Yang and Luo, 2004, which solves the CCPP problem in a varying environment. The neurons in the network describe the activity landscape of the environment and represent one cell of the grid each. The cleaning robot is then attracted by not yet cleaned cells and repelled by obstacles.

The fourth algorithm is the grid-based local energy minimisation introduced by Bormann et al., 2015. In their approach, the grid is shaped the same approximate way as in the previous two algorithms. The visited cells are remembered and a new neighbour cell is found with an energy function. With the proposed method the cleaning robot starts in a corner and prefers to drive straight rather than turning which results in track-following-like behaviour.

Contour line-based CPP is the only approach that works on the original map without discretisation and finds local minima in the Voronoi graph generated for this map. Between these critical points and the closest point to the obstacle contour lines are chosen with respect to the robot's size.

The last algorithm is the convex sensor placement CPP which is based on the art gallery problem and does not result in complete coverage. So this algorithm is not relevant to this work.

Bormann et al., 2018 shows the differences in the planned paths as well as an evaluation of the algorithms concerning the metrics computation time, path length, number of rotations, travelling time, and coverage percentage for the different algorithms. Their work results in the Boustrophedon algorithm as the overall best appeal with minimal travelling time and rotation as well as one of the highest coverage percentages.

The proposed algorithms could all be used for the stated research problem. Yet to be able to focus on the main objective to develop and evaluate obstacle avoidance strategies the simplest algorithm is implemented; the trapezoidal decomposition.

2.2 Existing Strategies

A cleaning solution in an industrial environment with a working AGV fleet does not only need a planning algorithm. Reacting to the environment is just as important. As the subject has been researched extensively the given setting is important for the choice of a fitting strategy. For example, if the setting is known and static, a simple planning algorithm is sufficient but if the environment is dynamic like in a warehouse, where palettes can be stored in any position the environment changes and brings more challenges.

A solution provided by Yan et al., 2020 is the FLOBOT which is designed to clean highly visited large public areas like supermarkets or airports. Yan et al., 2020 developed a human detection and tracking module, a method to detect dirty areas and use the information gathered for further improvements of the navigation. The robot should clean the floor during the opening hours. This scenario is interesting in the approach to handling collisions. Yet in public places, human safety has a higher priority such that the cleaning robot just needs to wait as humans are more flexible in circumnavigating the cleaning robot than other robots. In this research scenario, the AGVs also have priority, as they might have a restricted driving area or the overall task of the fleets vehicles is more time relevant. However, they follow specific paths and therefore can be navigated around more easily. Additionally, the necessary safety precautions are not as strict as for human interference. Whereas the shoppers in a supermarket roam around without specified paths and change direction easily.

An approach for a mowing robot with mobile obstacle avoidance is introduced by Hsu and Lin, 2014. They focus on efficiency in the planning algorithm and introduce three modes, one with minimal time and one with efficient energy consumption. The third mode is mixed. Their approach for obstacle avoidance is based on a potential field method and assigning avoidance difficulty values to the obstacles. The mowing robot, however, does not return to the avoided sequence, which does not guarantee full coverage especially not on the route of the moving obstacle.

Other examples of collision avoidance come from the area of robot navigation in general. The most interesting part for the adaption into a cleaning strategy is the dynamic window approach. This algorithm described by Fox et al., 1997 searches for paths directly in the velocity space. All paths that are reachable with the mobility constraints given for the cleaning robot and which are safe regarding obstacles are considered. Then the option which maximises an objective function describing the goal is chosen. The approach of Seder and Petrovic, 2007 adapts this algorithm to work with moving obstacles. In this adaption, multiple alternative trajectories are computed between the minimal and maximal speed of the motor and depending on the location and velocity of the obstacle. Out of these options, the optimal solution is chosen and a route back to the original path is found. The knowledge about the obstacles' location relies on an estimation from sensor data.

In the presented approaches the aspect of complete coverage has been neglected in the execution. Even when the path planning was complete, the obstacle avoidance leads to uncovered areas. So this research approach will focus on the complete execution, especially on coverage of the fleet's route. This is necessary as the AGV's might rely on visual aids for localisation and navigation and a clean path reduces the probability of the AGVs parts. So a simple approach of just driving around the object and continuing the path planned is not sufficient.

An entirely different approach to avoid collisions with the present AGV fleet is by including the cleaning robot into the fleet and letting the fleet manager handle the heterogeneous fleet. Fleets generally can be managed centralised or decentralised. An example of centralised fleets is given by Akkaya and Gökçe, 2022. In this approach a central module schedules and routes the paths for the AGVs. In the decentralised approach examples are proposed by Fragapane et al., 2021 and Azarm and Schmidt, 1997 . Here the AGVs communicate with each other for path allocation. Literature on heterogeneous fleets mostly covers different sizes of AGVs with fairly similar tasks. As in our case, the cleaning robot has an entirely different task, the portability of the approaches onto a mixed fleet with cleaning robots and AGVs is questionable. As this thesis work focuses on the cleaning solution solely, a decentralised version of fleet management can not be assumed and would limit the usage of the proposed system, as hardware restriction would be necessary. Thus, no communication with the AGV fleet is allowed in the cleaning solution.

3 Requirements

To design and implement a solution for the given research problem, first, the requirements have to be formulated. These can be divided into functional and non-functional requirements.

3.1 Functional Requirements

The functional requirements state requirements for the function of the system. The software solution for the cleaning process in an environment with an Automated Guided Vehicle (AGV) fleet can further be parted into the subsections of path planning and collision avoidance. The requirements are established according to these subsections, starting with general requirements needed for the entire system in Section 3.1.1. Continuing with the requirements for the path planning algorithm in Section 3.1.2 and the collision avoidance module in Section 3.1.3.

3.1.1 General Requirements

The scenario stated in the introduction can be directly translated into the requirements given in Table 3.1.

In general, the cleaning solution should be able to avoid all obstacles, static as well as dynamic ones resulting from the AGV driving in the environment. Furthermore, the AGV fleet must not be disturbed by the cleaning vehicle as their work is more time-dependent and therefore has a higher priority. The system should also be able to work in arbitrary environments, and a map of the environment should always be given for the system to be able to plan the cleaning path offline. Details regarding this topic are given in Section 3.1.2. The solution is meant to work for any size or model of cleaning robot. Multiple obstacle avoidance strategies should be implemented to provide a choice for the best option in different use cases and a choice to prioritize different metrics. Additionally, the fleets routes should be cleaned. In the given scenario, the course of the fleet should be clean for different reasons. If the path is dirty, the work of the fleet may

Requirements	**ID**
During cleaning, all collisions with obstacles are avoided.	FR_G1
All collisions with static obstacles are avoided.	FR_G1.1
All collisions with dynamic obstacles are avoided.	FR_G1.2
The AGV fleet is not actively disturbed.	FR_G2
Cleaning can be executed in any enclosed environment.	FR_G3
The cleaning solution takes a map of the cleaning environment as input.	FR_G4
Cleaning can be executed with various kinds of cleaning robots.	FR_G5
Multiple obstacle avoidance strategies are implemented.	FR_G6
The cleaning robot needs to be equipped with an onboard sensor.	FR_G7
The fleets routes are cleaned.	FR_G8

Table 3.1: Summary of the general requirements and their IDs

be disturbed because they have to circumnavigate trash, or localisation might be more difficult as some AGVs rely on external markers located on the floor or on the track.

3.1.2 Path Planning Requirements

The requirements of the path planning aspect of the cleaning solution, including the Complete Coverage Path Planning (CCPP) algorithm, are summarized in Table 3.2.

The most essential requirement in this subsection is the coverage of the entire area or a specified fraction by the user. The executed coverage percentage can be measured in the covered area compared to the free space before the cleaning process. Sufficient full coverage is 95%, as reaching into every corner in a complex environment is unrealistic for this simulation. The planning module has to be able to reach this specified percentage. Another requirement resulting from the discussion in Chapter 2 is the usage of all given information for the planning process. An optimal choice for the Coverage Path Planning (CPP) algorithm should be made considering this information. By requiring a map as input, the cleaning task gets significantly more straightforward because the environment discovery is omitted. In a setting with an AGV fleet, a map for the given environment is usually given as the AGV routes have to be planned as well. The chosen algorithm should therefore utilise the information given by the map. The obtained planned path may be updated during the cleaning process if that is necessary for the execution of obstacle avoidance strategies. Additionally, the planned path has to be drivable, which means if executed, the cleaning robot will not crash into static obstacles.

Requirements	ID
A cleaning path has to be planned.	FR_PP1
The given map has to be used.	FR_PP2
A required coverage percentage can be set by the user.	FR_PP3
The selected percentage has to be reached by the planning algorithm.	FR_PP4
The planned paths have to be drivable.	FR_PP5

Table 3.2: Summary of path planning requirements and their IDs

3.1.3 Collision Avoidance Requirements

The second aspect of the cleaning solution is the detection and avoidance of obstacles during cleaning. The requirements defining this part of the solution are again parted into two fields: collision detection and collision avoidance strategies.

For collision detection, the system must not communicate with the fleet. This requirement ensures that the provided solution can work with any fleet and that no restrictions to the fleet have to be made, as discussed in Section 2.2. The collision detection should be active at all times during the cleaning process of the robot to prevent all collisions with any obstacle at any time. All possible collisions have to be detected, starting with obstacles detected from the sensor information. For each obstacle, a collision point has to be calculated, and all detected collision points have to be avoided by one of the strategies. The stated requirements are listed in Table 3.3 with their respective IDs for evaluation.

Requirements	ID
No communication with the AGV fleet is used for collision detection.	FR_CD1
Collision detection is always active.	FR_CD2
All obstacles are detected.	FR_CD3
Collision points to all detected obstacles are calculated.	FR_CD4
All collision points are avoided.	FR_CD5

Table 3.3: Summary of collision detection requirements and their IDs

Concerning the collision avoidance strategies, the requirements extend some of the general requirements of the system in more detail and are listed in Table 3.4. One requirement all of the strategies have to manage is the avoidance of all calculated collision points stated by FR_CD5. No collisions are allowed, which can be ensured if the cleaning robot keeps a safe distance from static obstacles, for example, the room walls and if moving obstacles are not disturbed. Each strategy should also be targeted to prioritise fulfilling as many of the following requirements as possible to achieve the best efficiency. These requirements

Requirements	**ID**
Strategies can be chosen by the user.	FR_S1
Collision avoidance is executed efficiently.	FR_S2
No coverage percentage is lost.	FR_S2.1
Cleaning time is increased minimally.	FR_S2.2
Overlapping cleaning path is kept at a minimum.	FR_S2.3

Table 3.4: Summary of collision avoidance strategy requirements and their IDs

include the maximal possible coverage percentage fulfilling FR_PP3, minimal cleaning time, and minimal overlapping of newly planned paths. As some of these measures are not achievable when prioritising the nondisturbance requirements of the system FR_G2, it is sufficient to fulfil as many as possible, yet the more constraints are satisfied, the better the strategy. As the best-performing strategy might depend on the given environment, the user should be able to choose a suitable strategy for their specific environment.

3.2 Non-Functional Requirements

In contrast to the functional requirements, the non-functional indicate the requirements for the implementation and execution of the system that do not affect the functionality itself. In this work, this is mainly needed for the evaluation of the designed strategies and therefore affects the simulation requirements for debugging and evaluating the proposed system.

These requirements, shown in Table 3.5, concern the reliability of the evaluation of the functional requirements and the ability to use the implementation for a real cleaning robot. To begin with, the simulation should be realistic so that the physics and movements of the cleaning robot can be compared to real physical motion. The simulation should also help develop the solution by providing debug information and showing wrong behaviour. Additionally, the simulation should provide measures for the evaluation of the developed strategies. The simulation results and the cleaning processes itself should be visualised for the developer so debugging and evaluation can be simplified. A suited map should be used in the simulation to ensure the fulfilment of all the requirements. Another non-functional requirement not only concerning the simulation is that the developed system should be given as a Robot Operating System (ROS) package, as it is a common interface.

Requirements	**ID**
The physical motion in the simulation has to be realistic.	NFR_1
The simulation can be used for code testing.	NFR_2
The results have to be visualised for evaluation.	NFR_3
The simulation has to provide evaluating measures for the strategies.	NFR_4
A realistic map has to be used for testing.	NFR_5
The system has to be provided as a ROS package	NFR_6

Table 3.5: Summary of non-functional requirements and their IDs

4 Architecture

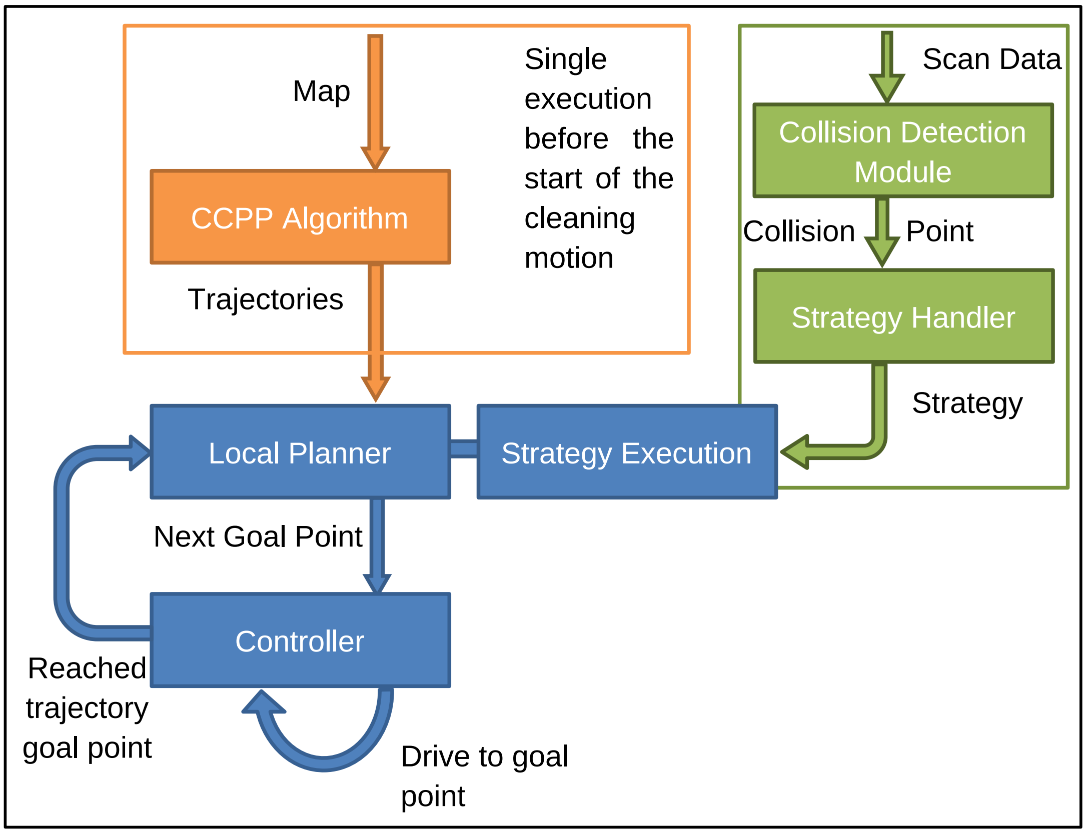


Figure 4.1: The system architecture can be separated into three main components. The `Complete Coverage Path Planning (CCPP)` implementation (orange), which is responsible for the path planning before the execution of the cleaning itself, the `Collision Avoidance` component (green) realised by the developed strategies, and the execution node (blue) with the `Local Planner` and robot controller.

The architecture for the cleaning solution of the proposed scenario is shown in Figure 4.1. This solution can further be separated into the `CCPP` module, the `Collision Avoidance` node, and execution of the cleaning plan through the `Local Planner` node. After a trajectory plan is received from the path planning module, the `Local Planner`, depicted in blue, handles the execution of the derived plan. Simultaneously, the `Collision Avoidance` node in green detects and chooses a strategy for collision avoidance to be executed by the `Local Planner`. The detailed architecture of each module is described in Sections 4.1 to 4.3.

4.1 CCPP Module

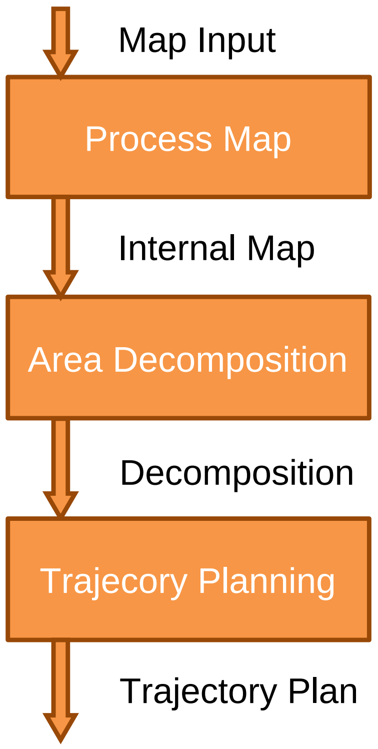


Figure 4.2: `CCPP` architecture: The required map is processed and transformed into an internal map. Then the free area is decomposed into segments. The trajectory goal points are then planned in the `Trajectory Planning` module and forwareded to the `Local Planner` node for execution.

The proposed architecture depicted in Figure 4.2 requires a map as input and processes it to receive an internal map used for the planning process. In the internal map, the cell size equals the cleaning robot's size to ensure coverage of the cell once it is reached. In the next step, the `Area Decomposition` module finds fitting segments according to the chosen algorithm. With the gained decomposition, the `Trajectory Planner` then finds the needed goal points for the execution. The CCPP algorithm is executed by the `Local Planner` so that all relevant information for the execution of the other modules is processed in a single point. These pieces of information include the processed map and decomposition that is also needed for the execution of the collision avoidance strategies. The parameters for the planning process are given in a parameter file shown in Figure 6.1 for ease of use.

4.2 Collision Avoidance Node

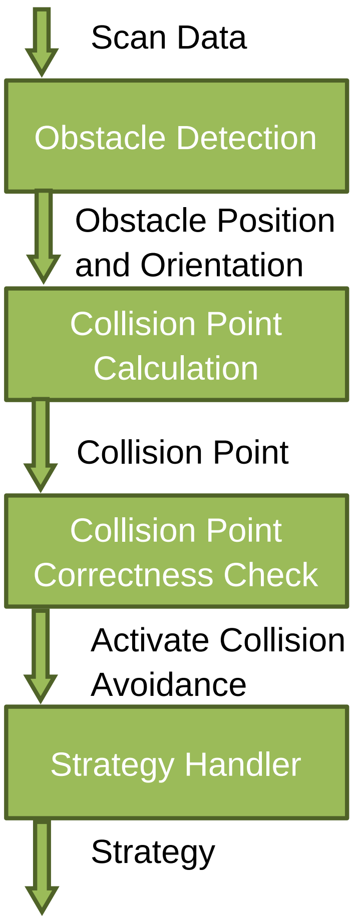

Scan Data
Obstacle Detection
Obstacle Position and Orientation
Collision Point Calculation
Collision Point
Collision Point Correctness Check
Activate Collision Avoidance
Strategy Handler
Strategy

Figure 4.3: The architecture of the `Collision Avoidance` node can be separated into modules: `Obstacle Detection (OD)`, `Collision Point Calculation (CPC)`, and `Strategy Handler`. The chosen strategy is executed by the `Local Planner`.

In Figure 4.3 the architecture of the `Collision Avoidance` node of the cleaning solution is described. This node can further be separated into the `OD` module, the `CPC` module, and the `Strategy Handler` module. In `OD`, the scan data needs to be clustered into objects and tracked over time. The obtained obstacles are further filtered into moving and non-moving obstacles, and the moving objects are then published for usage by the `CPC` module. The `CPC` has to compute a collision point from the cleaning robot's pose and the pose information of the obstacles detected by `OD`. The obstacle's and cleaning robot's speed are also taken into account in the collision point calculation. A validation check for each calculated collision point, which determines whether the collision point will be reached by the cleaning robot, is performed, and then the collision point information is transmitted to the `Strategy Handler`. The `Strategy Handler` chooses a strategy fitting to the given collision point and defined parameters by the user seen in Figure 5.7. This strategy is then executed by the `Local Planner`. The `Collision Avoidance` node is executed simultaneously with the `Local Planner` node in Robot Operating System (ROS).

4.3 Execution

The `Local Planner` node is the primary execution node, and all other components are managed from this node. The initial plan starts from here, and the trajectory points are sent to the controller one by one to control the cleaning robot's movement. Additionally, the necessary strategy nodes are initialised and started here. The primary execution node ensures that all information is gathered and large amounts of data do not have to be sent between different ROS nodes.

For the execution in ROS, the used communication tools and topics are defined in Section A.

5 Design and Implementation

This chapter focuses on the design and implementation choices of the described architecture from Chapter 4. For the execution and validation of the implemented algorithms, first, a simulation is set up, which is described in Section 5.1. Next, the `Path Planning` module is explained in Section 5.2, followed by the description of the visualisation implementations in Section 5.3, which are part of the planning module. In Section 5.4, the controller used in the simulation is described and how it can be substituted in the short validation simulation. The `Collision Avoidance` implementation is described in Section 5.5. Finally, in Section 5.6, the different collision avoidance strategies and their implementation are explained.

5.1 Simulation

A common package for controlling robots in the Robot Operating System (ROS) is the TurtleBot3[1] system. The package provides steering, Simultaneous Localisation and Mapping (SLAM), and further methods to control physical robots. Additionally, the XML-descriptions of their robots for simulation in Gazebo[2] or RViz[3] are included.

In the developed simulation, the XML robot description of the TurtleBot3 waffle pi (see Figure 5.1) is used for the active cleaning robot, as this robot resembles the size of regular cleaning robots and is equipped with a LiDAR that is used for localisation in the environment and obstacle detection. In the XML robot description, the two-dimensional LiDAR is mounted on top of the robot. This physical restriction means only objects with a height larger than the robot are detected in the scan. As the simulation environment is built manually for the particular requirement of this task, this restriction can be compensated during the setup of the environment. On account of simplicity, the same base robot model is also used for the simulation of the surrounding fleet Automated Guided Vehicle (AGV) but is modified for specific use. An additional box is added to

[1]http://wiki.ros.org/turtlebot3 with ROS noetic
[2]https://gazebosim.org version 11
[3]http://wiki.ros.org/rviz for ROS noetic

the AGV's top so the cleaning robot can observe it. For better visualisation, the colour of the robot representing the AGV is changed so its role can be seen directly.

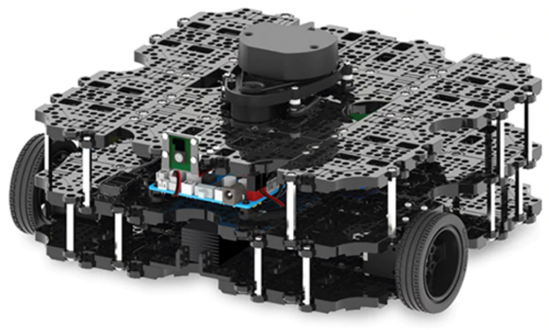


Figure 5.1: The physical TurtleBot3 waffle pi robot model, which xml-description is used for the simulation

With a functioning robot model, the simulation further needs a realistic environment, which is built with fundamental building blocks in an empty Gazebo world model. The simple self-built environment is shown in Figure 5.2. This environment allows the simulation of complex arrangements in the mapping and planning process while also providing enough space for multiple robots to drive around and simulate fleet AGVs executing orders. The simulation environment's size is chosen due to the simple sensor equipment on the robot. The larger the room is, the more static obstacles are needed for the robot to still be able to localise itself through the LiDAR. The small room size also reduces the simulation time for showcasing the different behaviours, as collisions occur more frequently the smaller the room is.

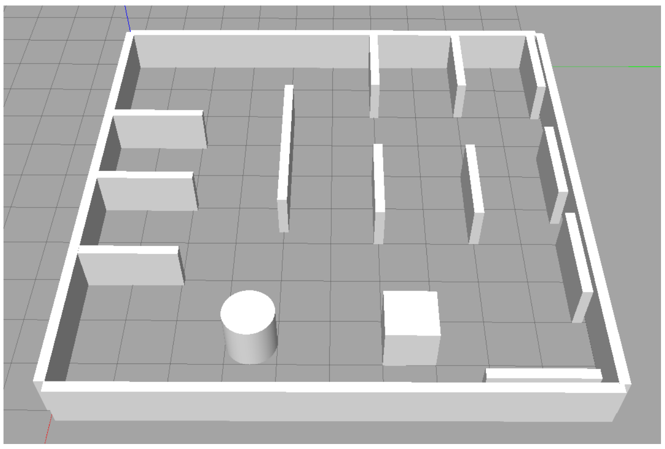


Figure 5.2: This is the simple simulation environment that is built in Gazebo

Although this realistic simulation is well suited for showing the behaviour of the cleaning robot in the environment and comes close to testing on a physical robot, it is not optimal for generating evaluation results. As it is executed in real-world time, generating the necessary number of simulations is very time-consuming. The cleaning time of this small environment with a size of 10 by 10 meters already takes 37 minutes without an AGV fleet present. A second faster simulation is implemented to speed up the evaluation process, which runs no longer in realistic time as the discrete time steps are shortened.

For this faster simulation, the controller module is replaced by a simulator module that publishes the cleaning robot's pose once it is computed. The module takes the trajectory goal point like the controller and visits every point between its current position and the goal point. For every cell, an average time derived from the robot's speed in the Gazebo simulation is added to the simulation time. Additionally, the moving obstacles' positions are computed and published. To compute the obstacle's position in each time step, the route of the obstacle AGV needs to be given at the beginning of the simulation. These positions are given via the $tb3\_1\_poses$ parameter in the parameter file in Figure 6.1. This route has to have the same form as the trajectory calculated for the cleaning robot to be able to reuse the controller. The trajectory goal points always have to be in the direction of the four direct neighbour cells and can not be diagonal. Additionally, the goal points have to form a loop such that the start point can be reached from the endpoint in a straight line. The speed of the cleaning robot and the simulated AGVs are assumed to be the same in this faster simulation, so in a single time step, both the cleaning robot's and the AGV's positions can be updated. If it was not the same this had to be done in different time steps and more adaptions would be necessary in the collision avoidance module. As the strategies should show the general behaviour when a collision is detected, the overall behaviour will not change with this assumption.

$$loc \quad = \quad p_o[t \bmod n_p] \tag{5.1}$$

At the beginning of the simulation, all location points loc of the obstacle p_o are calculated and stored as simulation information in the current map. The current obstacle location is then looked up, as shown in Equation 5.1, with the current simulation time t and the number of location points $n_p = len(p_o)$. In this simulation, the speed for the obstacle and the cleaning robot are assumed to be the same.

5.2 Complete Coverage Path Planning (CCPP) Algorithm

The decision to implement a CCPP Algorithm was made to provide a complete solution in the package developed. By implementing the CCPP algorithm and not using an existing implementation, changes required to work with the implemented strategies can be made more easily. In addition, the ROS packages providing already implemented CCPP Algorithms require specific maps that would limit the usage of the cleaning solution as a whole. The following data structures are developed to implement the algorithm as described in Chapter 4: An internal `Map` for planning, the `Area Decomposition` module, which divides the area into segments, a `Decomposition` class and the `Segment` itself.

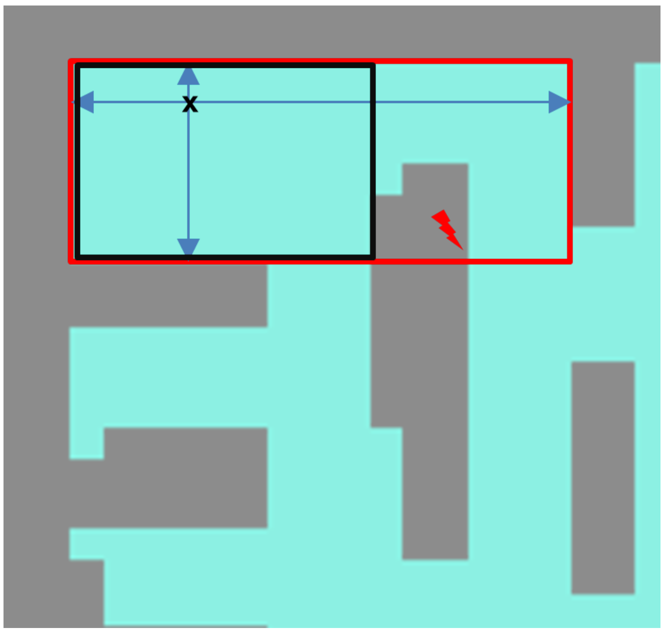


Figure 5.3: Process of `Area Decomposition` into segments. The x is the position the segment is planned from; the blue arrows show the maximal dimensions in each direction. The red segment would be the original segment before checking for inlying obstacles. The black segment is the correction of the coordinates

The internal `Map` consists of the occupation grid, which shows the static obstacles in the map used for initialisation, and the cleaning grid, which holds the cleaning information during the execution of the cleaning process by assigning specified values for cleaned and not cleaned to each cell. During initialisation, the original map is resized into cells the size of the cleaning robot by eroding the picture and then interpolating the values during the resizing with OpenCV's resize functions *INTER_AREA* option. Resizing the grid to the robot's size ensures coverage in

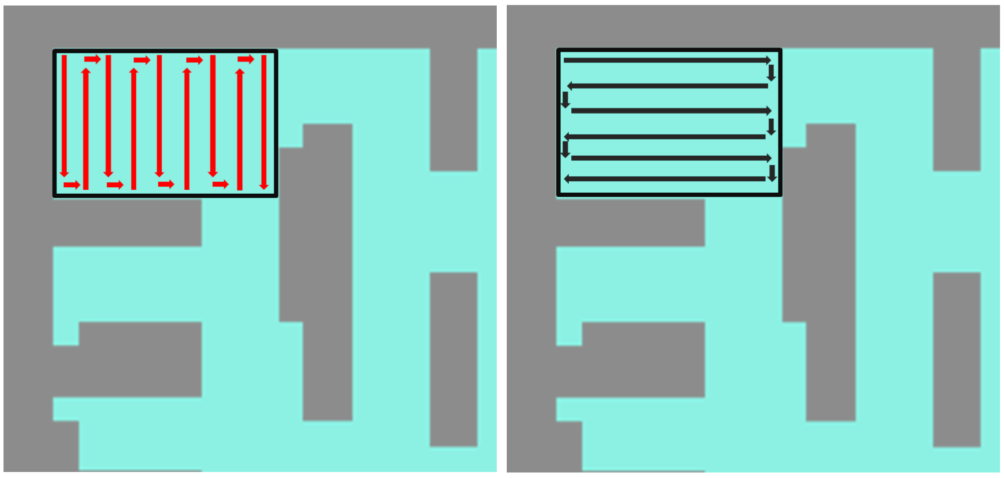

(a) expensive cleaning motion (b) more efficient cleaning motion

Figure 5.4: Trajectory planning in a segment

the cleaning process as soon as the cleaning robot reaches the centre of the cell, facilitating the planning process. The map module also transforms the trajectory point coordinates between global ROS coordinates and local coordinates, which results from resizing the original map. In addition, it plots the state of the map for visualisation.

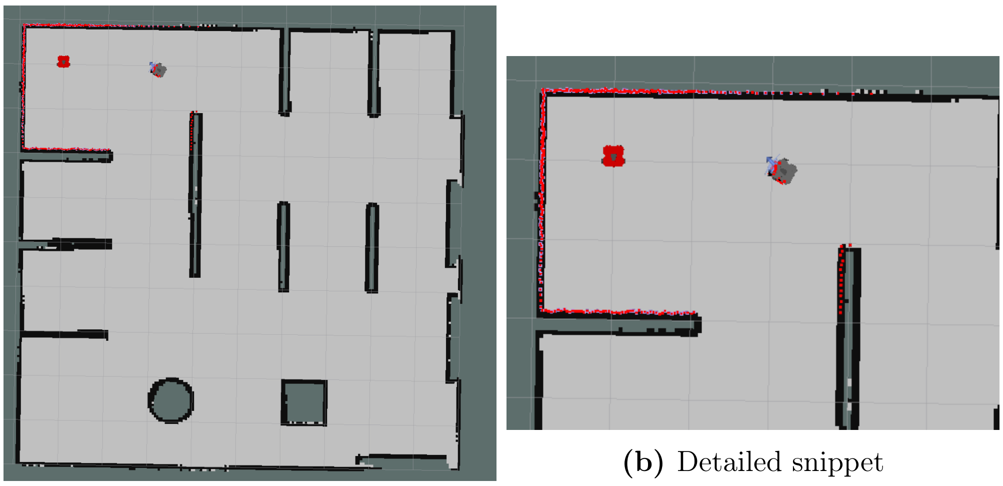
The data structure `Segment` consists of the corner points that define the area and a list of the planned trajectory for this segment. Each segment has an ID according to the order in which the segments are planned to be processed. Every segment also holds the segment ID of the next segment. The segment is used in the area decomposition process. The data structure `Decomposition` stores a list of segments and all trajectory points combined in a single route.

The `Area Decomposition` algorithm parts the given area into segments similar to the trapezoidal exact cell decomposition algorithm. Each segment must be shaped such that a back-and-forth motion can be planned. A segment can not contain static obstacles. The process to get such a decomposition is shown in Figure 5.3. The `Area Decomposition` process starts at the cleaning robot's starting position and finds the furthest distances in each direction with no obstacle present. These distances are combined into four corner points representing the biggest possible segment from this start point. If the obtained area is free of obstacles, it is stored as a segment. If it is not free, the area is decreased by using the next furthest static object. The next starting point is chosen randomly from the remaining not planned cells of the map.

In the next step of planning the cleaning route for the robot, the `Trajectory Planner` module finds intermediate goal points for each gained segment shown in

Figure 5.4. For the simple controller to work, the path between two trajectory points always has to be directed in north, east, south, or west direction. They can not be diagonal. The obtained segments must be ordered so that the path between the segments is minimal to obtain an optimal route. The segment sequence is determined by a greedy algorithm which always takes the segment with the shortest distance between one of its corner points and the corner points of the last chosen segment. Once the order is obtained, each segment is planned with a zig-zag pattern from the start corner point to the last. The zig-zag pattern is always planned in a way such that the turning points are at the maximum lateral distance. The distance between two planned trajectory points has the maximum length of the global parameter *traj_length*.

5.3 Visualisation



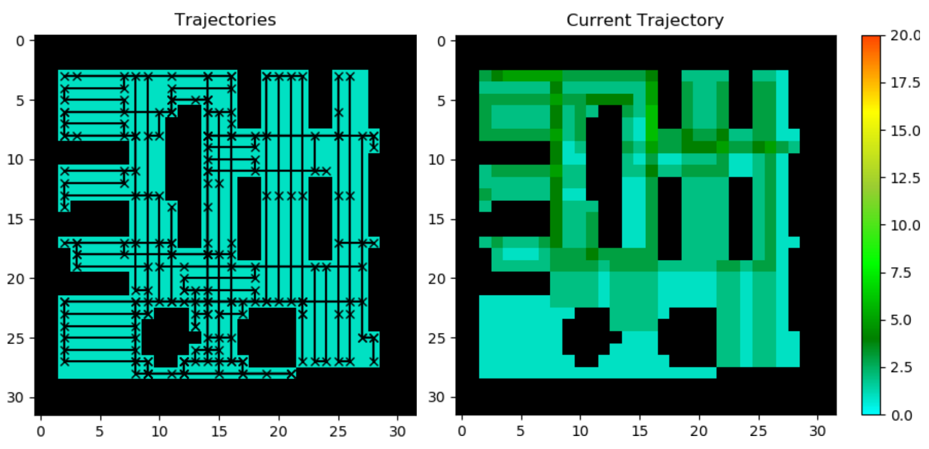
(a) Full Map

(b) Detailed snippet

Figure 5.5: Visualisation in RViz: The red robot is the cleaning robot. The second robot(grey) represents a fleet AGV. The red dots mark the LiDAR data and the small arrow shows the estimated velocity

To visualise the real-world time simulation RViz is used. Here the cleaning robot's movement is shown on the original map of the room, obtained by using the SLAM feature of the TurtleBot3 mapping tool. The LiDAR data is visualised as the red dots in Figure 5.5. Additionally, the clustered objects from the Detection and Tracking of Moving Objects (DATMO)[4] package can be seen as boxes, and the estimated speed of the object is displayed by the arrows at each obstacle. With the grid and the top view, RViz is chosen over Gazebo in this work. To prevent

[4]https://github.com/kostaskonkk/datmo



(a) Planned route **(b)** Current trajectory section

Figure 5.6: Visualisation plots of the internal `Map`. Black are the static and known obstacles; the crosses mark the trajectory goals. The current trajectory plot shows the current cleaning progress. The greener a cell, the more times the cell was cleaned, while light blue shows the yet to be cleaned area.

the computationally demanding Gazebo visualisation from interfering with the planning execution, it is deactivated.

The internal grids of the planning maps are plotted with matplotlib[5] to visualise the planning of the CCPP algorithm. In Figure 5.6a the initially planned trajectory points are shown. The subsequently updated current trajectory plot in Figure 5.6b shows the cleaning progress updated with the robot's location. Once the cleaning robot reaches the new trajectory goal point, the plot is updated. The different shades of green seen in this plot demonstrate the number of times the cleaning robot passed the individual cell. The greener the tone is, the more often the cell was visited. Black shows the static obstacle, and light blue is the not yet cleaned area.

For the faster simulation, these plots are the only visualisation available as the odometry of the robots is not constantly being published, and RViz can not visualise it. This means that in this simulation type, the position of the fleet AGV can only be monitored by checking the topics of the respective robots.

5.4 Robot Controller

A simple point-to-point controller is implemented to control the robot model. It uses the current position and orientation of the robot by listening to the odometry topic of the respective robot. The controller is driving straight

[5]https://matplotlib.org/ version 3.1.2

towards a goal position and calculates the angle between the robot's orientation and the orientation of the goal position. If the angle exceeds a holistically chosen threshold, the controller stops moving forward and turns in the desired orientation. Once the angle is small enough again, the forward motion continues. The robot's movement is realized by publishing the odometry topic of each robot.

As the fleet AGVs are modelled and controlled by the same robot description and controller during the simulation, and each has to be piloted individually, a separate namespace for each robot and AGV is introduced. For the fleet AGV, the controller receives the goal point from a list of points that form its route. This route is repeated until the simulation is stopped. The fleet AGV has no collision avoidance and constantly drives between its start and end point. Its trajectory is not separated into smaller sections unlike the cleaning robot's since the exact following of a line to get coverage is unnecessary.

The local planner initialises and runs the cleaning robot controller, combining the execution of obstacle avoidance and general steering. The subsequent trajectory goal points are published to the */predictions/trajectories* topic for visualisation and debugging purposes. The number of points published is defined as $num\_trajectories$ in the parameter file Figure 6.1. To execute the strategies the original controller returns and a new instance depending on the strategy is started once the new trajectory is planned.

5.5 Collision Avoidance

The distances received from the LiDAR data must be filtered first, so points within a threshold are clustered together to recognise obstacles with LiDAR scans. For realistic estimation of the moving obstacle's location, the ROS package DATMO is used for filtering as well as tracking the filtered objects. The identified clusters that match a specific form are tracked over time, and the estimated track information is published for usage. Each of these tracks gets an ID, a position, and orientation in the form of an odometry message type and the size of the tracked object. As this detection works with an estimation rather than known positions, a certain amount of error is introduced, which will affect the evaluation of the strategies in the simulation and execution of obstacle avoidance; a different detection approach is chosen for the evaluation with the simulation. In ROS, the position and orientation of a controlled vehicle are known through the odometry topic and can be used by any other active node. Using this information eliminates the error from the estimation for evaluating purposes. The DATMO solution is still necessary for the execution on a physical robot.

The possible collision point with the cleaning robot's trajectory is calculated from the odometry information of the obstacle derived in the last step in the

`Collision Avoidance` node. The obstacle's velocity is calculated as the mean value of the distance between each detected obstacle position in a specified time interval. The collision point is then calculated between the cleaning robot and a number of its following trajectory goal points, set by *num_trajectories* in Figure 6.1, and the current and extrapolated position of the moving obstacle. The estimation of the collision point does not only rely on space but also on time. If the time until the collision occurs is longer or shorter than the cleaning robot takes to reach that position, the collision point is invalid. After the temporal validation check, every collision is published to the */predictions/collisions* topic for debugging and visualisation purposes. If the point is valid and the time to reach the position lies within a threshold set by the *max_time_step* parameter, the collision point is sent to the strategy handler to choose a collision avoidance strategy. The `Collision Avoidance` node continues by checking if the collision with this obstacle is still valid. Once it is free, either because the obstacle is no longer detected or the estimated orientation changes, the cleaning robot can continue with its path.

5.6 Collision Avoidance Strategies

Once a collision point is detected and the `Strategy Handler` node receives the collision point and time, a fitting strategy is chosen. These decisions are influenced by parameters set by the user in the *strategy_params.yaml* file in Figure 5.7. If none of the strategies are chosen a default strategy is used, set at the overall best-performing strategy evaluated in Section 6.3. The `Strategy Handler` node then sends a service message to the local planner node for the execution of the chosen strategy. As all data structures needed for planning are initialised in the `Local Planner`, and it would be inefficient to send the data structures to different ROS nodes, the strategies must also be executed here. In this work, four strategies are implemented and `Replan` is described in Section 5.6.1, `Reorder` in Section 5.6.2, and both `Follow` strategies in Section 5.6.3.

```yaml
follow_strategy: True
strategy: "replan"
```

Figure 5.7: Strategy Parameter File *strategy_params.yaml*

5.6.1 Replan Strategy

The `Replan` strategy is a simple strategy based on the random factor in the planning algorithm. If a collision is detected, the cleaning robot stops and waits a previously chosen time. Either the collision resolves in this time, e.g., because the estimated orientation of the obstacle or the cleaning robot changes, or the current execution plan is deleted and an entirely new path starting from the current position is planned as described in Section 5.2. This replanning only decomposes the not yet cleaned area of the map. The rest of the planning process is the same as at the beginning of the cleaning process. As the collision point with the obstacle lies ahead, stopping prevents a collision in the immediate future. Due to the random factor in the planning algorithm, the chance of finding a new path that will not cross the AGVs path is high. Once the plan is calculated, the cleaning robot starts driving to the new goal points, and the procedure is repeated once a new collision point is determined.

5.6.2 Reorder Strategy

Contrary to the `Replan` strategy, upon a detected collision this strategy will only reorder the given segments from the decomposition into a new driving path. Again, as in the `Replan` strategy, the cleaning robot stops when a collision is detected and waits for a specified amount of time set by $max\_waiting\_time$ parameter to dissolve the collision automatically. Instead of finding a new decomposition after the waiting period, the existing decomposition is used for calculating the new cleaning path. The segments assigned to the estimated positions of the obstacle that were received during the waiting time are sought, and these segments are excluded from the reordering process to avoid the currently blocked area. The blocked segments are added at the end of the segment queue, as they are more likely to be accessible later. In future work, the segments could be tagged for considering the amount of blocked time for planning at the beginning of the next cleaning task. The reordering process is very similar to the original planning process. First, the segments are again ordered greedily by the smallest distance between the end and start points of the segments. Secondly, the trajectory between the endpoint of the last segment and the start point of the next segment is computed and queued with the other trajectory goal points. If the `Reorder` strategy is chosen, the maximum waiting time can be chosen longer than for the `Replan` strategy, as the replaning calculations are much more computationally expensive than the reordering.

5.6.3 Follow Strategies

The `Follow` strategy is the third strategy implemented in this work. It is more complex and can be combined with the two previously described options, namely `Follow-Replan` and `Follow-Reorder`. The idea for this strategy stems from typical robot navigation and orientation problems. Some providers use codes on the ground for navigating through the environment, resulting in high motivation to keep these marks on the driving path clean. Also, the cleaner the path for the fleet AGVs is, the less interruption they will get from mechanical failures.

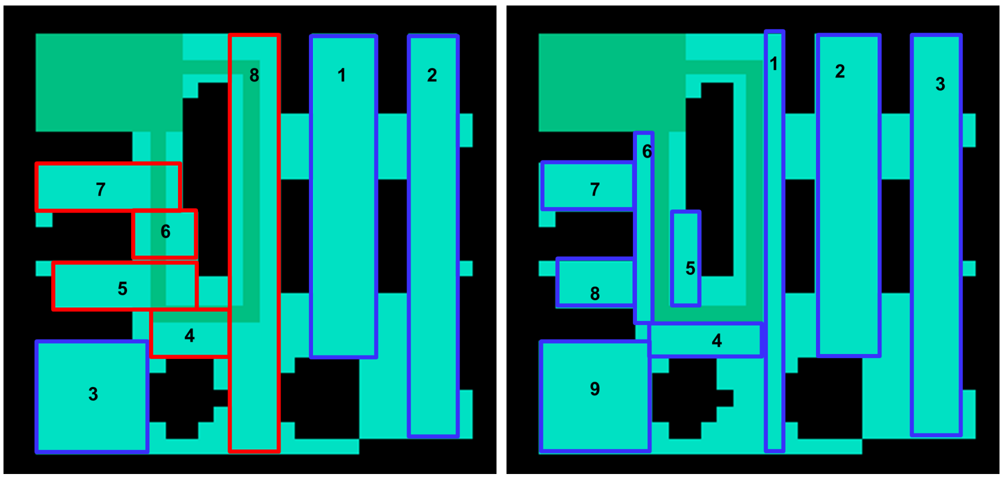
Additionally, if the cleaning robot follows the driving path of the fleet AGV, the most frequented part of the area is already cleaned and can be excluded from the future path. This will also result in reduced future collisions, because if the frequented areas do not have to be crossed again the collision probability reduces, and no collisions will occur during the following as the fleet management usually keeps a distance between the driving AGVs.

```
def check_if_following_possible(robot, obj):
    angle = robot.orientation - obj.orientation
    dist = robot.pose - obj.pose
    if the abs(angle) is smaller than 90 degrees:
        if abs(dist) is bigger than max_obstacle_dist:
            start following
        else:
            wait
    else
        if first check:
            do not start
        else:
            stop following

```

Figure 5.8: Pseudo code following check

When this strategy is chosen, the cleaning robot, in contrast to the other strategies, starts following the obstacle immediately if following would not result in a collision. The related checks, visualised in the pseudo-code in Figure 5.8, are performed once at the beginning and repeatedly during the following process. They rely on the position and orientation information of the obstacle and cleaning robot. If the cleaning robot and the obstacle are oriented in a similar direction, the cleaning robot starts or continues to follow. Suppose the cleaning robot is or gets too close to the AGV while following it. In that case, it stops and resumes following once the distance is big enough again or, in the beginning, wait until the appropriate distance is given before the following process is started. If the desired objects' orientations lie opposite, the cleaning robot will not start following or stop the following process. In the first check, the cleaning robot will not start following at all and directly execute the secondary given strategy, `Replan` or `Reorder`. During a later check, a new path must be planned using the already implemented `Replan` strategy. This replan is necessary as parts of the old route might have already been cleaned during the following process. Figure 5.9a shows the affected segments in red. If the original path is used, the benefits of cleaning the obstacle path first and having fewer collisions are diminished. Even with an immediate reordering process described by the numbers of the segments in Figure 5.9a, the benefits would be minor. Contrary to a replaning action after the following process, the results in Figure 5.9b show that no revisiting is caused through the segmentation alone. A drawback is that the obtained segments are smaller than the original ones, and the paths between the segments increase. The effects of this behaviour are evaluated in Section 6.2.3.



(a) `Follow-Reorder` strategy without additional replanning (b) `Follow-Reorder` strategy with replanning

Figure 5.9: This plot shows the necessity for an additional replanning after the following procedure in the `Follow-Reorder` strategy

The obstacle's poses are stored in the `Follow` module during the checking and waiting time. If the criteria are met, the cleaning robot will approach the stored locations of the obstacle. The interval between the goals used for following is set at a defined distance between the goal points and can be adjusted to the need of the specific environments. The coarser the interval is set, the more inaccurate the `Follow` procedure gets. Once the cleaning robot reaches a goal location, the point is added to a list which is used for the end check of the `Follow` procedure. If the new goal point received while following the obstacle is too close to an already reached following goal point from the same obstacle, the following ends there, and a new trajectory path is planned.

During the `Follow` procedure, the needed criteria are checked repeatedly to be able to stop the process if a collision with the followed obstacle is detected. The `Follow` module also takes over the controller during the strategy execution. If the following process does not start, one of the other strategies can be chosen. Which strategy combination, `Follow-Replan` or `Follow-Reorder`, is more efficient is discussed in Section 6.3.

6 Evaluation

In this chapter, the proposed cleaning solution is evaluated. First, the fulfilment of the requirements is checked in Section 6.1. In Section 6.2, the four introduced obstacle avoidance strategies for cleaning robots are evaluated by their performance achieved in the simulation. Finally, the obtained strategy results are compared, and an optimal strategy suggestion is made in Section 6.3.

6.1 Requirements Evaluation

The proposed solution is evaluated in terms of fulfilling the requirements stated in Chapter 3. An overview of the fulfilment status is also given in Table 6.1. First, the non-functional requirements listed in Table 3.5 are evaluated, as these are relevant for checking the fulfilment of the functional requirements.

NFR_1 is fulfilled by using the Gazebo simulation and the definition of the TurtleBot description. The Gazebo simulation was used for testing and debugging the implemented solution (NFR_2) and provided a realistic physical simulation. Through the visualisation in RViz and the plotting of the internal map, the position of the cleaning robot was tracked and verified. In addition, the visualisation described in Section 5.3 shows the results necessary for fulfilling NFR_3. In Section 5.1, the second simulation was introduced to speed up the evaluation process such that sufficient results could be generated for evaluating the strategies. The necessary implementations for generating evaluation metrics (NFR_4) were made and described in Section 6.2. Regarding NFR_5, the environment introduced in Section 5.1 is big enough for having multiple obstacles in the cleaning area and realistic scenarios for the fleet's movement. Requirement NFR_6 is satisfied through the design of the system itself.

Regarding the general functional requirements listed in Table 3.1, FR_G1 can be verified by the simulation shown in RViz. Sub-requirements FR_G1.1 and FR_G1.2 are also fulfilled since neither non-moving nor moving obstacles were hit during the testing of the software and the evaluation of the strategies. Through the same observations, FR_G2 is fulfilled since no disturbances were noted. The given map is preprocessed into the internal map and used in the planning process, meeting the requirement FR_G3. The path to the map is set in the parameter file in Figure 6.1, and through the design of the system, FR_G4 is also attained. The cell size of the internal map is adapted to the cleaning robot's size to satisfy FR_G5 since the Complete Coverage Path Planning (CCPP) algorithm does not depend on other features of the robot. The parameter *robot_size* needs to be set correctly by the user, with which the grid of the cleaning map is initialised. As stated in Section 5.6, four different collision avoidance strategies were implemented; therefore, FR_G6 is fulfilled. Requirement FR_G7 is fulfilled as the simulation works with the given LiDAR scan from the TurtleBot3 robot description. Requirement FR_G8 can be evaluated by comparing the obstacle's known positions with the updated cleaning process map provided, by the `Local Planner`. As a reminder, an example of this cleaning map can be seen in Figure 5.6. In the full coverage case this is implicitly given, for the other strategies with lower required settings, only the follow strategy fulfils this requirement.

Continuing with the path planning requirements stated in Table 3.2, the implemented path planning algorithm decomposes and plans paths for the cleaning robot to follow. As described in more detail in Section 6.2, the cleaning motion works, thus FR_PP1 is satisfied. As shown for requirement FR_G4 a path to the map is given and in the map processing module, the map is used, so FR_PP2 is fulfilled. FR_PP3 is assured by the parameter *stop_criterion* in the parameter file in Figure 6.1. If this is not set, the percentage defaults to full coverage. However, FR_PP4 is not fulfilled, since the required percentage is not always reached which will be discussed in Section 6.2. The trajectory plans were checked during the evaluation process, and no invalid trajectories were planned. Hence, FR_PP5 is fulfilled.

All collision detection requirements given in Table 3.3 are fulfilled. By design, FR_CD1 is fulfilled, since the collision detection is performed with the simulated LiDAR data, and no communication is necessary to detect collisions. Additionally, the collision detection node is started by the Robot Operating System (ROS) launch file and has a parameter set to *required*. Thus, if the node fails, the whole process is stopped. Hence at no time during the cleaning process the collision detection is inactive, and FR_CD2 is satisfied. The obstacle detection was verified in the simulation and by checking the topic information of the collision detector. By performing the same way of validation, FR_CD3 is also fulfilled. The designed collision point calculation is implemented and its functionality verified, and therefore FR_CD4 is accomplished. Moreover, all valid collisions trigger a service call to the `Strategy Handler`, which leads to the execution of the collision avoidance. If all collision avoidance strategies are valid, FR_CD5 is fulfilled as well. This was checked in the simulation using the known positions of the Automated Guided Vehicle (AGV)s and the robot.

Regarding the strategy requirements from Table 3.4, the first requirement FR_S1 is fulfilled by defining the strategy parameter in the *strategy_params_file.yaml* given in Figure 5.7. The best collision avoidance strategy for fulfilling FR_S2 and its sub-requirements is yet to be determined in the results evaluation subsection 6.3.

ID	**Fulfilled**	**ID**	**Fulfilled**
NFR_1	yes	FR_PP1	yes
NFR_2	yes	FR_PP2	yes
NFR_3	yes	FR_PP3	yes
NFR_4	yes	FR_PP4	no
NFR_5	yes	FR_PP5	yes
NFR_6	yes	FR_CD1	yes
FR_G1	yes	FR_CD2	yes
FR_G1.1	yes	FR_CD3	yes
FR_G1.2	yes	FR_CD4	yes
FR_G2	yes	FR_CD5	yes
FR_G3	yes	FR_S1	yes
FR_G4	yes	FR_S2	see 6.2
FR_G5	yes	FR_S2.1	yes
FR_G6	yes	FR_S2.2	yes
FR_G7	yes	FR_S2.3	yes
FR_G8	yes		

Table 6.1: Summary of requirement fulfillment

6.2 Simulation Results

Simulation runs of every strategy have been executed with different required cleaning percentages set by *stop_criterion* in the simulation environment introduced in Section 5.1. The used parameter values can be seen in Figure 6.1 and will now be explained in more detail. For the parameter *traj_length* choosing a high value is desirable. Yet the larger it gets the more inaccurate the driving of the robot gets and the driven path does not clean the desired area. However, high values also lead to a more cautious system as the detection range of the system increases. The higher range leads to more information and therefore earlier and more frequent detection of collisions. The detection range is also affected by the *num_trajectories* parameter, which selects the number of trajectory points looked ahead during collision detection. The more trajectory points are included in the collision detection the earlier collisions are recognised. If *traj_length* is already chosen high, *num_trajectories* can be selected smaller, yet the bigger it is, the more collisions are detected. The parameter *robot_size* is chosen to match the TurtleBot3 robot size, wherease *max_object_dim* limits the size of objects that are detected by the Detection and Tracking of Moving Objects (DATMO) package. The parameter *max_waiting_time* sets the seconds the robot waits until the strategy is executed.

```
    map_path: "/ccpp_fleet_env/maps/simple_sim_big_mod.yaml"
    traj_length: 10
    robot_size: 0.35
    max_robot_dim: 0.35
    max_object_size: 0.4
    max_time_step: 4
    num_trajectories: 3
    max_waiting_time: 4
    stop_criterion: 0.001

    tb3_1_poses: [5, 9, 20, 9, 20, 15, 4, 15, 4, 9]

    startx: 1.0
    starty: 1.0
```

Figure 6.1: Simulation parameter file showing the values used for simulation

For each strategy and desired cleaning percentage, 50 simulation runs with five different starting points $(startx, starty)$ have been executed. By varying the starting position of the cleaning robot the places of detected collisions vary from run to run. With the additional inconsistency of the planned path caused by the random factor in the decomposition, even more variety in collision points is generated. As this behaviour can already result in very different results and the room size is still relatively small, no second fleet AGV was introduced in the evaluation simulations.

The metrics used for the evaluation are derived from the sub-requirements of FR_S2. The first metric that is used for comparison is the execution time t_{exec}. This is the runtime of the fast simulation used for evaluating. As the execution of the calculated paths can be much faster than real-time, t_{exec} does not represent the time the cleaning process needs but gives some information on the path planning calculation time and complexity of the algorithm. This time is also influenced by the performance of the simulation host and as a consequence is inconclusive for the strategy's actual performance. The cleaning time t_{clean} is the time estimated for the cleaning process by assuming the cleaning robot takes two seconds per cell to clean. This value is derived from evaluating the runtime of the realistic Gazebo simulation and its cleaning percentages. The third metric is the coverage percentage p_c, which is expected to be higher than the required percentage and is used for comparing the performance of the cleaning strategies. It is acquired by counting the covered cells in the cleaning grid after execution and setting it in relation to the initial free area. The fourth metric used for comparison is the percentage of repeatedly visited cells during the cleaning process; the multi-coverage percentage p_{mc}. This metric counts the cells in the cleaning grid which were visited multiple times and puts it in relation to the cleaned cells. This behaviour increases t_{clean} and wastes resources by cleaning an already cleaned area. From the 50 runs of each parameter set, the mean values for each metric are compared, and for the coverage percentage metric p_c a normalisation with t_{clean} is also given for better comparison.

Base Run	t_{exec} [s]		t_{clean} [min]	
p_c^*	mean	std-dev	mean	std-dev
50%	44.3	10.25	12.5	1.85
75%	87.6	19.52	19.9	2.15
90%	134.6	24.24	26.3	2.62
100%	253.8	34.98	37.3	2.67

	p_c [%]			p_{mc} [%]	
p_c^*	mean	std-dev	norm [%/min]	mean	std-dev
50%	52.4	5.74	4.18	7.2	3.23
75%	72.7	3.97	3.65	13.1	3.97
90%	86.0	2.99	3.28	19.1	5.03
100%	94.9	1.38	2.55	33.8	3.43

Table 6.2: Base run simulation results for different required cleaning percentages (p_c^*) in terms of execution time (t_{exec}), cleaning time (t_{clean}), covered percentage (p_c) and multi-coverage percentage (p_{mc})

For a better assessment of the proposed strategies, simulation runs with no moving obstacles were performed 50 times with varying starting positions as well. The results of these base runs are shown in Table 6.2. Runs with 50%, 75%, 90%, and 100% required cleaning percentage p_c^* were executed to recognise the behaviour of the planning algorithm. The lower percentages are unlikely to be chosen by a user but can still give insight into the behaviour of the algorithm in general. Looking at the results concerning t_{exec} the mean values almost double for each required p_c^*. Additionally, with a standard deviation of 10 compared to 44 seconds runtime overall for 50% p_c^* the results already vary a lot. This might be caused by the algorithm of the planning process leading to a high variance in the coverage percentage as well as varying computation times of the planning process. In contrast, t_{clean} is more suitable for comparison and also the deviation is smaller. This metric is directly related to p_c. As more percentage is reached for every p_c^* the runtime is supposed to rise.

The p_c values reach p_c^* in the 50% case only, in the other cases p_c^* is not reached. This behaviour was not intended and means that the requirement FR_PP4 is not fulfilled. The implemented CCPP algorithm is not planning a path covering the entire area. During the decomposition, the desired area is still planned almost completely. Only single cells, which are too small for a segment, are left out in the planning process as shown in Section B. The minimal unplanned cells here are however not the main cause of the low p_c. Therefore the error of not reaching p_c^* must be caused by the trajectory planning module, as the cleaning robot follows the planned route directly, the segments could not have been planned completely.

The standard deviation for p_c gets smaller the higher p_c^* is. Putting that into relation to the highly varying t_{exec} this means the execution time is more dependent on the path planning time than the variance of cleaning percentage. Looking at the normalised values of p_c, the cleaning time does not only increase due to the higher percentage reached but also less coverage is executed in the same cleaning time. This can possibly be explained by the increase of p_{mc} for the different measurements. For 100% p_c^* a third of the cleaned area is cleaned multiple times. As the segments are planned randomly into the free area, the remaining free areas get smaller the bigger p_c^* is, and therefore also the segments that are planned at the end of the planning process. The path planning between the segments then increases the p_{mc}, as already cleaned areas have to be crossed. For example, if a newly planned segment is surrounded by cleaned cells, the path between the segments has to revisit the already cleaned area. To keep the algorithm's complexity manageable, it does not check whether the area was already cleaned or not for the paths to and from this segment. Additionally, the more segments there are, the more paths between segments have to be planned. Therefore p_{mc} rises as well and adds to the impracticability of the implemented planning algorithm.

The following subsections introduce and discuss the obtained results for each strategy compared to these reference results. Concluding, a comparison between the strategies is discussed.

6.2.1 Replan Strategy

The results of the `Replan` strategy can be seen in Figure 6.2, and mean, standard deviation and normalised values are given in Table 6.3. For the `Replan` strategy compared to the base runs, both t_{exec} and t_{clean} increase. t_{exec} grows drastically to 436 s from 253 s in the full coverage case and t_{clean} rises significantly from 48 min to 37 min in the base runs. As for p_c, the effectively covered area is larger than p_c^* and also than in the reference runs with 57%, 7Yet, as expected, the required 100% is not reached, but the strategy exceeds the 95% required for full coverage. Comparing the normalised p_c values to the base runs, the `Replan` strategy reaches slightly less coverage per minute for 50% and 100% p_c^*; in the other cases the normalised p_c increases. Looking at the area that was visited multiple times by the cleaning robot, the percentage increases when an obstacle

Replan p_c^*	t_{exec} [s] mean	t_{exec} [s] std-dev	t_{clean} [min] mean	t_{clean} [min] std-dev
50%	73.74	19.25	15.47	2.72
75%	162.50	37.70	27.14	3.25
90%	262.36	49.97	36.49	4.50
100%	436.24	101.42	48.63	6.46

p_c^*	p_c [%] mean	p_c [%] std-dev	p_c [%] norm [%/min]	p_{mc} [%] mean	p_{mc} [%] std-dev
50%	57.10	7.03	3.69	9.07	4.14
75%	78.46	3.18	2.89	17.49	4.76
90%	91.12	3.56	2.50	23.79	4.92
100%	99.12	0.77	2.04	36.55	5.85

Table 6.3: Replan simulation results for different required cleaning percentages (p_c^*) in terms of execution time (t_{exec}), cleaning time (t_{clean}), covered percentage (p_c) and multi-coverage percentage (p_{mc})

is met. The highest p_{mc} resulting from the 100% p_c^* run is 36.55%, which is 3% higher than in the reference. In general, the higher the required coverage percentage p_c^* is, the bigger p_{mc} gets, just like in the reference runs.

Since in the base runs, no obstacles must be avoided, the increase of t_{exec} is expected for all strategies. The higher p_c compared to the base runs show that an obstacle avoidance strategy actually benefits the cleaning solution in this work. p_c exceeding the values from the reference run could result from the implementation detail of only checking for completion of the cleaning process when a collision is detected. This behaviour was implemented intentionally because as long as no obstacles have to be avoided, a higher percentage of clean area is usually desirable, and no extra time for collision avoidance is used. The additional covered percentage is similar across the runs with different p_c^*, which means the area cleaned until the subsequently detected collision is similar. The decrease of the normalised p_c values could result from the waiting time before the execution of the strategy. In this time the robot does not clean any cells and therefore, in the same time, less area is cleaned. Additionally, the higher p_{mc} influences these results as the cleaned area also does not increase here. The higher p_{mc} stems most likely from the implementation of the planning algorithm in the same way as discussed in the base runs described in Section 6.2.

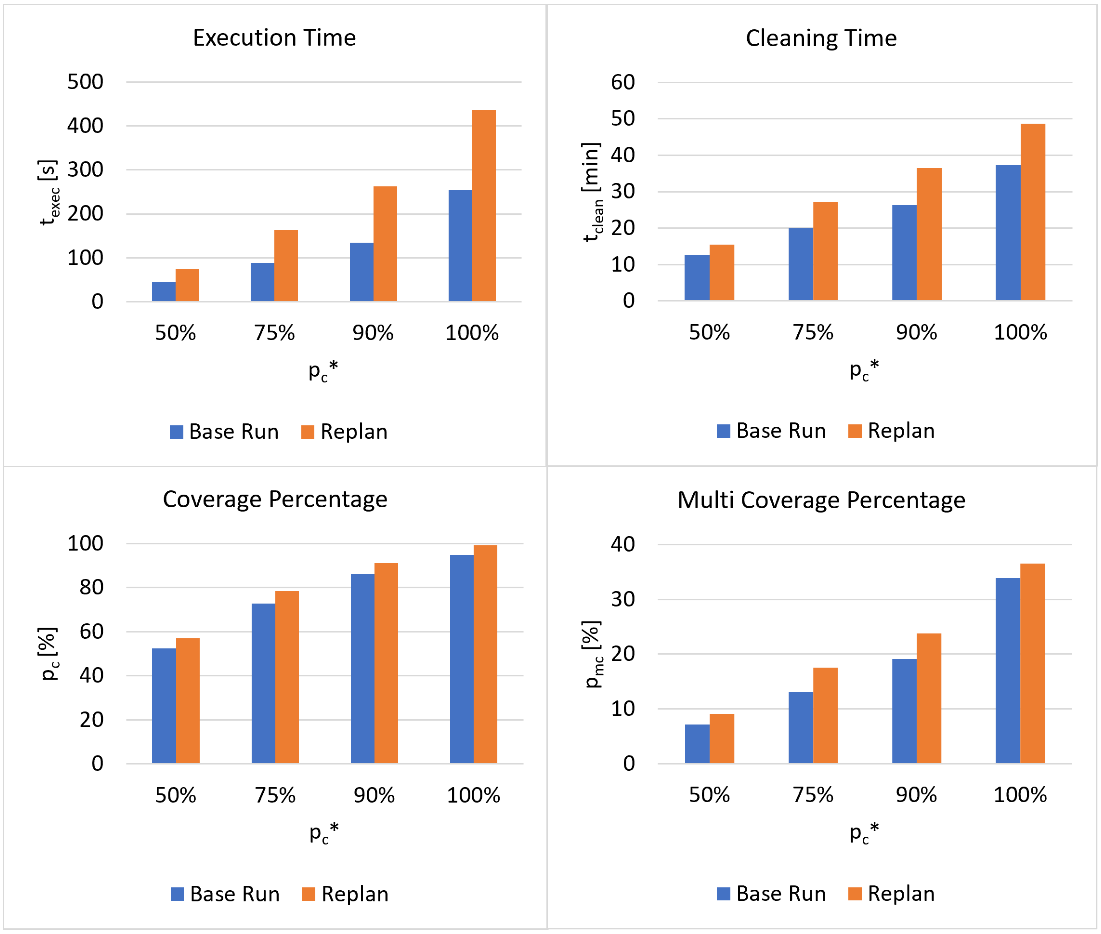


Figure 6.2: The comparison of metrics between the base runs with no obstacles, and the `Replan` strategy with obstacles for different required coverage percentages (p_c^*) is shown

6.2.2 Reorder Strategy

Similar to the `Replan` strategy, both t_{exec} and t_{clean} of `Reorder` also increase compared to the base runs, except for the full coverage case. The results can be seen in Table 6.4 and Figure 6.3. In the full coverage case, `Reorder` takes 245 s for t_{exec} and 33 min for t_{clean}. Compared to the base runs, these values are smaller. Regarding the desired coverage percentage, p_c^* is usually not reached and p_c is even smaller than for the base runs, as the mean p_c values are 50%, 72%, 84%, and 91%. The normalised p_c does not vary much from the base runs with below 0.2 % per minute less coverage. p_{mc} is relativley small with 25.3% in the full coverage case compared to the base mean p_{mc} is 33.8%.

The most unexpected results are the lower p_c compared to the p_c^*. This most likely results from the CCPP implementation already explained in the interpretation of

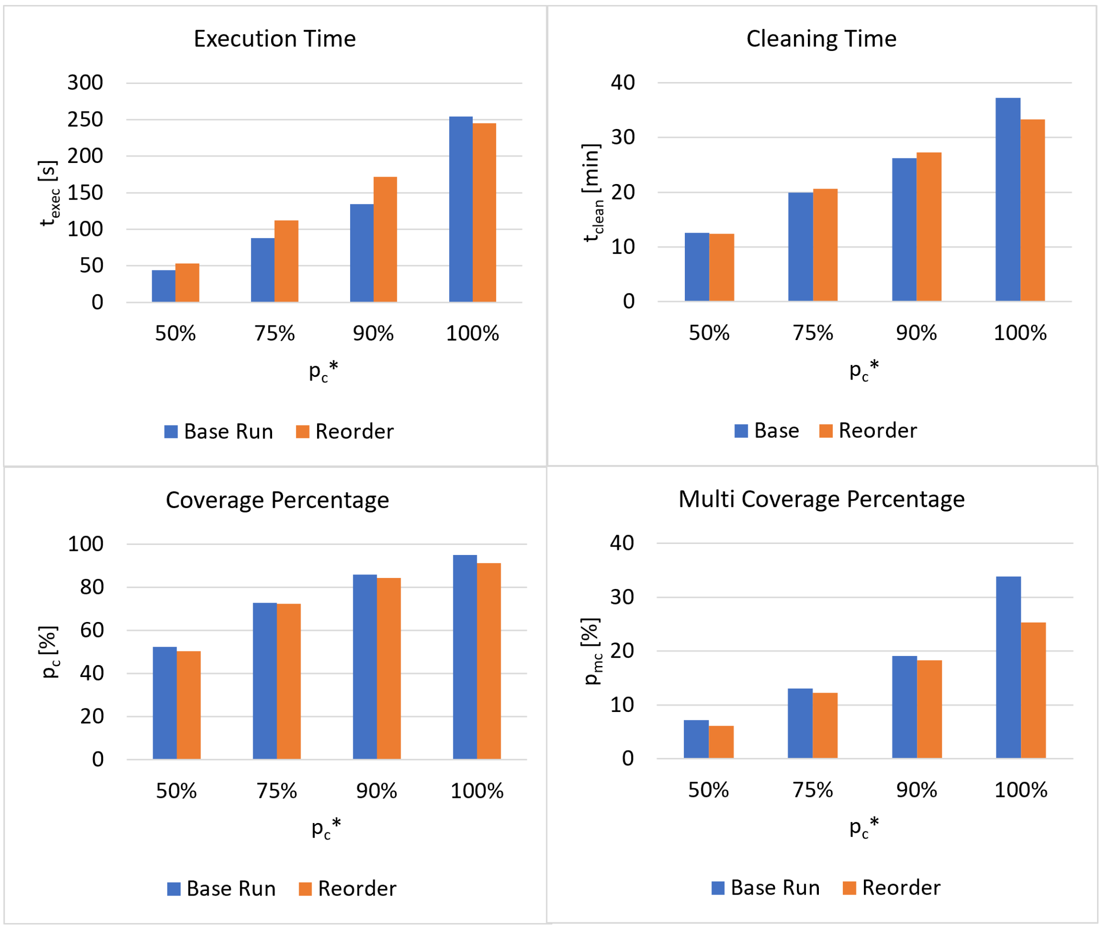


Figure 6.3: The comparison of metrics between the base runs with no obstacles, and the `Reorder` strategy with obstacles for different required coverage percentages (p_c^*) is shown

the reference results in Section 6.2. The `Reorder` strategy deletes already covered trajectory points from the trajectory list and only the paths between the segments are planned again. Therefore, the overall percentage can be reduced even more, as routes may be planned in already cleaned areas instead of uncleaned cells. This phenomenon is also more significant the higher the desired clean area is. The low p_{mc} could be explained by the overall smaller p_c, as for both the base runs as well as the `Replan` strategy p_{mc} is lower for lower coverage percentages.

6.2.3 Follow Strategies

For the `Follow` strategy, the combinations with both replanning and reordering are evaluated.

Reorder	t_{exec} [s]		t_{clean} [min]	
p_c^*	mean	std-dev	mean	std-dev
50%	53.28	13.12	12.42	1.56
75%	112.14	30.74	20.66	2.48
90%	171.38	30.58	27.27	2.43
100%	245.22	31.24	33.30	2.23

	p_c [%]			p_{mc} [%]	
p_c^*	mean	std-dev	norm [%/min]	mean	std-dev
50%	50.40	3.75	4.06	6.09	3.16
75%	72.38	3.87	3.50	12.26	4.00
90%	84.26	3.45	3.09	18.28	4.12
100%	91.17	2.57	2.74	25.31	3.92

Table 6.4: Reorder simulation results for different required cleaning percentages (p_c^*) in terms of execution time (t_{exec}), cleaning time (t_{clean}), covered percentage (p_c) and multi-coverage percentage (p_{mc})

Starting with the replanning combination `Follow-Replan`, the results are shown in Figure 6.4 and Table 6.5. t_{exec} and estimated t_{clean} are again, as expected, higher than for the base runs. `Follow-Replan` has extremely high execution times with 661.28 seconds for the full coverage and 317 seconds for 90% p_c^*. This is more than double the time needed in the base runs. These results, on the other hand, have a high deviation from the mean value, so the results very much depend on coincidental collisions caused by a specific trajectory plan, and a comparison is not very reliable. t_{clean} also increases; with 52 min, it takes 20 min longer to clean the area than without obstacle avoidance for a similar p_c. Additionally, the desired percentage in these cases is also not reached with 88% and 95% for p_c. The normalised p_c shows an increase of about 0.7 per cent per minute compared to the base run in all required p_c^*. Also, p_{mc} is a lot higher than in the base runs, which shows that combining `Follow` and `Replan` is not very well suited as a cleaning strategy.

A lot of computing time is needed to find a suitable trajectory plan for only a few free cells at the end of the cleaning process. The smaller the number of free cells gets, the more difficult it is to find a decomposition. The high deviation of t_{exec} and t_{clean} could be explained by taking the number of strategy executions into account. This varies notably from a minimum of 25 to a maximum of 241 collisions avoided. This could result from the random planning of the new decomposition. If the new path directly collides with the moving obstacle, a new collision is directly detected, and the planer will continue replanning until no collision is detected with the course. The time from waiting before the execution and planning the new path reduces the active cleaning time yet is included in

Follow-Replan p_c^*	t_{exec} [s] mean	t_{exec} [s] std-dev	t_{clean} [min] mean	t_{clean} [min] std-dev
50%	104.44	63.57	16.56	3.99
75%	193.41	69.92	26.91	4.53
90%	317.17	93.43	35.18	7.65
100%	661.28	208.89	52.72	10.53

p_c^*	p_c [%] mean	p_c [%] std-dev	p_c [%] norm [%/min]	p_{mc} [%] mean	p_{mc} [%] std-dev
50%	57.70	8.12	3.48	10.60	6.51
75%	78.88	3.57	2.93	19.13	5.60
90%	88.70	15.11	2.52	27.75	5.64
100%	95.84	6.66	1.82	41.29	7.15

Table 6.5: Follow-Replan simulation results for different required cleaning percentages (p_c^*) in terms of execution time (t_{exec}), cleaning time (t_{clean}), covered percentage (p_c) and multi-coverage percentage (p_{mc})

both time metrics. Similar to the `Replan` strategy, the paths are extensively covered multiple times. The path of the followed obstacle still is crossed often, and as a new trajectory plan has to be found, each time a collision is detected, the segments get smaller and smaller. Overall this strategy is very unreliable and influenced by the randomness of the CCPP algorithm with high deviations from the mean, especially in the high p_c^* cases.

Follow-Reorder p_c^*	t_{exec} [s] mean	t_{exec} [s] std-dev	t_{clean} [min] mean	t_{clean} [min] std-dev
50%	94.22	44.01	19.32	6.17
75%	320.85	85.47	40.69	6.85
90%	458.35	163.20	42.16	3.02
100%	347.40	50.98	42.88	2.62

p_c^*	p_c [%] mean	p_c [%] std-dev	p_c [%] norm [%/min]	p_{mc} [%] mean	p_{mc} [%] std-dev
50%	68.61	13.46	3.55	12.48	7.02
75%	95.24	5.07	2.34	36.37	7.69
90%	97.09	1.63	2.30	38.20	4.39
100%	96.61	1.97	2.25	40.60	3.82

Table 6.6: Follow-Reorder simulation results for different required cleaning percentages (p_c^*) in terms of execution time (t_{exec}), cleaning time (t_{clean}), covered percentage (p_c) and multi-coverage percentage (p_{mc})

For the `Follow-Reorder` combination, the results are very different from the other strategies. The results can be seen in Table 6.6 and Figure 6.5. t_{exec}, as

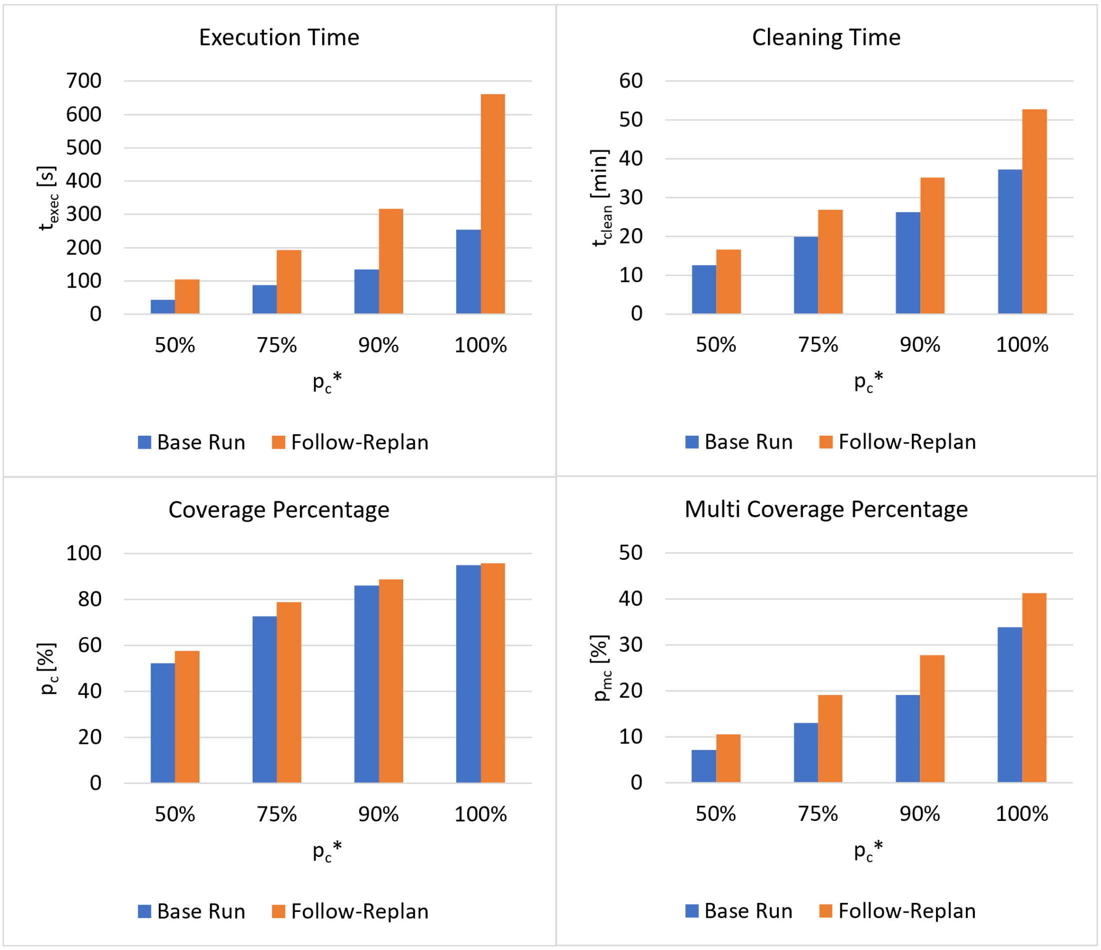


Figure 6.4: The comparison of metrics between the base run with no obstacles, and the `Follow-Replan` strategy with obstacles for different required coverage percentages (p_c^*) is shown

well as t_{clean}, are significantly higher than the base runs, especially for the results in 75% and 90% p_c^*. `Follow-Reorder` takes 458 s to execute the simulation and 42 min to clean the area with p_c^* 90%; the base runs for comparision is at 134 s t_{exec} and 26 min t_{clean}. This is related to the unexpectedly high p_c. The p_c for the simulation with 50% p_c^* is already at 68% and for the higher p_c^* simulations, all reach a similar percentage above 95%. The highest p_c results from 90% p_c^*. This behaviour also affects the other statistics t_{clean} and p_{mc}. Worth mentioning is also the deviation for the coverage percentages. In the 50% and 75% cases, the deviations are especially high.

The high cleaning percentage in this strategy combination could be explained through the mix of replanning and reordering already shown in Figure 5.9b. After the follow execution was successful, the single replan order results in a higher p_c,

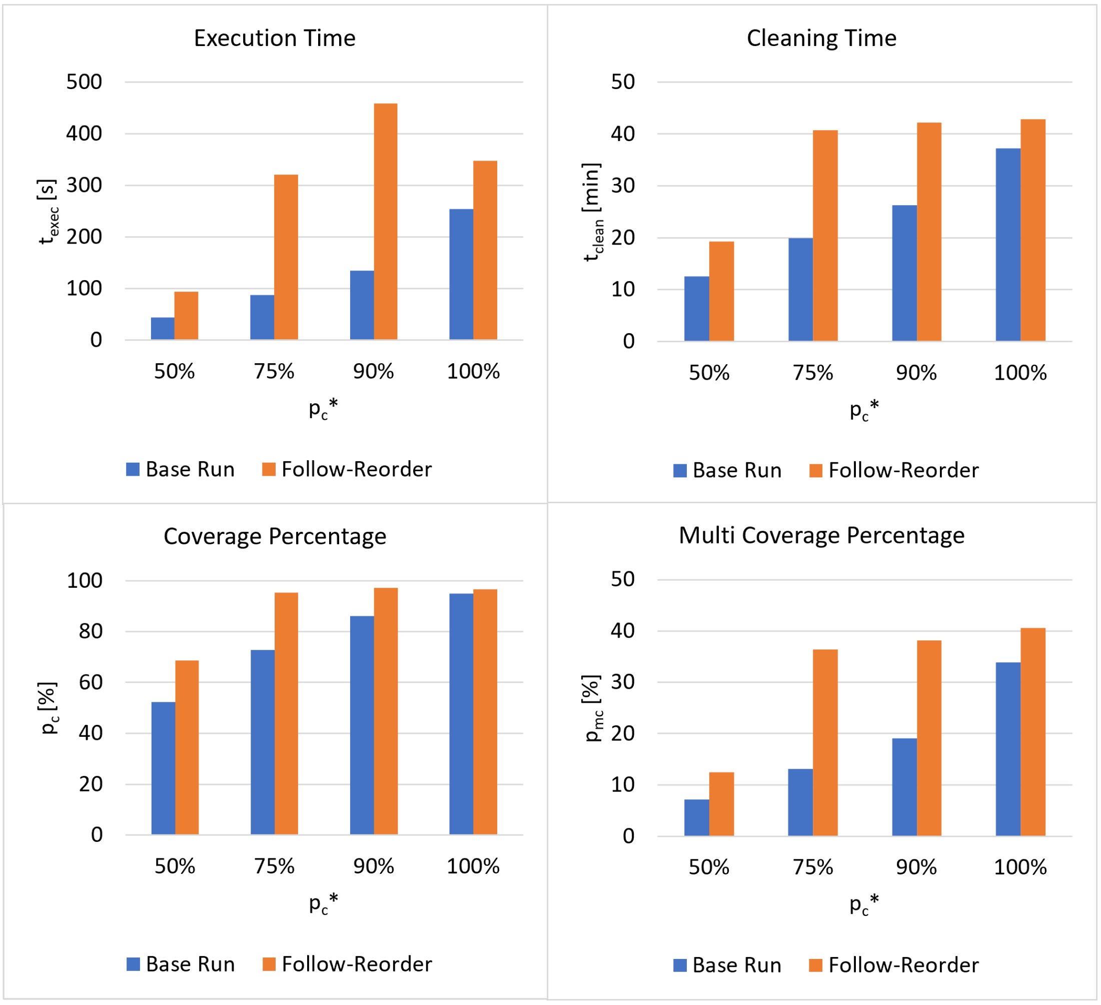
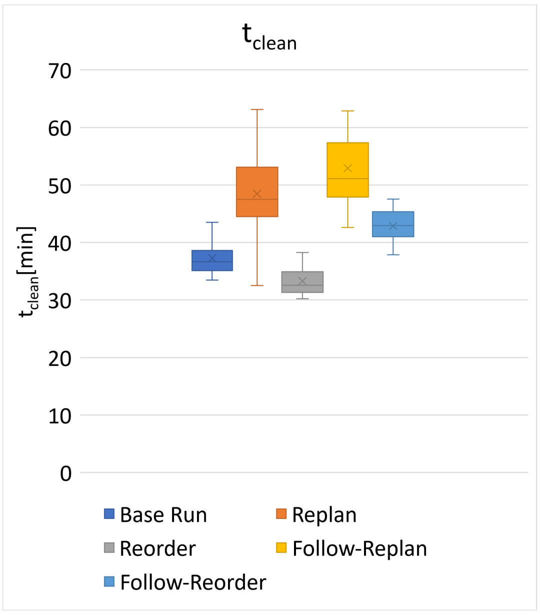


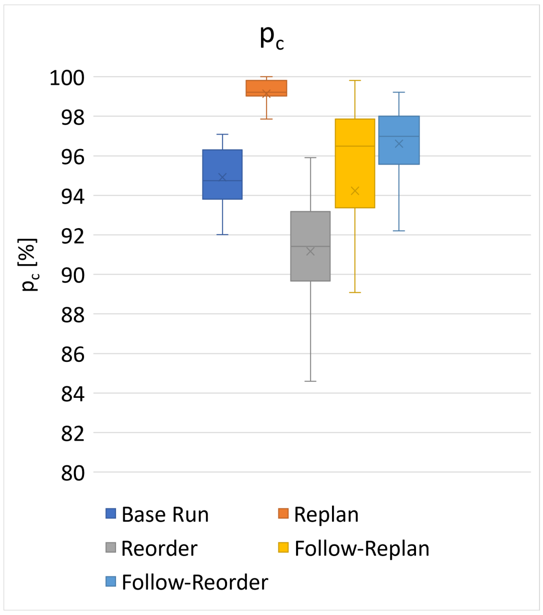
Figure 6.5: The comparison of metrics between the base runs with no obstacles, and the `Follow-Reorder` strategy with obstacles for different required coverage percentages (p_c^*) is shown

which is explained in Section 6.2.1. In the `Follow-Replan` strategy discussion the results from high p_c are already explained by the small segments planned at the end of the process. In `Follow-Reorder`, however, reordering the remaining segments leads to an overall better p_c as no additional area is planned after the first replan. Nevertheless, as the p_c^* completeness check is only performed when all trajectory points are reached, or a collision is detected, `Follow-Reorder` also reaches high p_c. The high deviations in the 50% case could result from the different timing of detecting a new collision. The cleaning stops when a collision is detected and p_c^* is reached. This can happen sooner or later, depending on the planned path. As the mean p_c is much higher than p_c^*, this more often happens later than sooner resulting in high deviations. The closer the p_c is to p_c^*, the smaller the deviation gets.

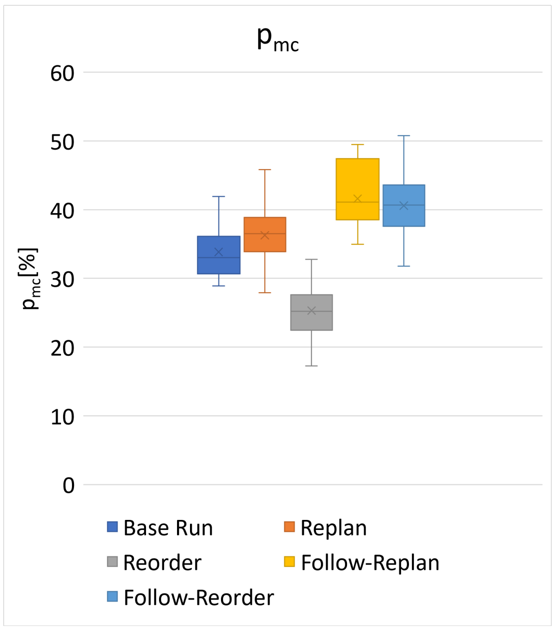
6.3 Comparison of Strategies



(a) cleaning time t_{clean}



(b) coverage percentage p_c, showing the relevant sector 80% to 100%



(c) multi-coverage p_{mc}

Figure 6.6: These box plots show the comparison of metrics between the implemented strategies for 100% required coverage percentage

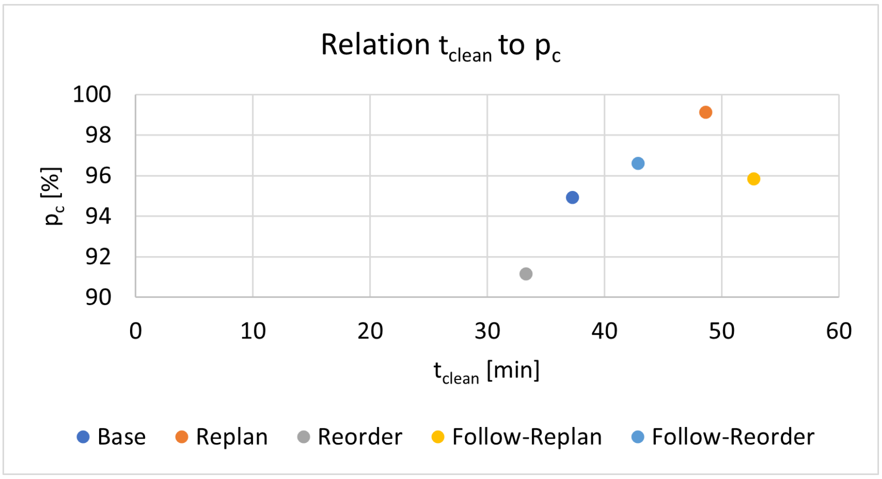
100% p_c^*	t_{exec} [s]		t_{clean} [min]		p_c [%]		p_{mc}[%]	
	mean	std-dev	mean	std-dev	mean	std-dev	mean	std-dev
Base	253.83	34.98	37.25	2.67	94.92	1.38	33.84	3.43
Replan	436.24	101.42	48.63	6.46	99.12	0.77	36.55	5.85
Reorder	245.22	31.24	33.30	2.23	91.17	2.57	25.31	3.92
F-Replan	661.28	208.89	52.72	10.53	95.84	6.66	41.29	7.15
F-Reorder	347.40	50.98	42.88	2.62	96.61	1.97	40.60	3.82

100% p_c^*	normalised p_c [%/min]
Base	2.55
Replan	2.04
Reorder	2.74
Follow-Replan	1.82
Follow-Reorder	2.25

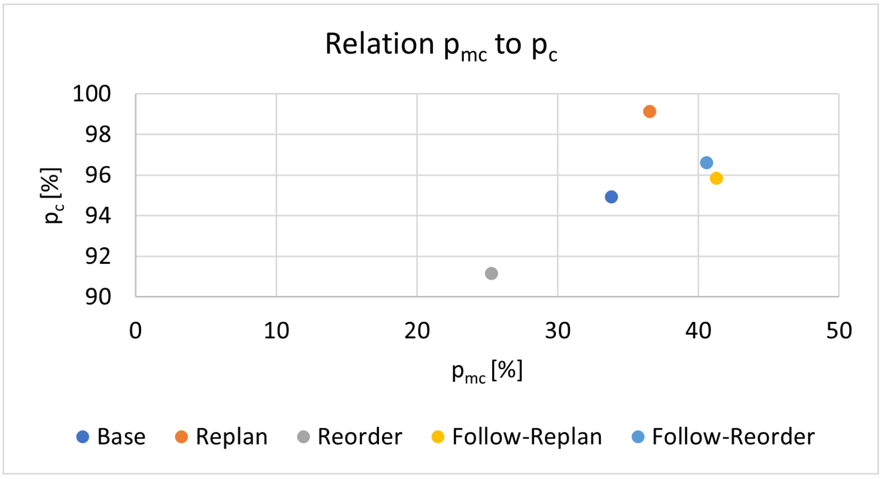
Table 6.7: Camparison of simulation results from all strategies including the reference runs, showing the metrics execution time (t_{exec}), cleaning time (t_{clean}), covered percentage (c_p) and multi-coverage percentage (p_{mc})

The Box plots in Figure 6.6 show the three comparable metrics of t_{clean}, p_c, and p_{mc} for each strategy with 100% p_c^*. The mean and normalised values of the runs are shown in Table 6.7. `Replan` has the best results in terms of coverage. However, t_{clean} is higher than for the other strategies, only topped by the `Follow-Replan` strategy. Regarding p_{mc}, the results of `Replan` are good compared to both `Follow` strategies. Only `Reorder` has better results, even lower than the reference runs. The second-best coverage percentage is achieved by the `Follow-Reorder` strategy. This strategy is also competitive in terms of t_{clean}, yet it has the second highest p_{mc}. `Follow-Reorder`'s high p_{mc} is only topped by `Follow-Replan`, which does not perform well in any other metric also. The `Reorder` strategy alone performs well for p_{mc} and t_{clean}, yet p_c is lower than the base runs, which equalises the good results from the other criteria. However, the cause for this behaviour could be resolved with a different segmentation and planning algorithm. The `Follow-Replan` strategy is not reliable enough, as the variation in the results shows. Therefore this strategy will not be discussed further as an optimal cleaning solution.

To be able to compare the strategies better, the relations of t_{clean} and p_c in Figure 6.7a and p_{mc} and p_c in Figure 6.7b are shown. In Figure 6.7a, the optimal solution would be at the top left with the highest coverage and minimum t_{clean}. The two strategies worse than the base are `Reorder` and `Follow-Replan`. `Reorder`'s lower cleaning percentage means that the requirements are not fulfilled and therefore the strategy is rated lower than the other strategies. The other worse strategy is `Follow-Replan` with 95.8% p_c in 52 min resulting in 1.82 %/min. The results from this strategy also deviate from the average relation



(a) relation cleaning time t_{clean} to coverage percentage p_c



(b) relation multi-coverage p_{mc} to coverage percentage p_c

Figure 6.7: Box plot showing the relations between metrics for the introduced simulation environment and full coverage of all collision avoidance strategies

and are slower compared to the other strategies. `Follow-Reorder` and `Replan` both have similar relations to the reference with `Replan` resulting in a higher p_c. `Follow-Reorder` covers 96.6% in 42 min resulting in 2.25 %/min, while `Replan` covers 99% in 49 min resulting in 2.04 %/min. The normalised values can also be seen in Table 6.7. The higher the priority of t_{clean} is set by the user, the better the results of `Follow-Reorder` are.

The upper left corner would be the optimum in Figure 6.7b again. The two strategies `Reorder` and `Follow-Replan` once more have worse performance compared to the base runs. `Replan` achieves a higher p_c to p_{mc} ratio. `Follow-Reorder` with similar results to base. The best choice again results from the user's wishes. If a minimum p_{mc} is highly prioritised `Replan` is the best choice.

Concluding, the combination of the CCPP algorithm not reaching the required percentage and `Reorder` strategy are not achieving good results and regarding the pending requirement FR_S2 this is not an efficient collision avoidance strategy. `Follow-Replan` achieves slightly better results yet is still not very efficient. The two overall best-performing strategies are `Follow-Reorder` and `Replan`. If an almost perfect coverage is desired the Replan strategy is the best choice, as the strategy achieves the highest coverage out of the proposed strategies and has a relatively low multi-coverage percentage. On the other hand, if the time efficiency is more relevant to the user `Follow-Reorder` is the best strategy. This strategy also ensures full coverage of the fleet's path even for small p_c^*.

7 Conclusion and Outlook

The objective of this work was to develop a strategy for cleaning robots in an environment with an active mobile robot fleet. The necessary requirements resulting from this problem were formulated and a structure for the system could already be derived from the requirements formulation by separating the solution into collision avoidance and path planning. The software architecture was designed accordingly in three section. Firstly, the path planning module implements the Complete Coverage Path Planning (CCPP) algorithm. Secondly, the collision avoidance module combines the detection node, the collision point calculation node, and execution by the strategy handler. Lastly the execution of the software is done by the local planner. During the implementation phase, the specified architecture was implemented and a simulation for executing and testing the cleaning solution was built. The result was a Robot Operating System (ROS) node which decomposes a given map of the surrounding area into rectangular segments, that are planned with back-and-forth motion. In between the segments, routes to and from the segments are planned. In parallel, the collision detector interprets the scan data and calculates collision points.

Additionally the four different collision avoidance strategies, `Replan`, `Reorder`, `Follow-Replan`, and `Follow-Reorder` were discussed as well as their implementations described. The implemented strategies were all evaluated regarding the metrics cleaning time, achieved coverage percentage, and multi-coverage percentage and compared to a reference run with no moving obstacle avoidance. The discussion of the results revealed the two best-performing strategies `Follow-Reorder` and `Replan`. While `Follow-Reorder` achieves a better percentage per cleaning time and ensures a cleaning of the fleet's path when a collision is detected, `Replan` accomplishes a higher coverage percentage overall and shows less multi-coverage. The `Reorder` strategy instead does not reach the required cleaning percentage and is therefore not suitable. `Follow-Replan` is very inefficient and takes almost double the time to clean the same percentage as the reference.

During the evaluation of the system, the insufficient coverage of the implemented CCPP algorithms was discovered. Still, the results from the strategies are considered valid at least in comparison to each other, as they were all tested and compared using the same planning algorithm. Considering the research question posed in Chapter 2 of whether the usage of prior information in planning is better than using an online algorithm seems wrong as the best-performing strategies both use a complete replanning process and can therefore not be considered offline. However, the evaluated results still leave room for improvements and some ideas for future work based on optional objectives from Chapter 2 and the evaluation in Section 6.2 will be given.

The most significant improvement can probably be made by using a different CCPP algorithm. New algorithms would be relatively easy to integrate into the developed ROS package as long as they deliver a trajectory plan fulfilling the restrictions of the implemented controller. Additionally, in the implementation of the trajectory planner, the given segments should be ordered more efficiently. Currently, the paths between the segments are relatively long; therefore, a lot of the multi-coverage percentage comes from this planning. This improvement would lead to better results immediately. Furthermore, introducing a cleaning fleet compared to just a single cleaning robot would be beneficial, especially for huge areas where the cleaning time is very long. The introduction is straightforward, as the segments can be matched to a region that will be covered by a selected robot in the cleaning fleet. Another idea resulting from the evaluation results would be introducing multiple cleaning robots not communicating with each other and setting a small required cleaning percentage per cleaning robot. With the random factor and different starting points the planned path may not overlap much and a better result could possibly be achieved. For more extensive evaluation, more advanced environments could be integrated into the simulation, for example, the Gazebo worlds from the Fraunhofer package cob_environments [1], which include a house or industrial environment. Then additionally a comparison to the implementations of the proposed Coverage Path Planning (CPP) algorithms from Bormann et al., 2018 could be made, as their evaluation was made with the cob_environmets package.

Additional strategies should also be considered for implementation. These include speeding up when the cleaning robot is currently on the fleet's path and moving out of the way proactively or speeding up and stopping to clean the already cleaned areas on the way to a new segment. Additionally, circumnavigating the obstacle could be considered as well, for example by using the dynamic window approach. A disadvantage might be, that the cleaning robot leaves the originally planned path for this and continue to follow the planned trajectory. The robot later would have to return to a position where most of the area is already cleaned. Therefore the multi-coverage percentage would increase, decreasing the

[1]https://github.com/ipa320/cob_simulation

overall efficiency. Another way to improve the strategies is to consider the fleet's routes in the planning algorithm right from the beginning and, alternatively, to remember the locations from the collision information. The consideration could be implemented by blocking these paths for trajectory planning completely and therefore get a safer cleaning process and reduce interruptions of the original cleaning path. The path planned might get longer, but the planning could be done more efficiently with this information, as replanning is costly. Moreover, as seen in the evaluation, the more area was cleaned prior to the replanning process, the more inefficient the planned path got.

Overall this work showed, that with the input of the map and a LiDAR mounted on the cleaning robot, the system can successfully navigate a robot through the given environment. The cleaning strategies provide a solution that does not disturb the surrounding fleet without communication necessary, so the system is designed to work with all types of fleets.

Appendices

A Defined Topics and Services

The topics the collision avoidance node is publishing it's results to and services necessary for the communication between the local planner and collision avoidance node.
Topics:

/predictions/collisions
/predictions/velocity
/trajectories

ROS Services:

collision
resolve collision
stop
replan
reorder
follow

B Full Coverage Decomposition

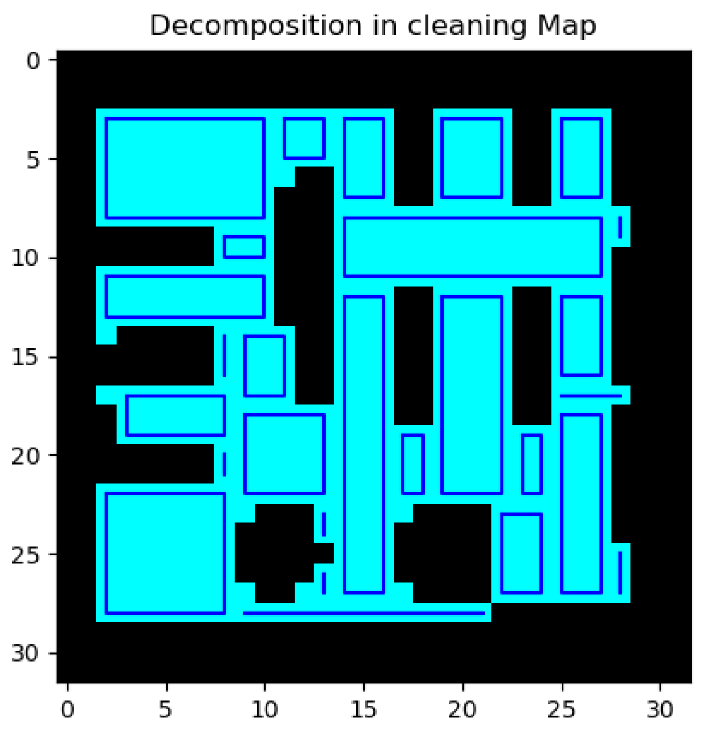


Figure 1: Full Coverage Decomposition: Regular decomposition shown in the initialized internal map. Only not planned cells are the single cells too small for a segment , resulting in p_c=98.5% with 8 unplanned cells [(6, 11), (14, 2), (17,2), (23, 9), (23,17), (27, 9), (27,12), (27, 17)] and 520 initial reachable cells

References

Acar, E., Choset, H., Rizzi, A., Atkar, P., & Hull, D. (2002). Morse decompositions for coverage tasks. *I. J. Robotic Res.*, *21*, 331–344. https://doi.org/10.1177/027836402320556359

Akkaya, S. B., & Gökçe, M. A. (2022). Intelligent scheduling and routing of a heterogenous fleet of automated guided vehicles (agvs) in a production environment with partial recharge. In C. Kahraman, A. C. Tolga, S. Cevik Onar, S. Cebi, B. Oztaysi & I. U. Sari (Eds.), *Intelligent and fuzzy systems* (pp. 568–576). Springer International Publishing.

Azarm, K., & Schmidt, G. (1997). Conflict-free motion of multiple mobile robots based on decentralized motion planning and negotiation. *Proceedings of International Conference on Robotics and Automation*, *4*, 3526–3533 vol.4. https://doi.org/10.1109/ROBOT.1997.606881

Bellmore, M., & Nemhauser, G. L. (1968). The traveling salesman problem: A survey. *Operations Research*, *16*(3), 538–558.

Bormann, R., Hampp, J., & Hagele, M. (2015). New brooms sweep clean - an autonomous robotic cleaning assistant for professional office cleaning. *Proceedings - IEEE International Conference on Robotics and Automation*, *2015*, 4470–4477. https://doi.org/10.1109/ICRA.2015.7139818

Bormann, R., Jordan, F., Hampp, J., & Hägele, M. (2018). Indoor coverage path planning: Survey, implementation, analysis. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 1718–1725. https://doi.org/10.1109/ICRA.2018.8460566

Choset, H. (2001). Coverage for robotics – a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, *31*, 113–126.

Choset, H., & Pignon, P. (1998). Coverage path planning: The boustrophedon cellular decomposition. In A. Zelinsky (Ed.), *Field and service robotics.*

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, *4*(1), 23–33. https://doi.org/10.1109/100.580977

Fragapane, G., de Koster, R., Sgarbossa, F., & Strandhagen, J. O. (2021). Planning and control of autonomous mobile robots for intralogistics:

Literature review and research agenda. *European Journal of Operational Research*, *294*(2), 405–426. https://doi.org/https://doi.org/10.1016/j.ejor.2021.01.019

Gabriely, Y., & Rimon, E. D. (2001). Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of Mathematics and Artificial Intelligence*, *31*, 77–98.

Galceran, E., & Carreras, M. (2013). A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, *61*, 1258–1276.

Hsu, P., & Lin, M., CL. an Yang. (2014). On the complete coverage path planning for mobile robots. *J Intel Robot Syst*, (74), 945–963.

Huang, W. (2001). Optimal line-sweep-based decompositions for coverage algorithms. *IEEE International Converence on Robotics and Automation*, *1*, 27–32.

Jan, G. E., Luo, C., Hung, L.-P., & Shih, S.-T. (2014). *A computationally efficient complete area coverage algorithm for intelligent mobile robot navigation.* https://doi.org/10.1109/IJCNN.2014.6889862

Latombe, J.-C. (2012). *Robot motion planning* (Vol. 124). Springer Science & Business Media.

Lumelsky, V., Mukhopadhyay, S., & Sun, K. (1990). Dynamic path planning in sensor-based terrain acquisition. *IEEE Transactions on Robotics and Automation*, *6*(4), 462–472. https://doi.org/10.1109/70.59357

Luo, C., & Yang, S. X. (2008). A bioinspired neural network for real-time concurrent map building and complete coverage robot navigation in unknown environments. *IEEE Transactions on Neural Networks*, *19*(7), 1279–1298. https://doi.org/10.1109/TNN.2008.2000394

Oh, J. S., Choi, Y. H., Park, J. B., & Zheng, Y. (2004). Complete coverage navigation of cleaning robots using triangular-cell-based map. *IEEE Transactions on Industrial Electronics*, *51*(3), 718–726. https://doi.org/10.1109/TIE.2004.825197

Seder, M., & Petrovic, I. (2007). Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 1986–1991. https://doi.org/10.1109/ROBOT.2007.363613

Viet, H., Dang, V., & Laskar, M. (2013). Ba*: An online complete coverage algorithm for cleaning robots. *Appl Intell*, *39*, 217–235. https://doi.org/10.1007/s10489-012-0406-4

Yan, Z., Schreiberhuber, S., G, H., & et. al. (2020). Robot perception of static and dynamic objects with an autonomous floor scrubber. *Intelligent Service Robotics*, *13*, 403–417. https://doi.org/10.1007/s11370-020-00324-9

Yang, S., & Luo, C. (2004). A neural network approach to complete coverage path planning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *34*(1), 718–724. https://doi.org/10.1109/TSMCB.2003.811769

Zelinsky, A., Jarvis, R. A., Byrne, J., & Yuta, S. (1993). Planning paths of complete coverage of an unstructured environment by a mobile robot. *13*, 533–538.