# Improving safety and performance for mobile robot navigation using self-adaptation in retail environments

Master thesis

by

Jasper Wijkhuizen

to obtain the degree of Master of Science
at the Delft University of Technology,

Student number: 4214846

| Committee: | | |
|---|---|---|
| | C. Hernandez Corbato | Supervisor |
| | D. Bozhinoski | Daily supervisor |
| | J. Kober | Chair |


# Abstract

There is a growing demand for autonomous mobile robots in industry. The robots need to solve, among other things, the problem of navigation. Since the robots operate in semi-structured and (partially) unknown environments, the local path planning sub-problem has received a lot of attention from researchers. The result is a large variety of algorithms to solve this sub-problem, which are often developed and optimized for a specific scenario of environment. This gives the robots a large set of different possible system configurations. Traditionally, the engineer makes the choice at design-time about which system configuration to use. However, there is a demand for more autonomy in robotic systems, and it is therefore desired to have the robot make this choice at run-time. Self-adaptive systems are typically used to switch between system configuration at run-time. The aim of this research is to improve the knowledge of the system about the navigation quality in different environments. This knowledge will help the self-adaptive system to make a better informed decision about adaptation. The contribution of this research is an approach to construct this knowledge for the system. First, metrics for navigation quality in terms of safety and performance and relevant environment characteristics in relation to the navigation quality are developed. Machine learning methods are used to train quality models that can be used to predict the safety level of a system configuration based on the measurements of the environment characteristics. These quality models are implemented in the MROS self-adaptive framework, and experiments showed that the use of quality models in this framework is useful to make better informed adaptation choices and and improves the navigation quality.

# Contents

**Abstract** i

**1 Introduction** 1
- 1.1 Motivation . . . 1
- 1.2 Research question . . . 2
- 1.3 Thesis structure . . . 2

**2 Background: Mobile robot navigation and self-adaptation** 3
- 2.1 The navigation problem . . . 3
- 2.2 Local path planning algorithms . . . 4
- 2.3 Self-adaptation in robotics . . . 7
- 2.4 Summary . . . 9

**3 Research approach** 10
- 3.1 Navigation quality definition . . . 10
- 3.2 Measuring environment characteristics . . . 11
- 3.3 Prediction using quality models . . . 11
- 3.4 Self-adaptive framework implementation . . . 11

**4 Quality and Environment metric development** 12
- 4.1 Related work on quality metrics . . . 12
- 4.2 Quality metric development . . . 13
- 4.3 Environment characteristics metric development . . . 15
- 4.4 Tests and validation . . . 17
- 4.5 Discussion . . . 21

**5 Quality model construction** 23
- 5.1 Machine Learning theory . . . 23
- 5.2 Data collection and preparation . . . 24
- 5.3 Model type choice . . . 25
- 5.4 Training and evaluation of the models . . . 26
- 5.5 Hyperparameter tuning . . . 26
- 5.6 Quality model validation . . . 26
- 5.7 Discussion . . . 27

**6 Self-adaptive framework** 29
- 6.1 MROS, default framework . . . 29
- 6.2 MROS, modifications . . . 30
- 6.3 Reasoning cycle modification . . . 31
- 6.4 Configuration construction . . . 33
- 6.5 Discussion . . . 34

**7 Experiments** 37
- 7.1 Experimental setup . . . 38
- 7.2 Experiment results . . . 40
- 7.3 Discussion . . . 42

**8 Conclusions and future work** 43
- 8.1 Conclusions . . . 43
- 8.2 Future work . . . 44

**Appendices** 46

**A Project description** 47

**B RoQME, an experience report** **49**

**C Experiment design** **51**

**D Default local planner configurations** **54**

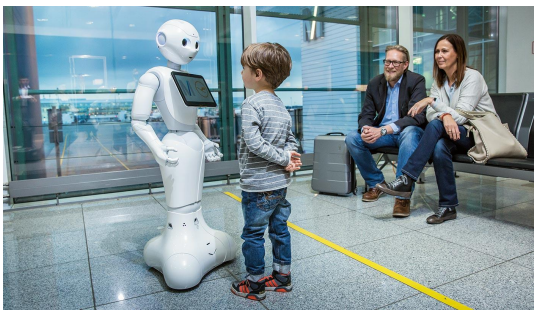
# 1 Introduction

## 1.1. Motivation

Mobile robots are used in many applications in industry these days to improve the level of productivity; in warehouses, factories, retail stores, and more. In the example of the warehouses, the environment is structured, static, and fully known for the robot. Next to that, the robots are often isolated from humans by restricting human access to the working area of the robot or by separate motion lanes [1]. This separation of work-spaces is done to simplify the navigation problem for the robot and to ensure the safety of the humans and the robots. A good example where robots are isolated from humans is the Kiva robot from Amazon (see Figure 1.1a). In the Amazon warehouses, robots pick up the shelves on request and bring them to a human co-worker, which in turn grabs the requested products and packs them for shipping [2]. In these warehouses, there is a clear separation between the work-spaces of humans and robots.

However, there is a growing demand for autonomous mobile robots in more complex environments. Retail stores are environments which can be a semi-structured, where obstacles such as boxes, crates and other obstacles show up in new (unknown) places. The working area of the robot can also be shared with humans, which makes the problem of navigation even more challenging since the movement of humans can be very unpredictable [3]. Next to that, the humans should not be worried about the robots driving around them. Ensuring the safety of the robot and the humans should be focused on in these environments. An example of a robot operating in an environment shared with humans is the service robot Pepper. This robot is for example used in a four week trial at Munich airport, to help people find their way [4] (see Figure 1.1b). A crowded environment like an airport is makes the navigation problem challenging. In this trial however, the robot was not driving around but stayed at one location.

(a) The Kiva robots in an Amazon Warehouse, picture from [5].



(b) The Pepper robot, at the airport of Munich, picture from [4].

Figure 1.1: Examples of mobile robots in different environments.

Mobile robots in industry need to solve, among others, the challenges of navigation, which is one of the main tasks for mobile robots [6]. The robot has to drive from its initial position to the goal position, and avoid collisions with static or dynamic obstacles on the way. A lot of research is done on mobile robot navigation, resulting in many different algorithms to solve parts of the navigation problem [7]. Each algorithm has its own advantages and disadvantages, and their parameters are usually optimized for a specific scenario or environment type [8]. This means that there can be a different (optimal) system configuration for each scenario or environment type.

Since navigation in different environments can be solved with different system configurations, the variability of the system is large. This large amount of variability is characteristic for robotic applications [9]. When the robot encounters a new environment, a choice has to be made about which algorithm to use. These choices are traditionally made by the engineer at design time [10]. But since there is a demand for more autonomy in robotic systems, it is desired to have this choice made by the robotic system itself. Self-adaptive systems are typically used for these situations where switching between the alternative configurations is needed [11].

Traditionally, fulfilling the functional requirements is the primary focus for robotic systems, but balancing the non-functional properties is becoming more important as well [12]. Functional properties define *what* a system does, while in contrast non-functional properties define *how* a system performs. Non-functional properties can be measured using quality attributes, such as performance, robustness, or safety [13]. The different system configurations can differ in some quality attributes, and can be used as selection criteria for choosing between the configurations [11].

## 1.2. Research question

Since the navigation problem can be solved with different algorithms and parameters, there are a lot of different possible system configurations for a mobile robot. The configurations can differ in the quality attributes, and for different environments different configurations can be preferred. A self-adaptive system can be used to manage the different configurations at run-time.

This research is done in the Cognitive Robotics research group at the TU Delft. In this research group, a self-adaptive framework is developed: the MROS framework [14]. This framework will be used in this research as a tool to obtain the self-adaptive capabilities.

The main research question for this thesis is:
*How can navigation quality be improved for mobile ground robots based on changes in the environments using a self-adaptive framework?*

## 1.3. Thesis structure

The structure of this document is the following: first, background on the navigation problem for mobile robots and self-adaptation for robotic systems are explained in chapter 2. Next, the approach of this research is explained in chapter 3 together with the research sub-questions. In chapter 4, metrics for navigation quality and environment characteristics are developed. Chapter 5 is about machine learning and the quality model construction. These quality models are then implemented together with the other new components into the self-adaptive framework in chapter 6. Experiments are setup up and the results are analysed in chapter 7. The research (sub)questions are answered in chapter 8, where a conclusion is drawn and recommendations for future work are proposed.

# 2

# Background: Mobile robot navigation and self-adaptation

To be able to improve navigation quality for a mobile robot, the general navigation problem and the different algorithms to solve the navigation problem needs to be understood. This chapter discusses the general navigation problem and its sub-problems. After that, the local planning sub-problem is analysed further and different algorithms are analysed.

Self-adaptation will be used to improve the navigation quality in this research. To understand the concept of self-adaptive systems, relevant literature of self-adaptive systems for robotics will be analysed. Different strategies and examples from literature are treated.

## 2.1. The navigation problem

The fundamental problem of mobile robot navigation is to plan and follow a collision-free trajectory from the initial position to the goal position. The navigation problem can be divided into four sub-problems [8][6]: Perception, Localization and Mapping, Path planning, Motion control (see Figure 2.1).

Successful navigation requires success at all of the four building blocks of navigation [6]. Perception is the process of interpreting the data from sensors to extract useful information. Localization and mapping is about building a map of the environment and determining the position of the robot relative in that map. Next, this map is used to plan a global and later a local path to the goal position without collisions with static and/or dynamic obstacles. In the last building block, motion control, the actuators of the robot are controlled to follow the planned path.

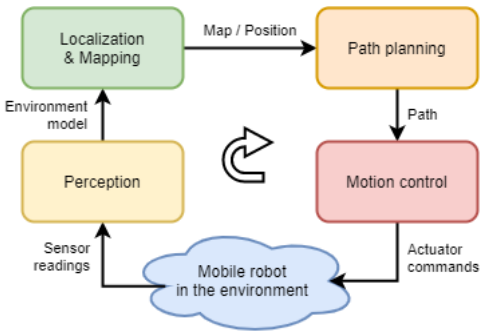



Figure 2.1: The general navigation structure for a mobile robot.

### 2.1.1. Perception

Robots are equipped with various sensors to extract measurements of their environment and the state of the robots itself. These measurements are used to obtain useful information, which is needed for the localization & mapping, path planning, and the motion control sub-problems.

Proprioceptive sensors are used to get information about the robot itself, typical examples are Inertial Measurement Units or wheel encoders [6]. On the other hand, exteroceptive sensors are taking measurements of the environment, such as cameras, lidars, and ultrasonic distance sensors.

### 2.1.2. Localization and Mapping

Localization is the process in which the robot determines its relative position in the world. Mapping on the other hand, is the process that creates a map of that world. For a mobile robot to be able to navigate successfully, a map of the environment and the location of the robot relative to that map needs to be (partially) known. When robots need to operate in (fully) unknown environments, localization is done at the same time as the mapping [15].

### 2.1.3. Path Planning

Path planning identifies a collision-free trajectory from an initial position to a goal position. Path planning is usually divided into global planning and local planning [6][8][16]. Global path planning is done assuming a fully static environment and is about determining the most optimal obstacle-free path to the goal position. While local path planning can take a dynamic environment into account. Since local path planning is often much more computational demanding, only the direct surroundings of the robot (a local map which is smaller than the global map) are taken into account. In this local map, fresh sensor readings are used to update the (local) map and to plan a new collision free path to the (local) goal position.

Local path planning has received more attention from researchers in recent times due to the higher demand for autonomous robots in dynamic and unknown environments [17]. The results of this active research field is a large range of different algorithms to solve the local path planning. Local path planning will therefore be the focus of this research as well.

## 2.2. Local path planning algorithms

Popular local planners from literature are identified and their performance in different environments are analysed. This is done to be able to make a choice for relevant local planning methods for the configurations of the self-adaptive system in retail environments for this research.

### 2.2.1. Popular algorithms in different environments.

Kulich, Kozák, and Přeučil [16] compare three local planning algorithms and state that those three are the most used algorithms for obstacle avoidance nowadays; the Dynamic Window Approach, Vector Field Histogram plus, and the Smooth Nearness Diagram. According to Mohanan and Salgoankar [18], the most active areas of research in local path planning are about Artificial potential field methods and velocity space-based methods. They mention various examples of those methods, of which the most relevant methods are the Dynamic Window Approach and Velocity Obstacles. Özdemir and Sezer [19] divide local planning algorithms into directional approaches and velocity-based approaches and mention a list of popular algorithms in those categories; Vector Field Histogram plus, Nearness Diagram, Follow the Gap, Dynamic Window Approach, Velocity Obstacles. Sanchez [20] uses in his research the algorithms which are available in the ROS navigation stack, a framework that is widely used by the robotics community. The algorithms implemented in the ROS navigation stack are: the Dynamic Window approach and the Timed Elastic Bands method. From these researches, the following list of popular local planners is constructed: Dynamic Window Approach (DWA), Velocity Obstacles (VO),

Timed Elastic Bands (TEB), Vector Field Histogram (plus) (VFH+), Smooth Nearness-Diagram (SND), and the Follow the Gap Method (FGM).

Kulich, Kozák, and Přeučil [16] did a comparison study to compare the DWA, SND and VFH+ methods. Their conclusion is that in narrow environments, the SND method is slightly better than the DWA method and the performance of the VFH+ method was insufficient. In wide open environments, the performance of the DWA method and the VFH+ method was equal and both were outperforming the VFH+ method. Sanchez [20] states that TEB performs better than DWA in their experiment, which is an environment with narrow corridors.

Next to the comparison studies, the characteristics of the algorithms can be used to conclude something about their behavior in different environments (see appendix **??** for the detailed analysis). The VO method takes the velocity of obstacles into account, and therefore this method is well suited for dynamic environments. And this is the only method which actually uses velocity information of dynamic obstacles. Because the TEB method takes a larger part of the global path into account, and not only the next steering command like other methods, the performance of this method will be better in cluttered environments than others. The FGM method searches for the maximum gap between obstacles to navigate to. This will result in safe travel, even when there are many obstacles between the robot and the goal. This does reduce the performance, but increases the safety. Therefore this method can be useful in cluttered environments.

In the ROS Navigation Stack, there are two local planners implemented: the DWA and the TEB. These methods differ in their performance in narrow, wide, and cluttered environments according to the comparison studies and the analysis of their characteristics. When these local planners are used to construct the configurations for the metacontroller to choose from, this will give a proper amount of options to choose from. Therefore, the two different local planners that will be used in this research are: the Dynamic Window Approach and the Timed Elastic Bands.

### 2.2.2. Dynamic Window Approach

The Dynamic Window Approach is developed by Fox, Burgard, and Thrun [21], and is used in many robotic applications since [19]. This is the default local planner in the ROS navigation stack[20].

#### Working principle

The DWA searches for admissible velocities within a short time interval and eliminates velocities that result in collisions. This is done in two steps: 1) generate the search space, 2) optimize for the best trajectory.

**Search space** The two-dimensional search space consists of the angular and linear velocities, where only admissible velocities are considered, i.e. limited velocities and accelerations. Only velocities that are reachable in the time interval $\delta t$ are considered. The admissible velocities are directly derived out of the kinematics of the synchro-drive robot. The origin of the search space lies in the current linear and angular velocity of the robot. This two-dimensional space is discretized based on the discrete interval

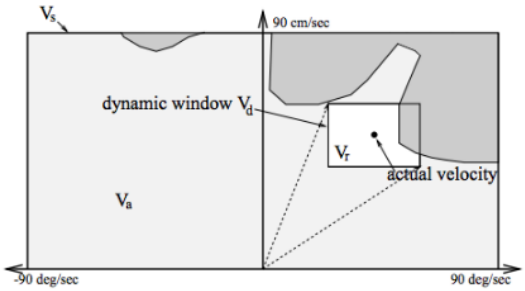



Figure 2.2: Example of a search space with a dynamic window plot [20]. In dark grey the blocked velocities due to obstacles, and in light grey the admissible velocities.

parameter, $\delta v$. Each velocity pair $(v, \omega)$ represents a circular arc, starting from the current position of the robot. The derived representation is called the *dynamic window* (See fig 2.2 for an example).

All trajectories outside this dynamic window are not reachable within $\delta t$, and are not considered for obstacle avoidance. The remaining trajectories are checked with the known positions of the obstacles and are considered safe if the robot is still able to stop before the collision with an obstacle.

The remaining safe trajectories are then used for optimization.

**Optimization** In this step, the best steering command is determined by optimizing an objective function. Included in this objective function are three terms: a measure of progress towards a goal location, the forward velocity of the robot, and the distance to the next obstacle on the trajectory.

The objective function trades off the desire of the robot to move fast towards the goal and its desire to go around obstacles:

$$G(v, \omega) = \sigma(\alpha \text{ heading } (v, \omega) + \beta \cdot \text{dist}(v, \omega) + \gamma \cdot \text{ velocity } (v, \omega)) \tag{2.1}$$

The three weighting parameters $(\alpha, \beta, \gamma)$ can be used to tune the behavior of the algorithm, from safe to fast[16].

The developers of the algorithm also introduced an extra parameter: speed-dependent side clearance. A safety margin around the robot is introduced to adapt the speed of the robot when it is close to obstacles. This margin grows linearly with increasing speed, to slow the robot down when driving through narrow corridors, and speeding up in open spaces.

### 2.2.3. Timed elastic Bands

The Timed Elastic Bands method, developed by Rösmann et al. [22], is an improved version of the original Elastic Bands method, by Quinlan and Khatib [23].

One of the biggest differences compared to the original Elastic Bands method is that dynamic and kinematic constraints are taken into account.

#### Working principle

This method converts the path to the goal, a sequence of waypoints which is computed by the global planner, into a smooth trajectory and avoids new obstacles using fresh sensor readings.

**Trajectory** The trajectory consists of $n$ geometric waypoints or vehicle poses (see fig **??**. The time interval between the configurations is denoted as $\tau = \{\Delta T_i\}_{i=0\ldots n-1}$.

The concept of this local planning method is to optimize both configurations and time intervals with a weighted multi-objective cost function.

**Objectives** The definition of the multi-objective cost function is:

$$f(B) = \sum_k \gamma_k f_k(B) \tag{2.2}$$

which is a weighted sum of objectives, with respect to the trajectory, and penalties for constraint violations, captured in $f_k$. Constraint violations consist of multiple components: geometric constraints, dynamic constraints, and non-holonomic constraints.

Geometric constraints bound the distance to obstacles and waypoints, waypoints are constraint from above by the maximal target radius $r_{p_{max}}$, and obstacles from below by minimal obstacle distance $r_{o_{min}}$. Dynamic constraints are used to take into account the maximum velocities and accelerations. The non-holonomic constraints are the kinematic characteristics of differential drive robots.

The objectives are about the performance of the trajectory of the TEB. The fastest path, as well as the shortest path, are evaluated.

Each objective and constraint in the cost function has a weight factor, $\gamma_k$. These parameters can be tuned to influence the performance and safety behavior.

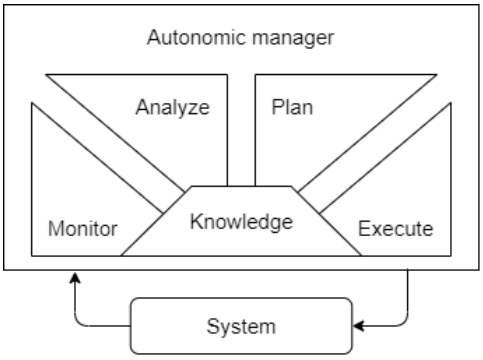



Figure 2.3: Overview of the MAPE-K loop.

## 2.3. Self-adaptation in robotics.

Now that the local planning algorithms that will be used in this research are analysed, the mechanism to manage the different possible system configurations is explained.

According to Arcaini, Riccobene, and Scandurra [24]: 'self-adaptation has been widely recognized as an effective approach to deal with the increasing complexity, uncertainty and dynamicity of these systems'. Self-adaptive systems typically have different configurations to choose from, which are defined at design-time [11]. Updated information which is gathered at run-time is then used to switch between configurations in order to maintain functionality or improve the execution performance.

### 2.3.1. What is self-adaptation?

A self-adaptive system is capable of changing its structure and/or behavior autonomously. The self-adaptive system can monitor the environment and the managed system itself, and can adapt its structure or behavior with the goal to maintain or improve the Quality of Service [25]. Adaptation can be done by changing parameters or artifacts of the system, when changes in the system itself or the environment occur [26].

Since the navigation system of a mobile robot can be configured with many different possible local planners and many different parameter sets, there are many different possible behaviors. Therefore, self-adaptation can be used to maintain or to improve the Quality of Service for the navigation system, especially when the robot is operating in changing environments.

### 2.3.2. MAPE-K feedback loop

A well recognized approach for the implementation of self-adaption is the MAPE-K approach [27] [24]. The MAPE-K approache adds a feedback loop to the managed system, which consists of the components: Monitor, Analyse, Plan, Execute, and centralized Knowledge component (see Figure 2.3).

**Knowledge** This component contains knowledge about the system itself, the environment, and adaptation goals. This knowledge is stored with the help of models, to provide an abstraction of relevant aspects of the system and environment [28].

**Monitor** The monitor component gathers information about the system and the environment, and updates the knowledge component with this new information. Data is gathered using sensors and probes for information about the environment and the system itself. This data is processed and then used to update the knowledge component.

**Analyze** The analyze component uses information from the monitor and the knowledge component to check the adaptation goals. These goals are compared with the state of the environment and the system itself, and the possible need for adaptation is analysed.

**Plan** If there is a need for adaptation, the plan component determines the actions which are required for this adaptation, to still achieve the system's goals.

**Execute** The last component does the actual execution of the adaptation, according to the plan that is determined in the plan component.

### 2.3.3. Strategies for self-adaptation

Different implementations of self-adaptation for robotic systems from literature are used to identify different possible strategies for self-adaptation. To understand the differences between these implementations, two classification schemes are used.

Krupitzer et al. [25] did an extensive literature survey on self-adaptive systems and came up with a classification for self-adaptive systems. Another classification for self-adaptive systems is developed by Rohr et al. [26], to structure and compare approaches from different domains. Parts from both classification schemes are used to explain the different approaches to build a self-adaptive system for mobile robots, and analyse the differences between the implementations. These differences are explained using the dimensions: time to adapt, reason to adapt, technique for adaptation, and reasoning about adaptation.

#### Time to adapt

The time when a system adapts can either be reactive or proactive according to Krupitzer et al. [25]. The traditional view for adaptation is reactive: the adaptation takes place after the performance drops under a certain level. In the proactive approach, the adaptation takes place before the drop of performance. The proactive approach is preferable according to Krupitzer et al. [25], since it prevents the actual drop of performance from happening. However, this is still an open challenge, since the predictive algorithms are complex to develop and are often very task specific which limits re-usability.

The time for adaptation in the method developed by Hernández, Bermejo-Alonso, and Sanz [14] is purely due to components failures and is thus pure reactive. Other methods [9][29][27] do not adapt only when the systems is not able to function anymore, but when the performance is under a predefined level, such that adaptation can improve with another control architecture configuration. These methods are still reactive, but the drop of performance to a non-functional level is prevented.

#### Reason to adapt

Rohr et al. [26] distinguished two reasons for adaptation: 1) changes in the environment, 2) changes in the system itself.

Both reasons to adapt are used in self-adaptive systems in robotics. Conventional fault tolerant control typically only adapts the system in case of internal failure [30]. The method developed by Hernández, Bermejo-Alonso, and Sanz [14] adapts only for changes in the system as well. On the other hand, there are methods that adapt to both changes in the environment and changes in the system itself [9][29][27].

#### Technique for adaptation

Different techniques for adaptation show up in literature. Krupitzer et al. [25] define three techniques for adaptation: 1) parameter adaptation, 2) structural adaptation, 3) context adaptation.

Since in mobile robot navigation the goal is to avoid collisions, making changes in the context is not desired. The environment is only monitored, and collisions are avoided without changing the environment. Therefore, this technique for adaptation is not further considered.

**Parameter adaptation** Parameter adaptation changes the behavior of the system by adjusting system parameters. This can be done relatively easy but can involve high complexity because different parameters can be dependent on each other [25]. Parameter adaptation is used by a lot of adaptive robotic systems [14][9][29][27]. However, the methods which do use parameter adaptation, use pre-determined parameter sets [27][14][9]. These parameter sets are determined at design time as possible algorithm variants. This requires to simulate different situations at design time, and reduces the amount of possible variants at run-time and therefore the flexibility.

According to Cheng et al. [31]: ”One of the main challenges that self-adaptation poses are that when designing a self-adaptive system, we cannot assume that all adaptations are known in advance.” There-

fore, pre-determination of parameter sets at design time cannot account for all scenarios the robot will encounter.

**Structural adaptation** In structural adaptation, algorithms or system components are exchanged dynamically at run-time to prevent performance loss, improve performance by adding new components or adjusting the system to new circumstances [25]. Structural adaptation will take place when big changes in behavior are needed, i.e. in the case of environmental changes.

Mobile robots can encounter different kinds of indoor environments. Narrow corridors can be found in stores and warehouses, as well as wide open spaces. Mobile robots operate traditionally in environments without humans, but nowadays it is desired to share the floor with humans, so the environment can contain only static or both static and dynamic obstacles. Mobile robots need to be able to handle many different environmental contexts.

Structural adaptation is used in all of the considered adaptation methods for robotics [27][14][9][29].

**Reasoning about adaptation**

A concept which is very useful for the purpose of reasoning about adaptation, is the concept of non-functional requirements (NFRs). Traditionally, fulfilling the functional requirements is the primary focus for robotic systems, but balancing the non-functional requirements is becoming more and more important [12].

Functional properties define *what* a system does, while in contrast non-functional properties define *how* a system performs [13]. Relevant examples of non-functional properties for mobile robot navigation are safety, performance, computational costs and energy consumption.

Non-functional requirements can be conflicting; higher velocity, and thus higher performance, will probably result in lower level of safety. The individual importance of non-functional requirements can vary in different contexts. For example, when the battery of the robot is low, power consumption can be more important than performance.

Functional requirements are used to determine the necessary components for the robot to be able to function, i.e. to accomplish the high level goals. The non-functional requirements, however, can be used to optimize the performance of execution. Lotz et al. [32] and Brugali et al. [27] both developed a way to model the non-functional behavior and they both use it to improve the overall execution performance.

## 2.4. Summary

The navigation problem of a mobile robot is divided into multiple sub-problems, of which path planning is one of the most important issue [7]. Path planning consists of two parts: global path planning and local path planning. Local path planning has received more attention from researchers in recent times due to the higher demand for mobile robots in dynamic and unknown environments [17]. Since this research is about navigation in changing and therefore dynamic environments, local path planning is the focus for this research.

There are a lot of different popular local planning algorithms and many algorithms are developed for a specific scenario or environment. Two local planning algorithms will be used in this research: the Dynamic window Approach and the Timed Elastic Bands method. Both the algorithms have multiple parameters that can be tuned to optimize the behaviour for specific environment.

Self adaptation can be used to manage the different possible configurations. Self-adaptive approaches differ in multiple dimensions: time to adapt, reason to adapt, and technique for adaptation. The MROS framework is a reactive framework, the time to adapt is after a drop in performance is happened. The reasons for adaptation for the MROS framework are both changes in the environment and in the system itself. Technique for adaptation is only structural adaptation. Nonfunctional requirements can be used to reason about adaptation.

# 3

# Research approach

This chapter explains the approach of this research; research sub-questions are defined and the approach to answer the research questions is explained.

The main research question for this thesis is defined as:
*1) How can navigation quality be improved for mobile ground robots based on changes in the environments using a self-adaptive framework?*

The navigation problem of a mobile robot is divided into multiple sub-problems, of which path planning is one of the most important issues [7]. Path planning consists of two parts: global path planning and local path planning. Local path planning has received more attention from researchers in recent times due to the higher demand for mobile robots in dynamic and unknown environments [17]. Since retail environments can be dynamic and (partially) unknown, local path planning will be the focus of this research. There are a lot of different popular local planning algorithms and they are often developed for a specific scenario or environment. This means that there are a lot of different possible system configurations when a robot operates in multiple different environments.

When, at run-time, the robot encounters a new environment, a choice has to be made about which configuration to use. This choice is traditionally made by the engineer at design time. But since there is a demand for more autonomy in robotic systems, it is desired to have this choice made by the robotic system itself. This higher level of autonomy can be reached using a self-adaptive framework. The MROS framework will be used in this research to obtain the self-adaptive capabilities [14].

To answer the research question in more detail, four sub-questions are used. These sub-questions together with the approach to answer them is discussed in the following sections.

## 3.1. Navigation quality definition

*1.1) How to define navigation quality for mobile robots in retail environments?*
The goal for mobile robots in retail environments is to improve productivity, and therefore the performance is an important element for the navigation quality. The clear separation of workspace between robots and humans in industry is used to simplify the problem and is not optimal. However, when this clear separation is removed, safety becomes more challenging. Although dynamic obstacles are not used in this research, the focus is to improve the safety to prepare for these scenarios. The two elements of navigation quality that will be the focus of this research are safety and performance.

There is often a trade-off between performance and safety; the robot can reach the goal faster when it drives closer to obstacles to shorten its path, but closer to obstacles means that the trajectory is less safe. Brugali [11] states: ”Designing robot software architectures that satisfy multiple NFRs is difficult because NFRs are typically conflicting, such as Safety and Performance.” The trade-off between these two non-functional properties will be challenging to solve. It is therefore chosen to focus on safety first,

since this is the open challenge for robots environments that are possibly shared with humans. Only after the safety requirement is met, performance will be considered.

## 3.2. Measuring environment characteristics

*1.2) Which environment characteristics are relevant for the navigation quality?*
To be able to switch configurations autonomously based on the changing environment, the robot needs to know what the current environment is. Relevant environment characteristics needs to be measured using information from the robot's sensor data. Different concepts for environment metrics are developed to measure these characteristics. Validation of the new metrics is done with the help of statistical tools to validate that there indeed is a relation between the quality levels and changes in the environment characteristics.

## 3.3. Prediction using quality models

*1.3) How can the navigation quality of the system configurations be predicted, using the characteristics from the environment?*
To be able to switch configurations autonomously based on changes in the environment, the robot needs to have knowledge about the navigation quality of the system configurations in different environments. If the measured quality levels of the current configurations do not meet the pre-specified requirements, a different configuration need to be searched for. To be able to do this search with more information, the quality levels of the other configurations can be predicted based on the current environment. Quality models are used to do this prediction. The input of the quality models are the environment measures, and the output is the predicted quality level for a configuration.

## 3.4. Self-adaptive framework implementation

*1.4) How can the predicted quality levels be used in the self-adaptive framework to improve the navigation quality based on changes in the environment?*
The MROS framework, first developed by Hernández, Bermejo-Alonso, and Sanz [14], will be used as a tool to obtain the self-adaptive capabilities. This is a reactive framework, i.e. adaptation takes place after a drop in a quality level is detected. The reason for the adaptation can be because of functional failures, or non-functional requirement violations. The adaptation technique used in this framework is structural adaptation.

In the current concept of the MROS framework, the reasoner can use (functional) knowledge about the configurations together with the perceived quality levels of the current configuration. The contribution of this research will be an additional source of information. Based on the characteristics of the environment, the QA levels of the other configurations will be predicted. The reasoner can use this additional source of information to make a better informed decision about adaptation.

# 4

# Quality and Environment metric development

In the previous chapter, the approach of this research is explained. The goal of this research is to improve navigation quality in changing environments using a self-adaptive system. Navigation quality is defined in terms of safety and performance and these quality attributes needs to be measured at run-time. To improve the knowledge of the robot about the navigation quality in different environments, the relevant environment characteristics need to be measured as well. The aim of this chapter is to show how the metrics are developed.

This chapter answers the first two research sub questions: *How to define navigation quality for mobile robots in retail environments?* And *Which environment characteristics are relevant for the navigation quality?*

First, relevant literature about quality attribute metrics is analysed. Next, different concepts for the navigation quality and environment metrics are discussed. After that, the different concepts are tested in simulation, and validated using statistical analysis tools.

## 4.1. Related work on quality metrics

In order to evaluate the quality attributes at run-time, the system needs to be able to measure these attributes. Different metrics for safety and performance for robotic systems show up in literature.

Muñoz, Valencia, and Londoño [33] evaluate the following navigation quality characteristics: mean distance between robot and obstacles, distance covered by the robot, time to completion, and smoothness of the trajectory. Distance to obstacles is an interesting measure for safety. For performance, the path distance and time to completion can be used.

According to Brugali [11]: ”The stopping distance when an obstacle is detected along the robot path can be used to specify safety constraints.” Performance is about computing the shortest path to the destination in order to reduce service time.

Steinfeld et al. [34] define potential performance measures for local navigation and obstacle encounter. Relevant examples of their measures are: percentage of navigation tasks successfully completed, deviation from the planned route, obstacles that were successfully avoided.

Brugali et al. [27] define safe navigation in terms of the ability to detect obstacles and the ability to stop before an obstacle or get around. They state that: ”for safe navigation, the minimum obstacle distance should be greater than the sum of the blind distance, the action distance, and the braking distance.”

Chung et al. [35] state that performance metrics can be defined in terms of minimum energy, minimum distance, or minimum time. And they define risk of collision using the distance to collision. $d_{collisions}$ is

calculated by summing three terms: delay distance, braking distance, and obstacle movement distance.

Corbato et al. [36] measure performance by the time to completion. They use the risk of collision, from Chung et al. [35], to define a safety metric using the braking distance as a safety margin.

Vicente-Chicote et al. [37] state that there are just a few component models that support the specification and management of non-functional properties. To solve this issue, they developed RoQME: RoQME is a model-drive tool-chain that provides tools to model relevant system-level non-functional properties. This tool can be very useful for this research, but unfortunately this tool is developed for the RobMoSys framework and is not yet compatible with the ROS framework (see appendix B for an experience report).

## 4.2. Quality metric development

Metrics from literature are used as inspiration for the development of the concepts of the quality attribute metrics. The metric concepts are first discussed and after that, experiments are done to test and validate the new metrics.

### 4.2.1. Safety metric development

Most of the safety metrics that are mentioned in section 4.1 have in common that the distance to obstacles is used. Comparing the distance to obstacles to the braking distance can indicate the risk of collision. This risk of collision can be used to determine the safety level.

The safety metric for this research combines the concepts found in literature. This metric uses a safety margin around the robot based on the braking distance. Obstacles within this safety margin are considered to be a risk for collision, and will therefore decrease the perceived safety level.

#### Safety margin

The size of the safety margin is determined using the braking distance, which is dependent on the current forward velocity of the robot and the maximum deceleration.

$$d_{brake} = \frac{v_x^2}{2a_{max}} \tag{4.1}$$

where $a_{max}$ is a constant, defined by the robot's limits. For this research, the maximum deceleration is defined to be $0.5m/s^2$.

For non-holonomic robots, the braking distance is only relevant for the forward motion. The safety margin can therefore be limited to the front side of the robot. However, in this research obstacles at the side of the robot are considered important for safety as well. The braking distance is used to define a safety margin not only in front but around the robot.

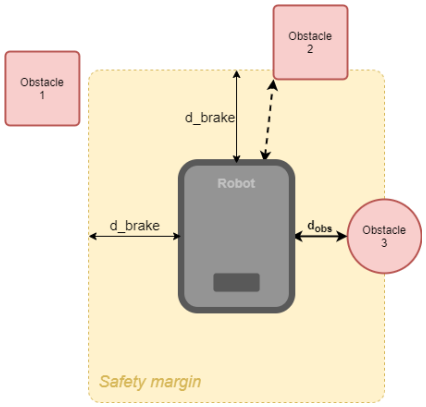
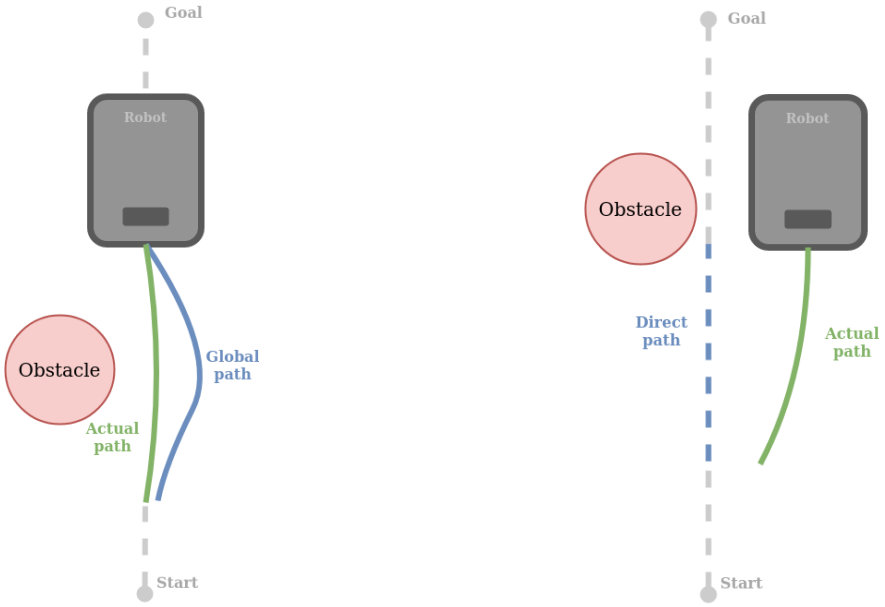



Figure 4.1: Visualization of the safety margin around the robot.




(a) Global path reference.

(b) Direct path reference.

Figure 4.2: Visualizations of the options for the reference paths which for the Performance3 metric.

The width and length of the robot are taken into account, resulting into a rectangular safety margin for the Boxer robot (see Figure 4.1). Since the braking distance differs for different velocities, the size of the safety margin is velocity dependent.

**Safety metric definition**

When there is no obstacle inside the safety margin area, the safety level is equal to 1. The safety level is 0 when the distance to an obstacle is zero. For obstacles inside the safety margin, the safety level scales linearly based on the obstacle distance. (see equation 4.2).

$$\text{Safety} = \begin{cases} 1 & \text{if } d_{obs} \geq d_{brake} \\ \frac{d_{obs}}{d_{brake}} & \text{if } d_{obs} < d_{brake} \end{cases} \tag{4.2}$$

where $d_{obs}$ is the distance to the nearest obstacle. Only the closest obstacle is used to determine the obstacle distance (see Figure 4.1.

### 4.2.2. Performance metric development

Multiple concepts for performance metrics are set up, based on performance metrics from literature. Most performance metrics from literature are either the time to completion or the distance travelled, and these will be used as a starting point for the development of the performance metric concepts of this research.

**P1, Time to completion**

The time to completion, $t_c$, is the total time it takes for the robot to move from its initial position to the goal position.

$$P_1 = t_{end} - t_{start} \tag{4.3}$$

This is a well known and often used performance metric in mobile robot navigation. Unfortunately, this information is only available after the goal is reached. It is however, desired to have a run-time metric: i.e. a metric that can be measured at run-time. Therefore, other concepts for performance metrics will be discussed in the following sections.

**P2, Speed ratio**

The second performance metric concept is defined as the ratio between current forward velocity, and the maximum velocity.

$$P_2 = \frac{v_x}{v_{max}} \tag{4.4}$$

where $v_x$ is the current linear velocity, and $v_{max}$ is a constant, determined by the limits of the robot.

In contrast to P1, which is not a run-time metric, the P2 metric can be measured at run-time. This metric is about the ratio of the current velocity of the robot, compared to the maximum allowed velocity. Local planners are often lowering the velocity when the robot is close to obstacles. This is done to improve safety and reliability, but this increases the time to completion and therefore the performance level is reduced.

However, after initial tests it is concluded that P2 is not a valid performance metric. When a configuration will stay further away from obstacles, the travelled distance will increase. This will also mean that the time to completion will increase, and therefore the average performance should be lower. But when the longer path is driven with the maximum speed, the average Performance2 metric will result in an average of 1.0, while the time to completion is increased. This metric will not be used as a run-time metric in the rest of this research.

**P3, Time to completion ratio**

This metric is inspired by the time to completion metric and the deviation from the global path from [34]. This metric uses the time to completion of the past 2 seconds of the actual path, and compares it to the time to completion of a reference path. For the reference path, two options are considered which will be explained later in this section. By using information from the past two seconds and not the complete run, the performance level can be evaluated at run-time (after the first two seconds).

The length of the reference path is determined, and divided by the maximum allowed velocity to obtain the time to completion of the reference path.

$$t_{ref} = d_{ref}/v_{max} \tag{4.5}$$

The performance metric is defined as the ratio between the time to completion of the actual path and the reference path:

$$P_3 = \frac{t_{ref}}{t_{act}} \tag{4.6}$$

where $t_{act}$ is constant at $t_{act} = 2s$.

There are two options for the reference path. For the first option, the actual path is projected on the global path to get the part of the global path of the past two seconds (See Figure 4.2a).

The second option uses the direct path as reference. The direct path is defined as the shortest path to the goal position, without taking into consideration new or dynamic obstacles. For a simple corridor, the direct path is equal to the center-line of the corridor(see fig 4.2b). In the final experiments of this research, simulations will be done in a larger supermarket world. The direct path in this world can be determined by planning a global path in the empty store.

The difference between the time to completion of the global path and the actual path (local path) is a good indication for the performance of the local planner alone. The comparison with the direct path will indicate the performance of the complete navigation system. The global path will be used as reference path for this metric, since in this research the focus is on the performance of the local planner.

## 4.3. Environment characteristics metric development

To be able to improve the knowledge of the robot about its navigation quality in changing environments, the characteristics of the environment need to be measured. In this section, two metrics are developed to measure relevant environment characteristics.

### 4.3.1. Narrowness

Kulich, Kozák, and Přeučil [16] inspired this concept for Narrowness; they state that the DWA has good results when the environment is wider than twice the diameter of the robot. This indicates that the width of the environment is relevant for the navigation quality of the local planner.

The Narrowness metric is a metric that indicates how many robot widths fit in the width of the current environment. The environment width is measured from the center of the robot, in the direction perpendicular to the direction of travel. The free space distance, which is the distance to the closest obstacle, on both sides of the robot are determined using the lidar data (see Figure 4.3). This metric is divided by the width of the robot to obtain the final metric.

$$N = \frac{d_{obs\_left} + d_{obs\_right}}{r_{width}} \tag{4.7}$$

Where $r_{width}$ is the width of the robot.

4 different variations of this metric are tested, with different maximum ranges per side: 1m, 2m, 3m, 4m.

### 4.3.2. Obstacle density

Information about the amount of obstacles around the robot is used in the research of Corbato et al. [36] as an experiment parameter. The information about the obstacle density can be a relevant environment characteristic as well. The Obstacle Density (OD) metric is defined as the ratio between the total area and the occupied area in a local window around the robot. The information about the occupied area is taken from the local costmap. Obstacle Density is equal to 0 when there are no obstacles, and Obstacle Density is equal to 1 when there is no free space.

$$OD = \frac{\text{Occupied area}}{\text{Total area}} \tag{4.8}$$

The default size for the local costmap in ROS is 5m x 5m. Since only obstacles close to the robot are relevant, a size equal to or smaller than the default local costmap is considered. Different sizes for the window are tested: 4mx4m, 3mx3m, 2mx2m, 1mx1m.

It is possible that obstacles which the robot will encounter in the future are more relevant for the quality attributes than obstacles the robot encountered in the past. Because of this, two types of windows are tested: type 1 is centered around the robot, and the center of type 2 is shifted to the front of the robot such that the rear side of the window intersects with the center of the robot (see Figure 4.4).

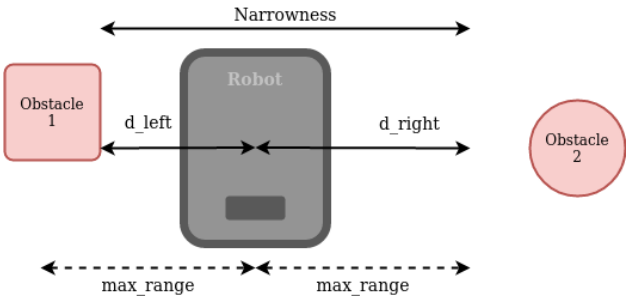



Figure 4.3: Visual explanation of the narrowness metric

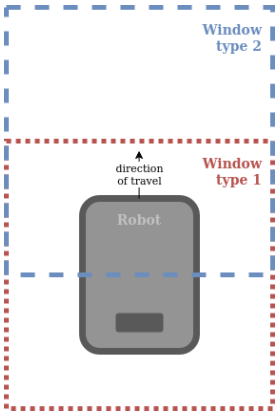



Figure 4.4: Visual explanation of the Obstacle Density window types.

## 4.4. Tests and validation

To test and validate the metrics, simulation experiments are designed. The data collected from these experiments is used to test and validate the metrics and to be able to make a choice about which metrics to use.

### 4.4.1. Experiment design

The design of the experiment and the environment parameters are explained in more detail in appendix C.

From the AHXL Delft floorplan, the dimensions of the corridors are measured, and the smallest and largest possible passages (See Table 4.1). These dimensions are used for determining the values for the environment parameters, to make sure that the experiment is done in a realistic environment.

Based on the dimensions from the AHXL floorplan and some initial tests, the values for the environment parameters are chosen as shown in table C.2.

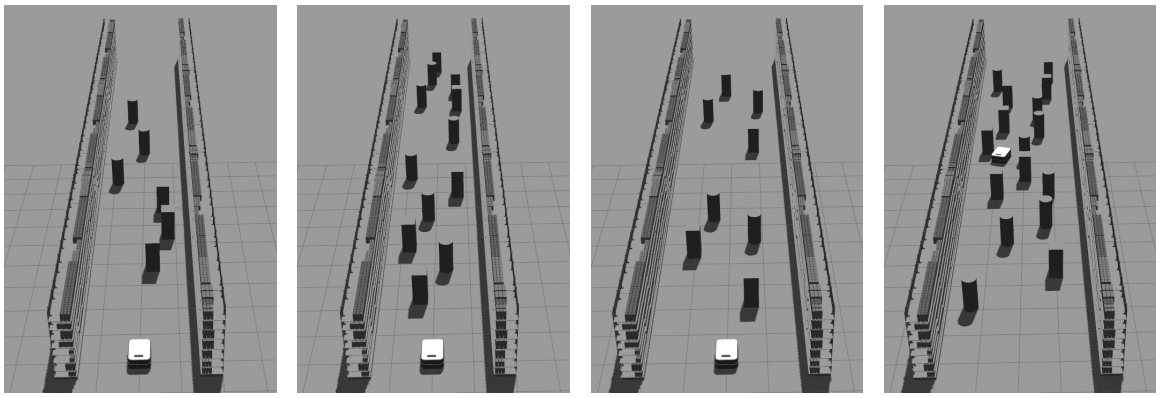
See Figure 4.5 for examples of these different environments.

Table 4.1: Dimensions from the AHXL floorplan

| Type | Width [mm] |
|---|---|
| Normal corridor | 2300 |
| Wide corridor | 4200 |
| Narrow corridor | 1800 |
| Smallest passage | 1500 |
| Widest passage | 5000 |

Table 4.2: Environment design parameters for the metrics validation experiments

| Environment id | Env1 | Env2 | Env3 | Env4 |
|---|---|---|---|---|
| Corridor width | 3 | 3 | 4 | 4 |
| Clutterness | 0.10 | 0.25 | 0.10 | 0.25 |
| Obstacle minimum distance | 1.4m | 1.4m | 1.4m | 1.4m |



(a) Environment 1 (b) Environment 2 (c) Environment 3 (d) Environment 4

Figure 4.5: Examples of different simple corridor environments.

### 4.4.2. System configurations

The default configurations are discussed in appendix D. In these experiments, four configurations are used: the two local planners with default parameters, and two variations with different maximum speeds (v).

- dwa_v0
- dwa_v1
- teb_v0
- teb_v1

### 4.4.3. Performance metrics validation

The first performance metric, the time to completion, is a metric that is often used to measure performance. The measurement of this metric is only available after the the goal is reached, and is therefore not useful for run-time adaptation based on performance. This metric will not be used for run-time measurements, but will be used to test the other metric concept is a valid performance metric.

The average performance score of the Performance3 metric is compared to the time to completion(P1) of the complete run. Performance3 is a valid performance metric, if the ranking is consistent with the ranking of time to completion. Meaning: the lowest time to completion must have the highest average performance score, and the same holds for the second, third, and fourth place.

This comparison is done for a run in all four environments, and the results for are shown in table 4.3. I can be seen that the ranking of the Performance3 metric is equal to the time to completion ranking. The Performance3 metric will be used as run-time metric in the rest of this research.

Table 4.3: Comparison of rankings per run of the performance metrics

| Environment | Metric | 1st | 2nd | 3th | 4th |
|---|---|---|---|---|---|
| Env1 | Time to completion | dwa_v0 (25.6 s) | teb_v0 (28.9 s) | dwa_v1 (31.6 s) | teb_v1 (32.9 s) |
| | Performance3 | dwa_v0 (0.99) | teb_v0 (0.9) | dwa_v1 (0.82) | teb_v1 (0.79) |
| Env2 | Time to completion | dwa_v0 (25.1 s) | teb_v0 (25.8 s) | dwa_v1 (31.3 s) | teb_v1 (32.2 s) |
| | Performance3 | dwa_v0 (1.0) | teb_v0 (0.98) | dwa_v1 (0.8) | teb_v1 (0.79) |
| Env3 | Time to completion | dwa_v0 (25.6 s) | teb_v0 (26.2 s) | dwa_v1 (31.5 s) | teb_v1 (32.7 s) |
| | Performance3 | dwa_v0 (0.98) | teb_v0 (0.98) | dwa_v1 (0.81) | teb_v1 (0.78) |
| Env4 | Time to completion | dwa_v0 (25.2 s) | teb_v0 (25.9 s) | dwa_v1 (31.4 s) | teb_v1 (32.3 s) |
| | Performance3 | dwa_v0 (0.99) | teb_v0 (0.98) | dwa_v1 (0.8) | teb_v1 (0.79) |

### 4.4.4. Environment metric validation

To be able to build a model of the navigation quality of the configurations of the robot in different environments, there must be a relation between the environment and the quality metrics. To validate this relation between the metrics, the cross-correlation function is used.

**Cross-correlation**

WestWick and Kearney [38]: ”Correlation functions describe the sequential structures of signals. In systems analysis, they are used to analyze relationships between signals. The values of the cross-correlation coefficient function can be interpreted as correlations in the statistical sense, ranging from complete positive correlation $(1)$ through 0 to complete negative correlation $(-1)$.The cross-correlation function describes the relation of signal $x(t)$ with signal $y(t)$, shifted by $\tau$ time units. The change in correlation as a function of lag characterizes the sequential structure of the signal.” A correlation close to 0 means that there is no relation, and a correlation close to 1 or -1 means that there is a relation (negative means inverted relation).

Cross-correlation function:

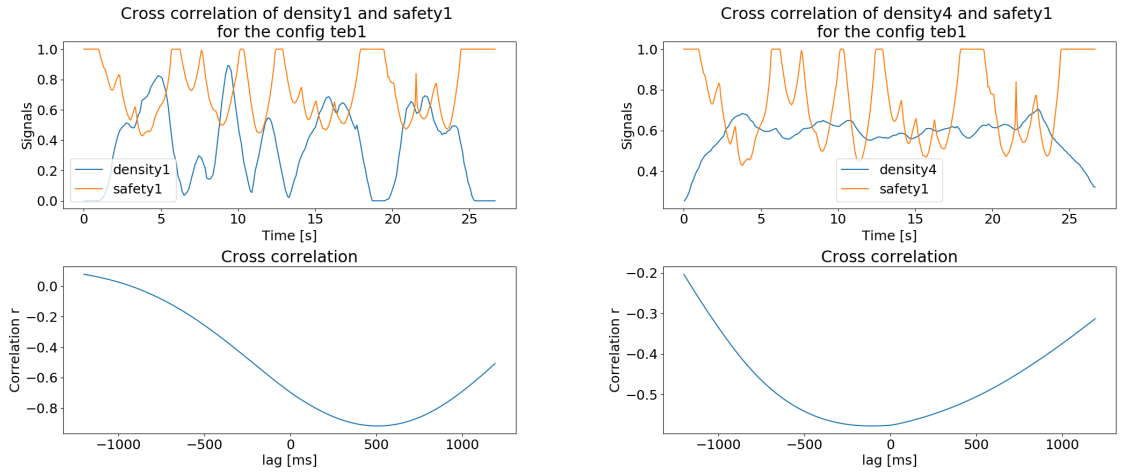
$$\phi_{xy}(\tau) = E[x(t-\tau)y(t)] \tag{4.9}$$

The cross-correlation function is used to validate that there is a relation between the environment metrics and the quality attributes. If there is a strong relation, the maximum correlation will be close to 1 or -1. The lag value at this maximum tells something about the lag of the response in the relation. For example: it is possible that if the Obstacle Density gets higher, the response of safety level will be half a second later. In this case the maximum correlation will be at a lag of 500ms.

Multiple variations of the environment metrics are tested, and the metric with the highest maximum correlation will be selected. Not only the value of the maximum correlation is important, the amount of lag where this maximum is at is also important. The environment metrics will be used at run-time to reason about adaptation. If the maximum correlation is at a lag of 500ms, this means that at run-time, the environment measure of half a second earlier needs to be used. Using a metric with a positive lag is possible, but negative lag values cannot be used, since the measure in the future is not known yet. To minimize the amount of data pre-processing steps, it is desired to have a metric with a lag close to zero.

Therefore, the metric with the highest maximum (absolute) correlation and with a lag close to 0 is the preferred choice.

**Environment metric correlation**




(a) Cross correlation of Obstacle Density1 and Safety

(b) Cross-correlation of Obstacle Density4 and Safety

Figure 4.6: Figures with the two signal that are compared, and their respective cross-correlation.

The maximum correlation coefficient for the relations are determined for each run and each configuration, to see which of the environment metrics are more useful i.e. have a higher (absolute) maximum correlation. The results are shown in table 4.4. The maximum correlation is shown per configuration, and for the average of all configurations. This is done to make sure that the correlations are high for all configurations.

**Correlation with Safety**

It can be seen in Figure 4.6a that when the Obstacle Density gets higher, the Safety will be lower.

The cross correlation for Obstacle Density1 with Safety shows a maximum (absolute) value of -0.838 at a lag of 580ms (see table 4.4). A value of -0.838 is very high, meaning that there indeed is a (strong) relation between the quality attribute Safety and the environment metric Obstacle Density1. For Narrowness, correlations up to 0.699 are reached, which also means that there is a relation between Safety and Narrowness.

When the results of the correlations of the different Obstacle Density metrics with Safety are compared, it can be seen that the correlation of the metrics with the small windows(OD1.1 and OD2.1) is higher

Table 4.4: The average maximum correlation with the safety metric, per configuration and the mean over the configurations.

| | Average max correlation with Safety | | | | |
|---|---|---|---|---|---|
| Metric | dwa_v0 | dwa_v1 | teb_v0 | teb_v1 | mean |
| N1 | 0.628 | 0.591 | 0.743 | 0.709 | 0.668 |
| N2 | 0.595 | 0.599 | 0.667 | 0.641 | 0.625 |
| N3 | 0.475 | 0.537 | 0.541 | 0.546 | 0.525 |
| N4 | 0.473 | 0.526 | 0.52 | 0.544 | 0.516 |
| OD1.1 | -0.798 | -0.725 | -0.875 | -0.884 | -0.821 |
| OD1.2 | -0.788 | -0.696 | -0.751 | -0.717 | -0.738 |
| OD1.3 | -0.726 | -0.653 | -0.675 | -0.601 | -0.664 |
| OD1.4 | -0.185 | -0.479 | -0.131 | -0.115 | -0.227 |
| OD2.1 | -0.799 | -0.71 | -0.862 | -0.877 | -0.812 |
| OD2.2 | -0.794 | -0.704 | -0.739 | -0.705 | -0.736 |

Table 4.5: Lag at the average maximum correlation with the safety metric, per configuration and the mean over the configurations.

| | Lag (ms) at average maximum correlation | | | | |
|---|---|---|---|---|---|
| Metric | dwa_v0 | dwa_v1 | teb_v0 | teb_v1 | mean |
| N1 | 0 | 25 | 0 | 0 | 6.25 |
| N2 | 50 | 25 | 0 | 25 | 25 |
| N3 | 75 | 25 | 0 | 0 | 25 |
| N4 | 75 | 50 | 0 | 0 | 31.25 |
| OD1.1 | 600 | 600 | 700 | 700 | 650 |
| OD1.2 | 600 | 525 | 400 | 1150 | 668.75 |
| OD1.3 | 575 | 700 | 500 | 725 | 625 |
| OD1.4 | 425 | 775 | 400 | 275 | 468.75 |
| OD2.1 | 100 | 75 | 50 | 25 | 62.5 |
| OD2.2 | -200 | -250 | -725 | -75 | -312.5 |

Table 4.6: The average maximum correlation with the performance metric, per configuration and for all combined.

| | Average maximum correlation with Performance3 | | | | |
|---|---|---|---|---|---|
| Metric | dwa_v0 | dwa_v1 | teb_v0 | teb_v1 | mean |
| N1 | -0.105 | 0.314 | -0.2 | -0.126 | -0.029 |
| N2 | -0.223 | 0.234 | -0.259 | -0.12 | -0.092 |
| N3 | -0.074 | -0.061 | -0.231 | -0.104 | -0.117 |
| N4 | -0.074 | 0.055 | -0.234 | -0.104 | -0.089 |
| OD1.1 | 0.105 | -0.331 | 0.192 | 0.17 | 0.034 |
| OD1.2 | 0.2 | -0.192 | 0.24 | 0.119 | 0.092 |
| OD1.3 | 0.333 | -0.238 | 0.317 | 0.139 | 0.138 |
| OD1.4 | 0.092 | 0.115 | 0.176 | -0.052 | 0.083 |
| OD2.1 | 0.245 | -0.332 | 0.195 | 0.133 | 0.06 |
| OD2.2 | 0.232 | 0.061 | 0.226 | 0.124 | 0.161 |

than the others. This means that an Obstacle Density using a window of 1x1m is more useful than the other metrics with a larger window.

The same can be concluded for the Narrowness(See table 4.4 and 4.6). Narrowness1 (max range of 1m) has the highest correlation with Safety, and is therefore the best Narrowness metric to use in combination with Safety.

When the lag values are compared (table 4.5), it can be seen that OD2.1 has a mean lag of 30ms, while the mean lag of OD1.1 is 530ms. Since a lag close to 0 is preferred, OD2.1 is selected as preferred Obstacle Density metric in relation to the Safety metric.

#### Correlation with Performance

It can be seen in table 4.6 that the correlations for performance with the environment metrics is low. This means that there is not a (strong) relationship between the performance level and the environment metrics. It is therefore chosen to only use the environment metrics in combination with Safety.

## 4.5. Discussion

In this chapter, metrics for the quality attributes and for the environment characteristics are developed and validated. Safety will be measured by using a safety margin around the robot, with a size that is dependent on the braking distance. The safety level scales linearly with the distance to the closest obstacle within this safety margin.

For performance, there are multiple concepts developed. The first performance concept is the time to completion, a metric that is often used to measure performance for mobile robot navigation. Since this is not a run-time metric, other performance concepts are developed as well. The second performance concept is defined as the ratio between the current speed and the maximum speed. The third and last performance concept is a ratio between the time to completion of the actual path and the global path of the past second. Experiments are done to test if the newly developed metrics are valid performance metrics.

To measure relevant environment characteristics, two metrics are developed. Obstacle Density is the ratio between occupied space and total space in a local window close to the robot. Narrowness is defined as the amount of robot widths that fit in the current environment of the robot, measured in the direction perpendicular to the direction of travel. Experiments are done to test the relevance of these metrics. And the results of these experiments are used to test different window sizes and types for the Obstacle Density metric, and different maximum ranges for the Narrowness metric.

### Results

The average of the run-time performance metrics are compared to the time to completion to validate the run-time metrics. The validation results showed that the Performance3 metric is a valid run-time metric. This metric will be used in the remainder of this research to measure the performance level.

For both environment metrics, different variations are tested and validated. These experiments showed that the Obstacle Density metric with the smallest window sizes has the highest correlation with Safety. The window type that is shifted to the front resulted in a lag close to zero. Therefore, Obstacle Density2.1 is selected as the first environment metric. For Narrowness, the smallest maximum range resulted in the highest correlation with Safety, and is therefore selected as the second environment metric.

The environment metrics do not have a strong relationship with the Performance metric. Since Performance is not the main focus for adaptation, it is chosen to not use the Performance metric to build quality models.

### 4.5.1. Limitations and future work

In this section, improvements for the developed metrics are proposed. The metrics can be developed and optimized further using these ideas, but this is out of the scope of this research.

#### Safety metric

The safety margin is an area with a constant distance around the robot. However, the braking distance is more relevant for the area in front of the robot than the area on the side, since the robot cannot move sideways (for non-holonomic robots). An obstacle 0.5m in front of the robot should therefore lower the safety level more than an obstacle that is at 0.5m at the side of the robot. This can be solved by using two different safety margins, one for obstacles in front and one for obstacles at the side of the robot,

or by using a non constant safety margin size; meaning that the safety margin is smaller at the side of the robot than in front.

For the safety metric, the safety level scales linearly with the obstacle distance for obstacles within the safety margin ($Safety = \frac{d_{obs}}{d_{brake}}$). It can be argued, however, that this linear scale is not realistic. The perceived risk level is increasing more rapidly when obstacles become closer. Which can be approximated by a logarithmic scale for example.

**Performance metric**

It is chosen for the Performance3 metric to use the global path as the reference path. The focus of this research is on the performance of the local planner, which can be measure by the difference between the global and local plan. However, if in future research the complete navigation system will be considered, the direct path will be a better choice for the reference path. The direct path is the optimal path from start to goal, and is therefore a perfect reference for the performance level of the complete system.

**Environment metrics**

For the Obstacle Density window size, and the Narrowness maximum range, fixed integer values are used. The value is selected based on the correlations with the Quality attributes. It is perfectly possible that a non-integer value has even better results. Or a window size or maximum range equal to the size of the safety margin. A different robot size can also influence the choice for the OD window size or Narrowness maximum range.

To obtain the Obstacle Density metric, information from the local costmap is used. This information is however not necessarily complete. This is what the robot knows while in reality the obstacle density can be higher than the robot knows since the robot can only see the front of an obstacle. Therefore this obstacle density is the perceived obstacle density, and not the ground truth. The same holds for the other metrics, they are perceived values and not the ground truth.

# 5

# Quality model construction

The knowledge of the system about the navigation quality in different environments will be improved by the implementation of quality models. These quality models can be used to predict the values of quality attributes for the configurations based on the current environment. The environment metrics, which were developed in the previous chapter, will be used as inputs for the quality model. The output of the quality model will be the predicted safety level for a configuration.

In this chapter, the third research sub-question is treated: *How can the navigation quality of other system configurations be predicted, using the characteristics from the environment?* This chapter will explain how the quality models are constructed. This is done by first analysing the general machine learning theory. Then, 7 steps are followed to construct the models. This involves, among other things, the collection of data, choosing of the model type, training and validating the models.

## 5.1. Machine Learning theory

A lot of research is done in machine learning, and therefore a lot of different machine learning methods exist. The type of method that can be used is different per problem.

Different types of learning can be categorized into [39]: supervised, unsupervised, and reinforcement learning. Supervised learning uses a data-set which is labelled, the algorithm will use the input features and the labels to extract a model of the relation between the two. Unsupervised learning uses unlabelled data, the algorithm will try to make sense of the features and patterns on its own. Reinforcement learning uses a reward function to train agents to improve performance on a goal. The performance and safety levels are measured at each time-step, and therefore each data point is labeled. Supervised machine learning methods are therefore the most relevant for this research.

Supervised machine learning tasks can be divided into two categories: regression and classification [39]. The former predicts continuous values while the latter predicts discrete values or classes. Since all the signals (quality attributes and environment metrics) are continuous, regression is the relevant category for the quality models.

Since there are a lot of different supervised regression methods, this research only considers the methods from a popular python library for machine learning: Scikit-learn. Scikit-learn is a Python module for supervised and unsupervised state-of-the-art machine learning problems [40].

The different machine learning methods are developed for different problems. The problem types are defined by (among other things): the type of the system, number of features, number of outputs, or data-set size.

**Features** These are the input variables for the model: Narrowness and Obstacle Density.

**Number of outputs** There is only one output variable for the model, the predicted safety level.

**Data-set size** Since data is generated in simulation, data-set size is not limited like it is in many other machine learning problems. A single run is approximately 30 seconds. With a time-step of 0.1 this will results in 300 data points for a single run. For the training of a single model, 8 different runs will be used, resulting in 2k+ data points. Next to that, the data-set size can easily be extended if needed.

The machine learning process involves seven steps, from data collection to actually predicting [41] (see Figure 5.1): Data collection, data preparation, choosing a model, training, evaluation, parameter tuning, and prediction. These steps will be followed to construct the quality models.

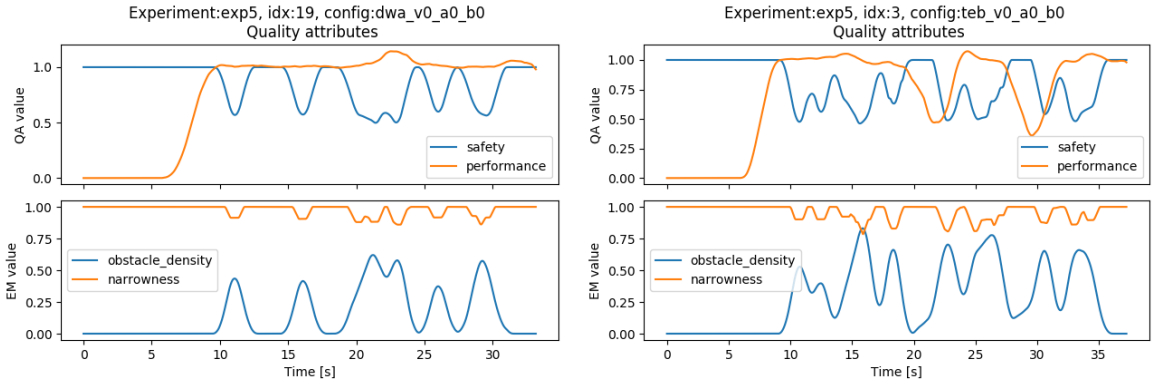

Figure 5.1: Seven steps for machine learning, from [41].

## 5.2. Data collection and preparation

Different environments are created by using different corridor widths and clutterness values, see table 5.1. This in done according to the experiment design of appendix C. Each environment is generated five times with randomly spawned obstacles to be able to collect enough data. See Figure 5.2a and 5.2b for an example of the measured signals for one run.

Table 5.1: Environment design parameters for model training tests.

| Environment id | Env1 | Env2 | Env3 | Env4 |
|---|---|---|---|---|
| Corridor width | 3 | 3 | 4 | 4 |
| Clutterness | 4 | 8 | 4 | 8 |




(a) For the config: dwa_v0

(b) For the config: teb_v0

Figure 5.2: Example of the measured signals for the dataset x.

For machine learning, the data-set is normally split into three sets:

- Training set
- Test set
- Validation set

The model is trained on the training set, and the performance of the model is tested on the test set and later validated on the validation set. This data-set separation method is called cross-validation, and is done to avoid over-fitting.

The five different runs are used to construct the train and test sets for cross validation. Four sets are used for training, and one set is used for testing. An extra data-set is later constructed in a different environment. There are five different combinations for the train and test sets (see table 5.2).

Table 5.2: The division of the train and test sets.

| Dataset | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| run0 | Train | Train | Train | Train | Test |
| run1 | Train | Train | Train | Test | Train |
| run2 | Train | Train | Test | Train | Train |
| run3 | Train | Test | Train | Train | Train |
| run4 | Test | Train | Train | Train | Train |

## 5.3. Model type choice

A list of requirements is set up to help with the choice of the model type.

To model the relation between the quality attributes and the environment metrics requires a supervised regression method. There are only 2 input features and a single output. It is desired to minimize the pre-processing steps for the data, since the quality models need to be used at run-time. It is desired to have a fast and memory efficient method to minimize the computational expenses.

Requirements:

- Supervised
- Regression
- Suited for low amount of features
- Single output method
- Minimal data pre-processing
- Fast and memory efficient method

From the available methods in the Scikit-learn library, the first methods that meet the requirements are the Simple Linear Regression and the Polynomial Regression. Since initial tests showed that a simple linear model is not sufficient, the Polynomial Regression model will be used.

**Simple Linear Regression**

This method assumes that the relationship between explanatory variables and the response variable is linear. The equation, in case there are two explanatory variables, is:

$$\hat{y(w,x)} = w_0 + w_1 x_x + w_2 x_2 \tag{5.1}$$

Where $x_k$ are the explanatory variables, and $w_k$ are the coefficients that need to be tuned/trained/learned/-fitted.

The coefficients can be trained using the Linear Least Squares method.

**Polynomial Regression**

This is a variation on the Linear Regression method. The relation between explanatory variables and the response variable is not assumed linear, but this method is linear in the parameters (coefficients). Additional terms are added; interaction terms and higher order terms. The equation for a polynomial degree of 2:

$$\hat{y(w,x)} = w_0 + w_1 x_x + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2 \tag{5.2}$$

Since the additional terms can be added in the data pre-processing step, the equation is linear and can be solved with the Linear Least Squares method.

## 5.4. Training and evaluation of the models

There are two popular scoring metrics for regression in scikit-learn: the coefficient of determination ($r^2$), and the mean squared error (mse).

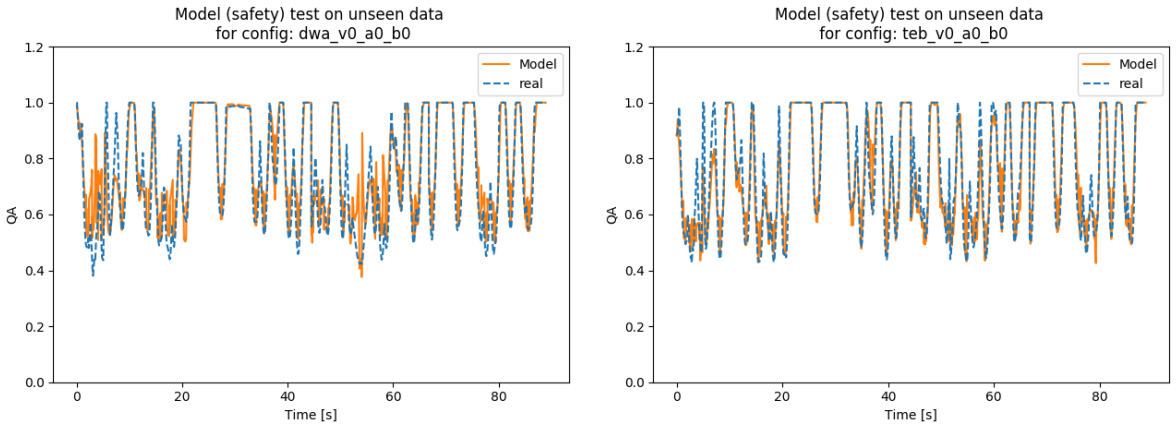
The coefficient of determination, which is the default scoring metric for regression problems, measures the amount of variance in the predicted data versus the original data [42]. Another popular scoring metric is the mean squared error, which is defined as the average of the squared error of each predicted point with the data-point. Both metrics will be used in the evaluation of the models.

Models are trained for the first two configurations to test and choose the parameters. Since there are five different data-set combinations, each model is trained five times and the model with the best score is selected.

An example of the results of the scoring metrics for the best model for safety for the DWA configuration can be seen in table 5.3, and the results are visualized in Figure 5.3a and 5.3b.

Table 5.3: Results on the test set, in terms of the r2score and the mean squared error.

| model | config | polynomial | r2 | mse |
|---|---|---|---|---|
| Safety | dwa_v0 | 4 | 0.7596 | 0.009284 |
| | teb_v0 | 4 | 0.8814 | 0.004797 |




(a) For the config: dwa_v0

(b) For the config: teb_v0

Figure 5.3: Tests of the Safety model using the test set.

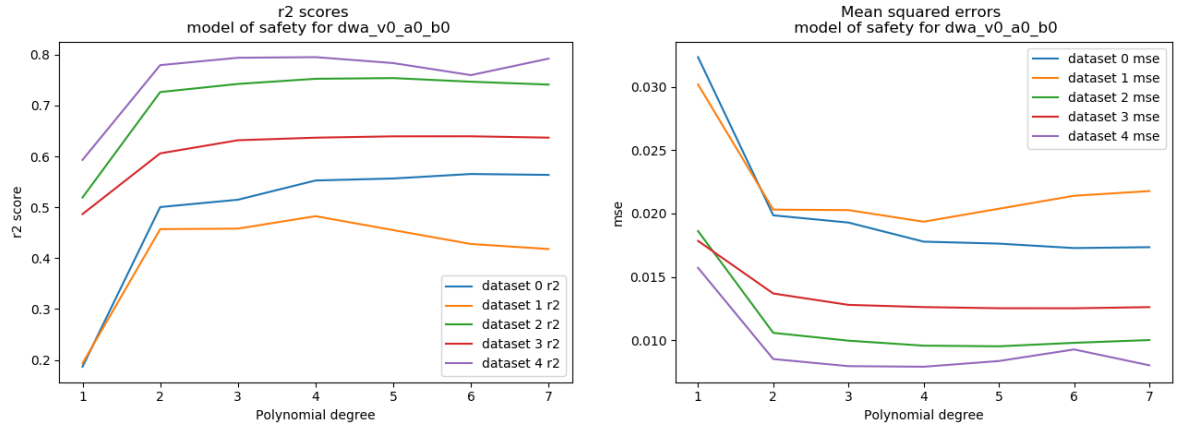
## 5.5. Hyperparameter tuning

The polynomial regression method only has 1 hyperparameter: the polynomial degree. Different polynomial degrees are tested. It can be seen(figure 5.4a and 5.4b) that a higher polynomial degree, up to a polynomial degree of 4, improves the results.

For the Safety models, a polynomial degree of 4 is the selected polynomial for all configurations (to make the pre-processing step the same for all configurations).

## 5.6. Quality model validation

Extra experiments are done to generate a validation data-set, for which the environment is different than the test and train sets. This is done to avoid over fitting on specific environment parameters and validate that the model performs well in different environments as well.




(a) The r2scores per polynomial degree for dwa_v0

(b) The mse scores per polynomial degree for dwa_v0

Figure 5.4: Training scores per polynomial degree .


Figure 5.5: Environment for the validation test.

Validation runs are done the in the retail store environment. The design of this environment is discussed in appendix **??**.

The results for the TEB the configuration can be seen in figure 5.6.

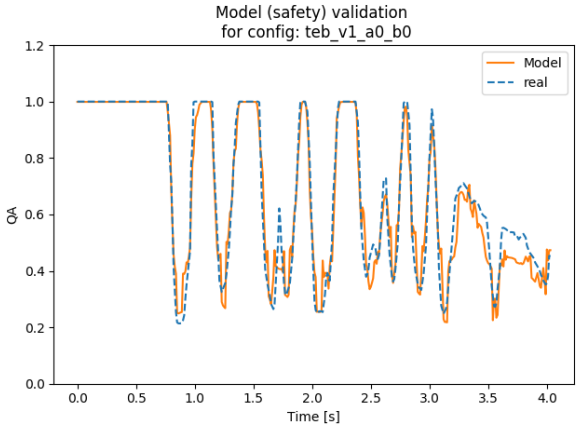



Figure 5.6: Results for the configuration teb_v1, perceived compared to the predicted safety level in the validation environment

## 5.7. Discussion

In this chapter, a method to construct the quality models is developed. 7 steps are used to construct the models: data collection, data preparation, model type choice, training, evaluate the model, parameter tuning and validation. These steps are all followed to develop a method that can be used to build the

quality models for all configurations, which will be done in the next chapter.

### Results

Data is collected in different simulated corridors of a supermarket where the quality attributes and the environment characteristics are measured for each time-step.

Polynomial regression is the machine learning method that is used to train the models. The data pre-processing requirements are minimal for this method, and it is a fast and computational cheap method.

The results of the trained models show that the constructed quality models are a suitable method to predict the safety level of a system configuration based on the environment characteristics.

### Limitations and future work

Some simplifications are made for the data collection. Some runs fail because the local planner cannot find a feasible trajectory. In this case, the run is tried again (with a maximum of 3 tries). The results of the failed runs are discarded. These results can be however, very interesting for the knowledge about the performance. If a configuration often fails in a certain environment, the performance knowledge should be lower. A lower performance means that the self-adaptive system will be less likely to choose this configuration, and will avoid that the system will get stuck and therefore improve the overall performance at run-time.

In this research, data is collected to train the models before run-time. It is also possible to use the run-time data to improve the models further. This will also add more different environments/scenarios to the train data-set. However, it can be difficult to guarantee the quality of this data, since the experiments environments are less controlled.

# 6

# Self-adaptive framework

The MROS framework is used as a tool to obtain the self-adaptive capabilities. The quality and environment metrics and the quality models, that are developed in this research, are implemented in this framework. The last research sub-question is answered in this chapter: *How can the predicted quality levels be used in the self-adaptive framework to improve the navigation quality based on changes in the environment?*

First the general concept of the MROS framework is analysed. Then, the implementation of the new (sub)components in the framework is explained. Different modifications of the MROS framework are developed, to test different methods of using the quality models in this framework. After that, the configurations that the self-adaptive framework can choose from are developed, using different configurations of the DWA and TEB local planners.

## 6.1. MROS, default framework

Corbato et al. [36] state that the most important feature that differentiates MROS with respect to other frameworks is the strict separation of concerns. The model-based meta controller and the application logic are separated. The meta-controller ensures that the available configurations fulfill the objectives that are commanded by the application logic. The objectives consist of functional and non-functional requirements that are defined at design-time. The metacontroller adapts the runtime configuration of the the managed system according to the requirements and the dynamically changing context at runtime.

The MROS framework follows the MAPE-K feedback loop concept [43] (see Figure 6.1). The working principle of the MROS framework is explained in more detail following the MAPE-K loop phases.

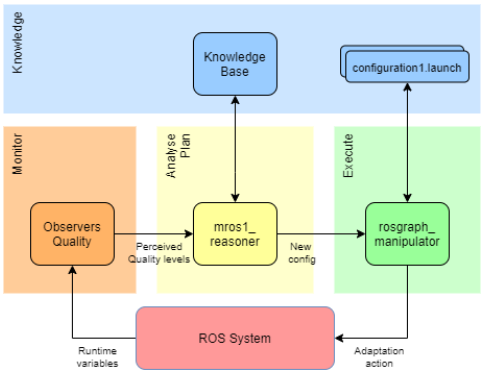



Figure 6.1: Overview of the default MROS framework.

### 6.1.1. Knowledge

The knowledge base contains the set of possible configurations of the managed components and their quality attribute values. The knowledge base conforms to the TOMASys ontology, the run-time model which contains the functional information about the managed system, the non-functional requirements, and the objectives.

The knowledge base is implemented using OWL, the Web Ontology Language. An advantage of using OWL is that an off the shelf reasoner can be used [44].

### 6.1.2. Monitor

The observers (rosgraph_monitor) monitor the application runtime context and the quality attributes of the current configuration. The information from the observers can be used to detect multiple different events: mis-configured parameters, erroneous behavior of components, violation of system requirements, and the level of the quality attributes. The information that is determined by the observers is passed to the reasoner.

### 6.1.3. Analyse, Plan

The updated information is used by the mros_reasoner to investigate if the current objectives and the non-functional requirements are violated. The mros_reasoner is an ontological reasoner, and uses the OWL extended with the Semantic Web Rule Language to reason.

If the reasoner finds an objective violation, the plan phase will be triggered. In this case, a search is done for the best alternative configuration that complies with the objectives and the non-functional requirements. This step is executed by the mros_reasoner using ontology search queries.

### 6.1.4. Execute

If a new system configuration is desired and found by the reasoner, the rosgraph_manipulator is triggered. This node will launch the new configuration. Since not all parameters can be dynamically changed in ROS1, the configuration nodes are killed and (re)launched with the new configuration.

## 6.2. MROS, modifications

The implementation of quality models and the other new components are explained in this section. The difference between the new and the default MROS system are analysed per phase (see Figure 6.2).

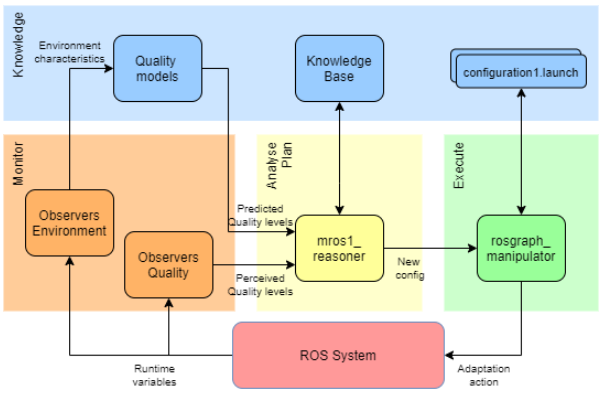



Figure 6.2: Overview of modified MROS framework.

### 6.2.1. Knowledge

The knowledge base ontology is equal to the default MROS ontology, build upon the TOMASys ontology. The configuration launch files in the knowledge base are updated with the new configurations. Next to this ontology and the launch files, the quality models are also implemented as knowledge. A separate rosnode is created, where the quality models can be loaded in when the robot in launched. This rosnode will subscribe to the topics where the observers are publishing in, and will send on request the predicted quality levels of the configurations.

### 6.2.2. Monitor

Next to the quality observers that will monitor the current safety and performance levels, environment observers are implemented. These environment observers will monitor the environment, using data from the robot's sensors, and measure the Obstacle Density and Narrowness levels.

### 6.2.3. Analyse, Plan

The mros_reasoner is used, but some modifications are made. These modification are needed to implement the quality models, and to research different methods of how the quality models can be used. This is in more detail explained in section 6.3.

### 6.2.4. Execute

The rosgraph_manipulator is not modified. The only difference is that the launch files of new configurations are used by the rosgraph_manipulator.

## 6.3. Reasoning cycle modification

### 6.3.1. Modification 0: Default MROS system.

This is the default MROS system. The knowledge base is updated with the configurations of this research. The QA values in the knowledge base are initialized with the average quality values from experiments.

The reasoner flow of the default MROS system is displayed in Figure 6.3. If the NFR is violated, the used configuration gets an objective error and cannot be used again. The list of configurations without objective error is further filtered to only contain the configurations that meet the NFRs. When no configuration meets the NFRs, no adaptation takes place. When more than one configuration meets the NFRs, a utility function is used. The utility function uses the estimated performance levels, and the configuration with the highest estimated performance level is selected.

### 6.3.2. Modification 1: MROS with quality models.

This is an updated version of the default MROS reasoner flow where the quality models are implemented. The reasoner flow is displayed in Figure 6.4.

The reasoner first tests if the NFRs are violated. If this is the case, the quality models are used to update the predicted safety values in the knowledge base. The reasoner now has more information than the default MROS system. The configurations that once were in violation are not discarded anymore, since it can be predicted if they will be in error now.

From the list of configurations that meet the NFRs, the one with the highest performance is selected, equal to the default MROS flow. But when there are no configurations that meet the NFRs, the configuration with the highest safety level is selected. This is done since this will improve the safety level, and therefore the navigation quality. This different from the default MROS system, since the default system would not adapt in this case.

### 6.3.3. Modification 2: MROS with quality models, and new adaptation rules

The second modification is an update over the first modification, the mod2 reasoner flow is displayed in Figure 6.5.

The only difference with the first modification is that in the case that the NFRs are not in violation, adaptation is still an option. In this case it is tested if the system can switch back to the initial configuration. It is assumed that the system is launched with an initial configuration that is the desired configuration. Therefore the option to switch back is analysed. Since the initial configuration is usually the fastest configuration, the hypothesis is that average performance level will be improved when switching back is enabled.

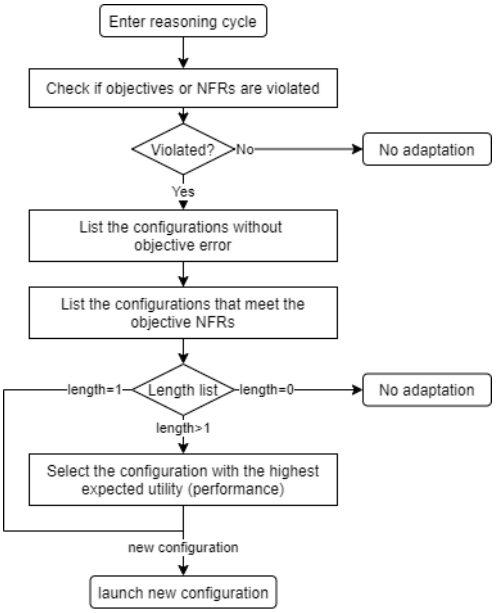



Figure 6.3: Reasoner flow MROS default (m0).

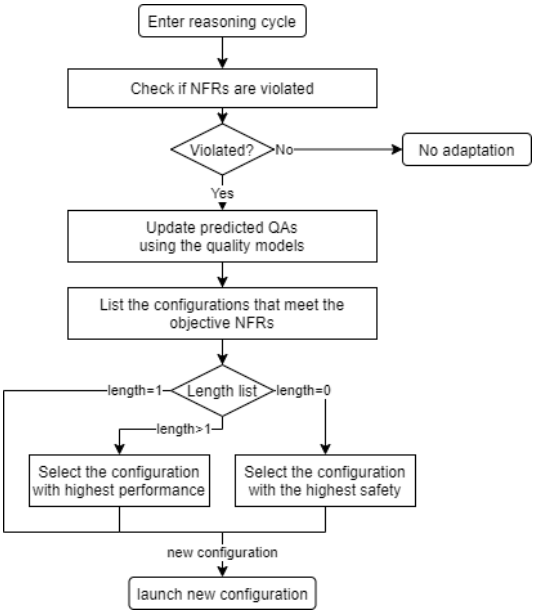



Figure 6.4: Reasoner flow modification 1.

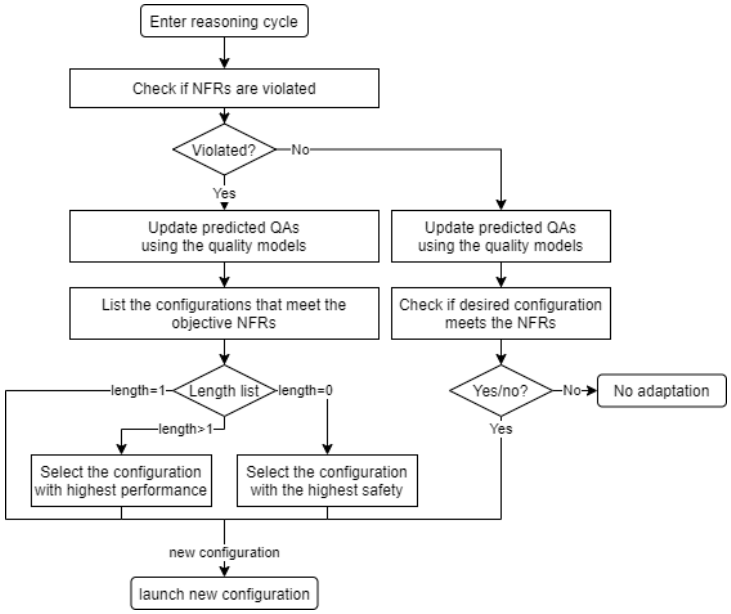



Figure 6.5: Reasoner flow modification 2.

## 6.4. Configuration construction

The self-adaptive system uses the different system configurations to adapt according to the objectives and the changing context at runtime. In this section, the different configuration parameters are analysed to construct the configurations. The quality models for the configurations are trained using the steps that are developed in the previous chapter.

### 6.4.1. Configuration parameters

The parameters of both the DWA and the TEB local planners are analysed, and their dependency on the quality attributes are estimated. It is chosen to use the maximum speed (parameter v), and 2 parameters (parameter a and b) per local planner as the configuration parameters. For the DWA local planner, these configuration parameters are the sim_time and the scaling_speed. The configuration parameters for the TEB local planner are the inflation_radius and the weight_obstacle. Their dependency estimations on the quality attributes are shown in table 6.1.

Table 6.1: Configuration parameters

| | Dependency | | |
|---|---|---|---|
| Parameter | Local planner | Performance | Safety |
| max_vel_x | both | Higher | Lower |
| sim_time | DWA | Higher | Lower |
| scaling_speed | DWA | Higher | Lower |
| inflation_dist | TEB | Lower | Higher |
| weight_obstacle | TEB | Lower | Higher |

The configuration parameter values are selected as shown in table 6.2. It is however, decided to not use the configurations with a maximum speed of 1.0 m/s to reduce the negative effects of some implementation issues. This is explained further in section 6.5.2.

### 6.4.2. Configuration quality models

The safety model for all configurations is trained using the steps that are developed in the previous chapter. The validations scores are shown in table 6.3. The knowledge base will be initialized with the average performance and safety scores (see Figure 6.6).

Table 6.2: Configuration parameters

(a) For the dwa local planner

| | v | a | b |
|---|---|---|---|
| Configuration | max_ vel_x | sim_ time | scaling_ speed |
| dwa_v0_a0_b0 | 1.0 | 1.0 | 1.0 |
| dwa_v0_a1_b0 | 1.0 | 1.7 | 1.0 |
| dwa_v0_a0_b1 | 1.0 | 1.0 | 0.25 |
| dwa_v0_a1_b1 | 1.0 | 1.7 | 0.25 |
| dwa_v1_a0_b0 | 0.8 | 1.0 | 1.0 |
| dwa_v1_a1_b0 | 0.8 | 1.7 | 1.0 |
| dwa_v1_a0_b1 | 0.8 | 1.0 | 0.25 |
| dwa_v1_a1_b1 | 0.8 | 1.7 | 0.25 |
| dwa_v2_a0_b0 | 0.6 | 1.0 | 1.0 |
| dwa_v2_a1_b0 | 0.6 | 1.7 | 1.0 |
| dwa_v2_a0_b1 | 0.6 | 1.0 | 0.25 |
| dwa_v2_a1_b1 | 0.6 | 1.7 | 0.25 |

(b) For the teb local planner

| | v | a | b |
|---|---|---|---|
| Configuration | max_ vel_x | inflation_ dist | weight_ obstacle |
| teb_v0_a0_b0 | 1.0 | 0.5 | 50 |
| teb_v0_a1_b0 | 1.0 | 0.8 | 50 |
| teb_v0_a0_b1 | 1.0 | 0.5 | 80 |
| teb_v0_a1_b1 | 1.0 | 0.8 | 80 |
| teb_v1_a0_b0 | 0.8 | 0.5 | 50 |
| teb_v1_a1_b0 | 0.8 | 0.8 | 50 |
| teb_v1_a0_b1 | 0.8 | 0.5 | 80 |
| teb_v1_a1_b1 | 0.8 | 0.8 | 80 |
| teb_v2_a0_b0 | 0.6 | 0.5 | 50 |
| teb_v2_a1_b0 | 0.6 | 0.8 | 50 |
| teb_v2_a0_b1 | 0.6 | 0.5 | 80 |
| teb_v2_a1_b1 | 0.6 | 0.8 | 80 |

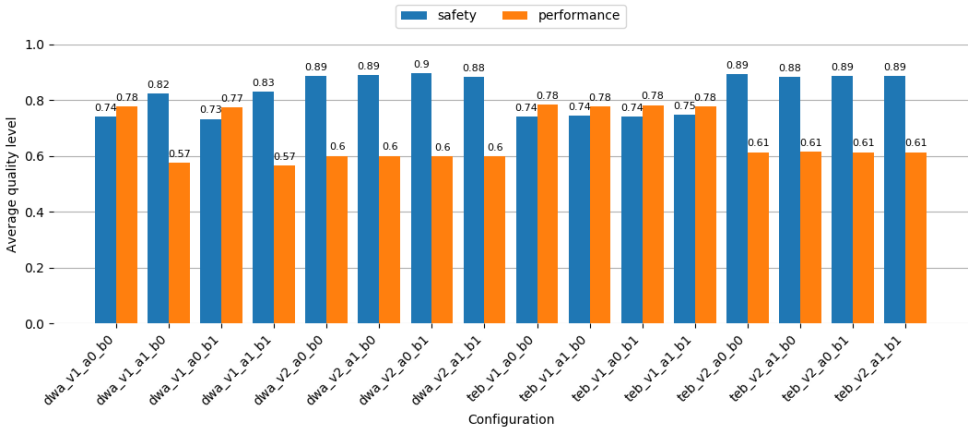



Figure 6.6: The average quality levels for the configurations.

Table 6.3: Model training scores

(a) for the DWA configurations

| | Safety model | |
|---|---|---|
| Configuration | r2score | mse |
| dwa_v1_a0_b0 | 0.9151 | 0.0068 |
| dwa_v1_a1_b0 | 0.8963 | 0.0093 |
| dwa_v1_a0_b1 | 0.9256 | 0.0057 |
| dwa_v1_a1_b1 | 0.9079 | 0.0078 |
| dwa_v2_a0_b0 | 0.8201 | 0.0055 |
| dwa_v2_a1_b0 | 0.8437 | 0.0059 |
| dwa_v2_a0_b1 | 0.8212 | 0.0054 |
| dwa_v2_a1_b1 | 0.8318 | 0.0071 |

(b) For the TEB configurations

| | Safety model | |
|---|---|---|
| Configuration | r2score | mse |
| teb_v1_a0_b0 | 0.9479 | 0.0039 |
| teb_v1_a1_b0 | 0.9435 | 0.0043 |
| teb_v1_a0_b1 | 0.9528 | 0.0036 |
| teb_v1_a1_b1 | 0.9502 | 0.0038 |
| teb_v2_a0_b0 | 0.8555 | 0.005 |
| teb_v2_a1_b0 | 0.8619 | 0.005 |
| teb_v2_a0_b1 | 0.8707 | 0.005 |
| teb_v2_a1_b1 | 0.8511 | 0.0053 |

## 6.5. Discussion

In this chapter, the quality models that were developed in the previous chapter, are implemented in the MROS framework and the different navigation configurations are constructed.

### 6.5.1. Results

Two different modifications on the MROS framework are developed in this chapter (see figures 6.3, 6.4, 6.5). The first modification adds the quality models, and uses the prediction from the models in the plan phase. The second modification adds an extra analyse phase. The prediction of the the quality models can also be used to analyse if switching back to the initial configuration is possible. The hypothesis is that this will improve the overall performance of the system.

The parameters of the local planners are analysed and their dependency on the quality attributes is estimated. Based on this dependency estimation, the system configuration are constructed. For the configurations, two different local planners are used together with three different maximum speeds. And from both local planners: two configurations parameters are selected (see tables 6.1, 6.2). This results in 24 different navigation configurations that the self-adaptive system can use for adaptation.

The quality models are trained for all configurations, and the validation results are shown in table 6.3.

### 6.5.2. Limitations and future work

In this section, the limitations and ideas to improve the framework and the configurations are discussed.

### Adaptation response delay

Table 6.4: The ROS time each MAPE-K phase takes. (Intel Core i9-9900K @ 4.7 GHz)

| | Monitor | Analyse | Plan | Execute |
|---|---|---|---|---|
| Default MROS | 97 | 500ms | 1ms | 3575ms |
| Modified MROS | 97 | 500ms | 16ms | 3575ms |

Each step in self-adaption process adds a delay in the adaptation response. A large delay is not desired since this means that when a NFR is violated, the adaptation response of the system is executed too late and the robot will drive in an unsafe situation.

The time each MAPE-K phase takes is measured in ROS time and shown in table 6.4. ROS time is used since this is considered more relevant than real time, since ROS time is what the robot experiences.

The execution phase is the phase that takes the longest time, since move_base is killed and launched again. Since the navigational system is not active during this time, the robot brakes and comes to a stop. The overall performance of the system will be decreased and since the velocity decreases, the level of safety will increase. The focus of this research is not the optimization of the execution speeds. The results will be filtered to remove the influence of these delays in the rest of this research.

A solution to this could be to only use parameters that can be dynamically changed in ROS1, but this will limit the amount of configurations. ROS2 introduces the concept of system modes, which makes it possible to dynamically change the configuration of a node [45].

### Reasoning rate

The reasoning cycle is executed periodically with a fixed frequency. The MROS (inverted) reasoning rate is by default once every 2.0 seconds. The sum of the delays of all the reasoning processes results in a maximum adaptation response of 2551ms. This is the time between the NFR violation and the start of the execution phase.

Since 2551ms is considered too long, the (inverted) reasoning rate is decreased to the minimum: once every 0.6 seconds. Since the monitor and analyse phase take 597 ms, the reasoning rate cannot be lower than this value. A reasoning rate of 0.6 seconds results in a maximum adaptation delay of 951ms.

951ms is still a large adaptation delay, and this results directly into NFR violation time. More research should be done on optimizing the reasoning process.

### Initialize after relaunch

The local costmap is cleared after move_base is killed and relaunched. This means that the observed Obstacle Density value is equal to zero initially. Modification 2 uses this observed value to predict the safety level of the initial configuration and test if the system can switch back. Since this will result in a predicted safety level of 1, the system will always immediately switch back after adaptation to a safer configuration took place.

To give the system the time to initialize and fill the costmap again, a counter is implemented in the reasoning cycle, Switching back is not allowed for the first 4 reasoning cycles (on average 2.4 s) after adaptation.

Since in this time the global path is also not available and not replanned, the measurements of the performance run-time metric are not realistic and therefore not usable. This run-time metric will therefore not be used to measure the performance level in experiments where the self-adaptive system is used.

### Configuration maximum speed

Since there is a relatively long delay in the adaptation response, it is decided to not use the highest maximum velocity for the configurations. Only the configurations with a maximum velocity of 0.8 m/s and 0.6 m/s are used. This is done to reduce the distance that the robot moves during the execute phase, which can result in the robot come very close to obstacles.

**Configuration parameters**

The parameters of the local planners are analysed and their dependency on the quality attributes is estimated. This is done to be able to select the parameters that will results in configurations with different behaviors. However, the results of the average performance and safety levels (figure 6.6 are very similar for the different TEB configurations. The goal of having different configurations was to have different behaviors for the system, which is currently not the case for the TEB configurations.

This could be improved in future work by doing experiments first to optimize the configuration parameter values.

# 7

# Experiments

All the new components are now developed and integrated in the MROS framework. Different modifications of the MROS framework are developed to test how the quality models can be used and to test if the navigation quality will be improved. Hence the experiment question is: *Is the use of quality models useful for the self-adaptive system?* The different modifications of the self-adaptive system are compared against default ROS systems(fixed configurations).

First, the experimental setup is explained by defining the simulation environment and scenarios. The systems and the benchmarks that will be used in the experiment are briefly discussed. Different metrics are defined to evaluate the results and to be able answer the experiment research question. After that, the results of the experiments are analysed followed by a discussion.

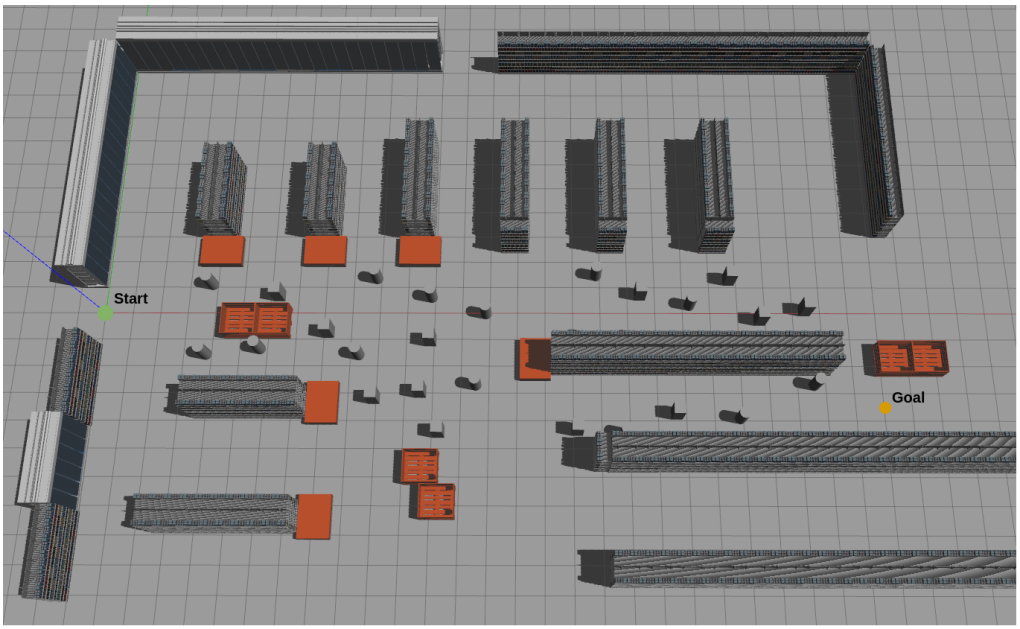



Figure 7.1: The simulation world for the experiments.

# 7.1. Experimental setup

## 7.1.1. Albert Heijn environment scenarios

The floorplan of a real supermarket, the Albert Heijn XL in Delft, is used to copy the layout of (a part of) the store. This is done to create a realistic simulation environment (see Figure 7.2).

The mission in this experiment is to navigate from the start position to the goal position. The start and the goal position are: start_position = $[0, 0, 0]([x, y, yaw])$, goal_position = $[28, -2.5, 0]$ as shown in Figure 7.2.

Two different area's between start and goal are defined where obstacles can be spawned, as indicated in Figure 7.2. The amount of obstacles is determined by the area size and the obstacle clutterness parameter. To simulate the change from a normal environment from or to a cluttered environment three different scenarios are simulated, see table 7.1.

Five runs are done for each scenario and the results are averaged to improve the quality of the results.

Table 7.1: The configuration of the different experiment scenarios.

| | Obstacle clutterness | |
|---|---|---|
| | area1 | area2 |
| Scenario 1 | 0.4 | 0 |
| Scenario 2 | 0 | 0.4 |
| Scenario 3 | 0.4 | 0.4 |

## 7.1.2. Systems

The default and the two new developed version the MROS framework will be tested:

- Default MROS, modification 0
- MROS with quality models, modification 1
- MROS with quality models, modification 2

MROS modification 0 is the default MROS framework, without quality models. The quality models are implemented in MROS modification 1. MROS modification 2 allows the system to switch back to the initial configuration, and is an update over modification 1. The MROS modifications are explained in more detail in chapter 6.3.

The benchmarks will be 'fixed configuration' ROS navigation systems. Two configurations are selected: a 'fast' configuration (dwa_v1_a0_b0) and a 'safe' configuration (dwa_v2_a1_b0). These are selecting based on their average safety and performance levels which are determined in the previous chapter (see Figure 6.6).

The non-functional requirement that is used by the self-adaptive systems is a safety level of 0.6. When the measured safety level is lower than this value, adaptation is triggered.

Table 7.2: The systems and that will be used in the experiments

| system | (initial) configuration | Quality models | Adaptation rules | Non-functional requirement |
|---|---|---|---|---|
| benchmark_fast | dwa_v1_a0_b0 | No | None | None |
| benchmark_safe | dwa_v2_a1_b0 | No | None | None |
| SA_mod0 | dwa_v1_a0_b0 | No | Default | Safety: 0.6 |
| SA_mod1 | dwa_v1_a0_b0 | Yes | Default | Safety: 0.6 |
| SA_mod1 | dwa_v1_a0_b0 | Yes | With switching back | Safety: 0.6 |

### 7.1.3. Experiment evaluation metrics

The experiment metrics are the independent variables that are used to evaluate the results.

The improvement in safety will be measured with two different metrics: the average safety level over the complete run, and the time that the NFR is violated.

Performance will be measured using the time to completion. For the self-adaptive systems, the time that the execute phase takes is subtracted from the time to completion, as explained in 6.5.2.

Experiment evaluation metrics:

1. Time to completion (performance)
2. Average safety level
3. Time violating the non-functional requirement

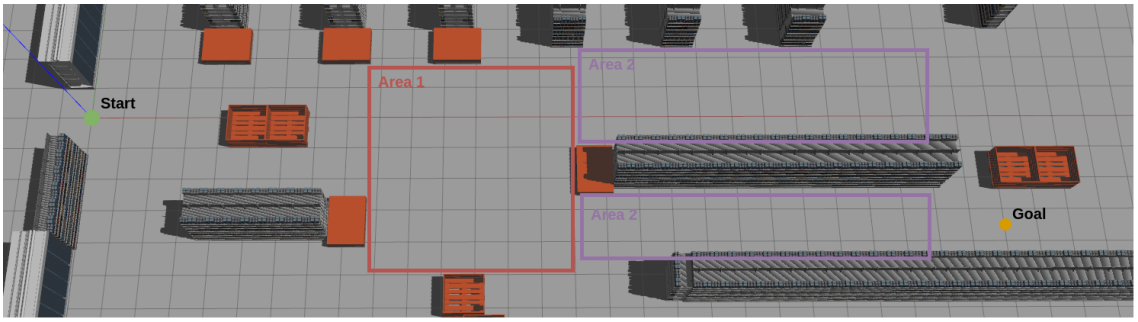



Figure 7.2: The simulation environment for the final experiments. The 2 areas where obstacles will be spawned are indicated.

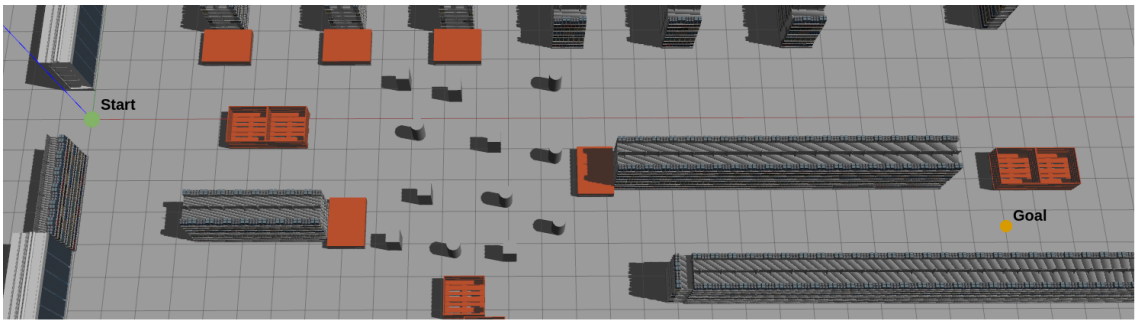



Figure 7.3: The simulation environment for the final experiments, Scenario 1.

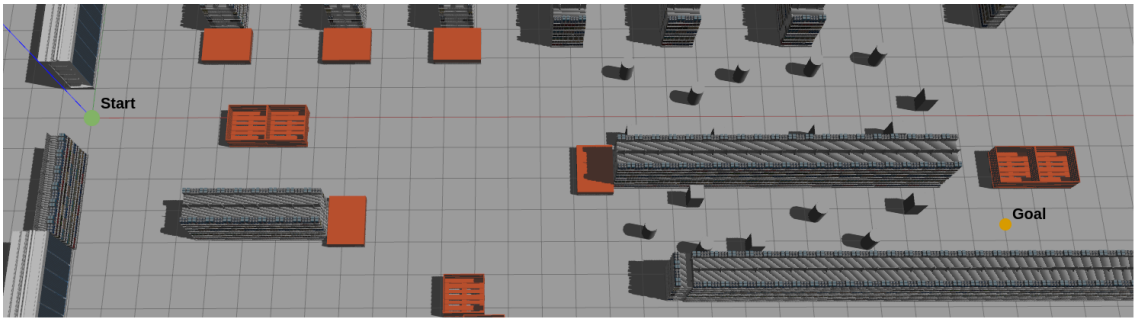



Figure 7.4: The simulation environment for the final experiments, Scenario 2.

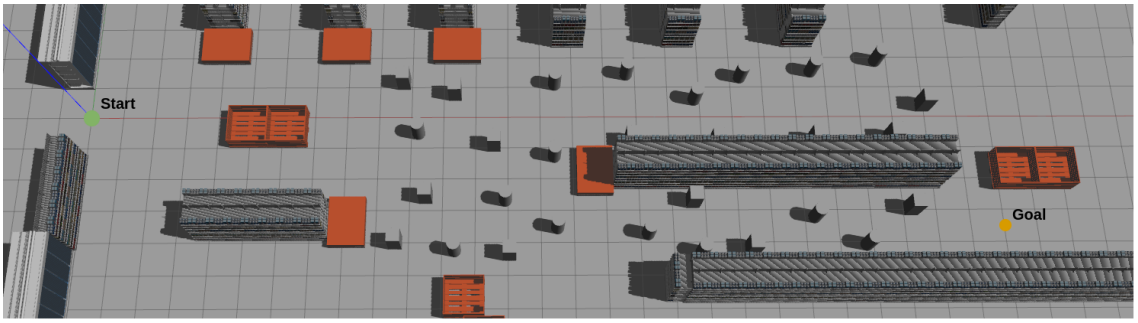



Figure 7.5: The simulation environment for the final experiments, Scenario 3.

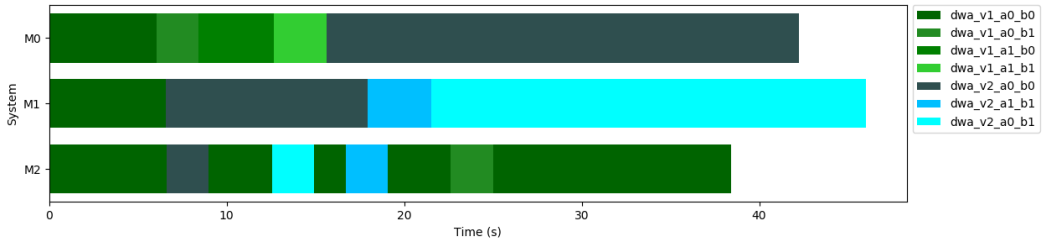
## 7.2. Experiment results

The results of the experiments are analysed and evaluated using the three experiment metrics in the following sections.

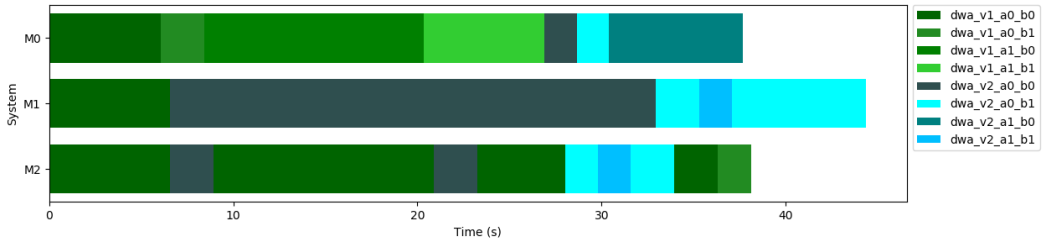
### 7.2.1. Adaptation choices

It can be seen in Figure 7.6 that the different MROS modifications do make different reconfiguration choices.

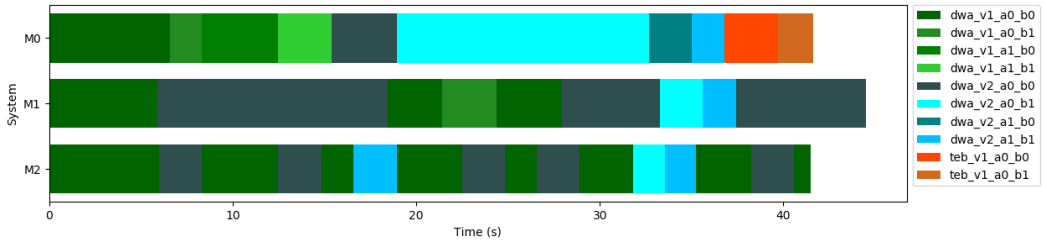
Modification 0 switches to the next best configuration at every adaptation action. This results in the




(a) For scenario 1




(b) For scenario 2




(c) For scenario 3.

Figure 7.6: Adaptation choices of the different self-adaptive systems.

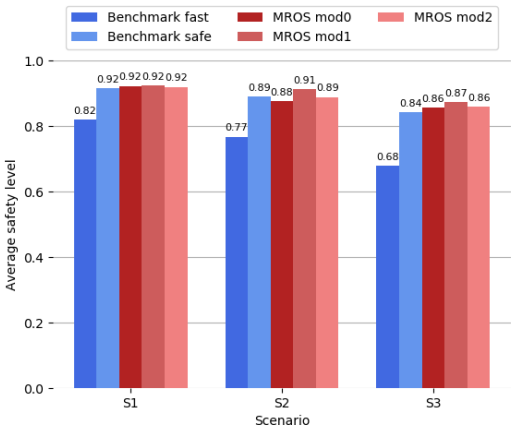
same order of re-configurations in all three scenarios. This is because the system does not have information about the environment: about which configuration is currently in violation. The system only knows what the next best configuration is. The configuration that was in violation in the past cannot be chosen again in this system, so the next best configuration is chosen at every adaptation action.

The adaptation choices are different for modification 1 compared to modification 0. This is because this system does have predicted information about the safety level of all the configurations in the current environment. The first adaptation action is therefore not to the next best option, but to the best performing configuration that is predicted to not violate the NFR in the current environment. Since the list of configurations that did not violate the NFR the past, the system also switches back to an earlier used configuration. It can be seen that the configurations dwa_v1_a0_b0 and dwa_v1_a0_b0 are used multiple times in Figure 7.6b.

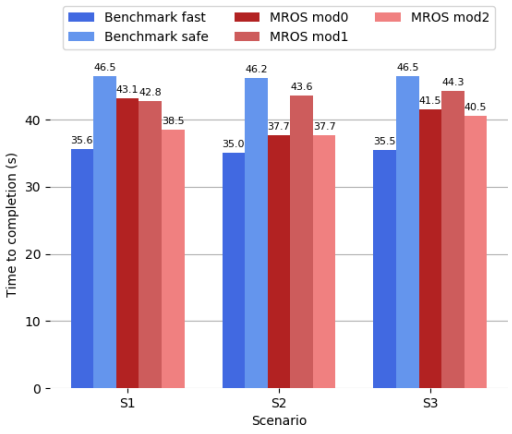
The second modification makes totally different choices. It can be seen that almost the entire time, the initial configuration (dwa_v1_a0_b0) is used. Only for multiple short periods safer configurations are used. The system switches to a safer configurations when the NFR is violated, and switches back to the initial configuration if the predicted safety level of the initial configuration is high enough to not violate the NFR anymore.

### 7.2.2. Evaluation metrics

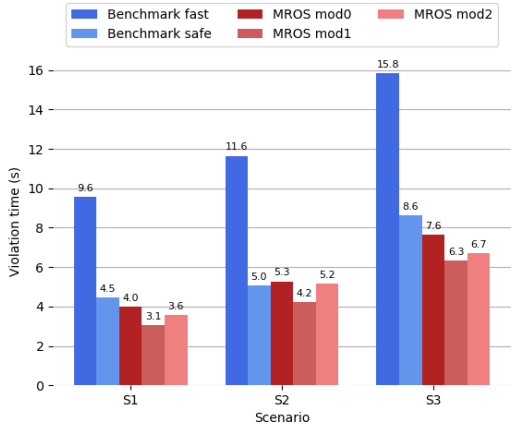
The average evaluation measures of five runs per scenario are shown in Figure 7.7.




(a) Safety




(b) Time to completion




(c) Violation time

Figure 7.7: Metrics

Modification 0 does not use the quality models. This system cannot predict which configuration will be in error. Therefore. the system just chooses the next best configuration, based on the average safety and performance levels from the knowledge base. It can be case, however, that this configuration will also violate the non-functional requirement.

Modification 1 does use the quality models. When adaptation is needed, the quality models will update the QA levels in the knowledge base based on the current environment. By doing this, the reasoner knows which configurations will be violating the NFR in the current environment, and can therefore make a better informed choice for adaptation. The result is less violation time and improved average safety compared to modification 0. The cost of this is a lower performance since more often a safer configuration is chosen instead of the next best option, which decreases the performance level.

Modification 2 is able to switch back to the desired (initial) configuration when there are no NFR violations. The result is a lower average safety compared to modification 1, and a higher violation time. This is because, after switching back, in the next unsafe situation that the robot encounters, the fast and less safe configuration is used. But the improvement of this modification can be seen in the performance level. Since it is possible to switch back to the faster configuration when the environment allows this, the time to completion is lower compared to modification 1.

## 7.3. Discussion

*Is the use of quality models useful for the self-adaptive system?* It can be concluded that the use of quality models in this self-adaptive framework is indeed useful. The effect of using the quality models is that the average safety level increases and the NFR violation time decreases for modification 1. The cost of this is however, a lower performance level.

For modification 2 the results can be seen in an increased performance, at the cost of a lower average safety level and longer violation time. The navigation quality of modification 2 is similar to the default MROS system (modification 0).

**Future work** For modification 2, the navigation quality is not improved compared to modification 0, but the result are promising for future research on a proactive approach. With the introduction of the quality levels, it is possible to change the reactive approach of the framework into a proactive approach. IT is possible to predict the quality levels and the violation of the NFR in the future (1 second in front of the robot for example). By doing this, the drop in safety can be predicted and adaptation can take place before the drop in the safety level actually happens. This will therefore improve safety a lot (the drop in safety level is prevented). The cost of this will probably be a bit lower performance, since the switch to a safe configuration is done earlier (so the safer configuration is used for a longer time). But when switching back to a fast configuration is allowed, the performance will increase again.

# 8

# Conclusions and future work

This aim of this thesis was to improve the navigation quality of a mobile robot based on changes in the environment. This work proposed a definition for navigation quality, and identified relevant characteristics of the environment in relation to the navigation quality. These definitions were used to train the robot's knowledge about its navigation quality in changing environments. This knowledge is tested and is proven to be useful for the reasoning process in a self-adaptive framework in simulation experiments.

## 8.1. Conclusions

The main research question for this research is: *1) How can navigation quality be improved for mobile ground robots based on changes in the environments using a self-adaptive framework?* This question is answered with the help of the research sub-questions.

*1.1) How to define navigation quality for mobile robot in retail environments?*
Navigation quality is for this research defined in terms of performance and safety. Robots in industry are used to improve the level of productivity, which makes performance an important focus. Next to that, the robots will share the workspace with humans in for example retail stores. The shared workspace makes the problem of navigation more challenging since the movement of humans can be very unpredictable. Safety is one of the main challenges in these situations. Although dynamic obstacles are not used in this research, the focus is to improve the safety to prepare for these scenarios.

The safety metric is defined using a safety margin around the robot with the size of the braking distance. For obstacles within this safety margin, the safety level scales linearly with the obstacle distance.

Performance can be measured with the time to completion, but this is not a run-time metric. A performance run-metric is developed in this research: the time to completion ratio. For this ratio, the time to completion of the actual path and the global path in the past second are compared to obtain a run-time measure for performance.

*1.2) Which environment characteristics are relevant for the navigation quality?*
Metrics are developed to measure two characteristics of the environment: Obstacle Density and Narrowness.

Obstacle Density is measured in a local window in front of the robot, and is defined as the ratio between occupied space area and the total area. The Narrowness metric is defined as the amount of robot widths that fit in the current environment. This is measured at the center of the robot, in the direction perpendicular to the direction of travel.

Validation test show that there indeed is a correlation between the environment metrics and safety. Therefore it can be concluded that Narrowness and Obstacle Density are relevant environment metrics in relation to safety. For performance however, there is not a high correlation.

*1.3) How can the navigation quality of the system configurations be predicted, using the characteristics from the environment?*
To be able to make a prediction about the navigation quality of the configurations in the current environment, quality models are constructed. The input for these quality models are the measured environment metrics, and the output is the predicted safety level. These quality models are trained using a machine learning method: polynomial regression. It can be concluded that for safety, a polynomial model can be used to predict the safety level of the configurations based on the measures of the environment characteristics.

*1.4) How can the predicted quality levels be used in the self-adaptive framework to improve the navigation quality based on changes in the environment?*
Two different modifications of the MROS framework with the quality models are compared to the default MROS framework and to two fixed configuration benchmarks. Is is shown that quality models are useful for the self-adaptive framework.

The first modification of the MROS framework is using the quality models. The reasoner can make a better informed decision about adaptation, and therefore the average safety level is improved. Next to that, the violation time is decreased as well. The cost is a lower performance, because safer and therefore slower configurations are used.

The second modification is an update over the first modification. This update enables the possibility for the system to switch back to a 'faster' configuration. This results in a higher performance level, at the cost of a lower average safety level and a higher violation time.

## 8.2. Future work

During this research, multiple ideas to improve the metrics, models, and framework are emerged. These are already explained in more detail in the discussion sections of the previous chapters. The future work ideas are summarized here, categorized per chapter.

### 8.2.1. The metrics for navigation quality and the environment characteristics

For the safety margin, the area in front the robot can be considered more important, since the braking distance is the most relevant for this area.

In this research the safety level scales linearly with the obstacle distance. It can be interesting to consider a non-linear scale, since the risk of collision gets higher when obstacles are closer to the robot.

The environment metrics use a window size or maximum range. This value can be based on the robot size or even based on the size of the safety margin. This does require further tests and validation.

The environment characteristic metrics that are developed in this research do not show a high correlation with performance. A higher correlation could be obtained by modifying these metrics or developed new metrics.

### 8.2.2. The quality models

During the data collection to train the model with, simulation runs that failed were discarded. This information is however, interesting to consider for the knowledge about the performance. The self-adaptive system should not choose the configurations that will often fail in a particular environment.

In this research, the models are trained at design-time and are static at run-time. It can be interesting to improve the models further using data that is collected at run-time. Further research should be done to ensure the quality of the data that is used to improve the models further.

### 8.2.3. The local planner configurations

The parameters of the local planners are analysed and their dependency on the quality attributes are estimated. This information is used to define the different values for the configuration parameters. Experiments could be done to optimize the configuration parameters with the goal to cover a wider range of average safety and performance levels in the configurations.

### 8.2.4. The MROS framework

Several improvements for the MROS framework are proposed:

### 8.2.5. Adaptation execution time

The execution of adaptation is done by killing and relaunching the new configuration. This does take a long time and the robot stops moving during this process. This is an implementation issue and therefore not focused on for this research. This can be solved by choosing only parameters that can be dynamically adapted in ROS1 but this will limit the possibilities. ROS2 introduces the concept of system modes, which makes it easier to switch between configurations at run-time.

This long execution time has several consequences:

- The robot decelerates and stops, but does move for a short distance. This can result in the robot to come very close to an obstacle, which actually was tried to avoid by the adaptation action.
- The robot stops moving for 3.5 seconds, so the performance does decrease a lot.
- The data from the measured safety levels need filtering in order to be useful.
- The data from the measured performance levels are affected even more, and is not used in this research to measure the performance of the self-adaptive systems.
- The local costmap is cleared, so after relaunching the navigation system the Obstacle Density measure is not immediately reliable.

**Reasoning delay**

the analyses of the violation of the NFRs actually does take a long time, 0.5 seconds ROS time. The total delay between the violation of the NFR and the start of the execute phase is minimized to 951 ms, during which the robots keeps on moving in a ’unsafe’ behavior. The reasoning time could maybe be decreased by doing a smaller reasoning cycle first, where only the violations of the requirements are checked.

**Reactive to proactive approach**

The MROS framework can be improved further by predicting the quality levels and the NFR violations in the future (one second in front of the robot for example). By doing this, the drop in safety can be predicted and adaptation can take place before the drop in the safety level actually happens. This will improve the level of safety since the drop in safety is prevented. The cost of this will be a lower performance, since the switch to a safe configuration is done earlier (so the safer configuration is used for a longer time). This will change the reactive approach of the framework into a proactive approach and will make the quality models even more useful.

# Appendices

# A

# Project description

This appendix contains the original project description for this master project.

MSc Project: **Self-adaptation in a mobile manipulator to meet quality requirements**
student: Jasper Wijkhuizen

## Background

At AIRLab Delft we are addressing multiple challenges to robotize annoying tasks in retail stores. One of this is the re-stocking of products in the AGF (Fruit & Vegetables) department (see image below). We are looking for a solution to automate the re-stocking of products, e.g. bananas. This presents multiple robotic challenges, such as navigation through the store, perception of products and shelves, manipulation under uncertainty, and the integration of solutions for them into a reliable, efficient and safe autonomous control architecture.

Control architectures for mobile manipulators involve multiple components for different capabilities, e.g. localization, collision avoidance or navigation, and a deliberative layer responsible for coordinating and configuring them, to perform task, contingency and error handling. Current approaches are based on ad-hoc reasoning [1] that mixes all these concerns [2], resulting in solutions that are not reusable and maintainable. We propose to use appropriate architectural abstractions and symbolic reasoning to create reusable self-adaptation (diagnosis and reconfiguration) mechanisms that address contingency and error handling meeting the quality requirements (e.g. reliability, performance, safety) in mobile manipulators.

## Objective

The project will develop self- diagnosis and reconfiguration mechanisms [3] for the control of a mobile manipulator, based on a symbolic representation of its control architecture and non-functional requirements. This symbolic representation will be developed using the ISE&PPOOA methodology [4] for model-based systems engineering, and OWL ontologies and reasoners, for example with state-of-the-art frameworks such as KnowRob [5] and CRAM [6]. The self- diagnosis and reconfiguration mechanisms will be developed using the Metacontrol framework [3]. The concrete objectives of the project are:

1. Develop an ontological representation of the mobile manipulator's control architecture and non-functional requirements.
2. Create a ROS or ROS2 package that implements self- diagnosis and reconfiguration for a ROS or ROS2 system.
3. Analyze the performance of the self- diagnosis and reconfiguration mechanisms in the operation of the mobile manipulator.

## References (for the project discription only)

[1]. A. Ramaswamy, B. Monsuez, and A. Tapus. Model-driven self-adaptation of robotics software using probabilistic approach. In 2015 European Conference on Mobile Robots (ECMR), pages 1–6, Sep. 2015.
[2]. Bosch, micro-ROS Lifecycle and System Modes: https://micro-ros.github.io/docs/concepts/system_modes/
[3]. C. Hernandez, J. Bermejo-Alonso, and R. Sanz. A self-adaptation framework based on functional knowledge for augmented autonomy in robots. Integrated Computer-Aided Engineering, 25(2):157–172, 2018.
[4]. C. Hernandez and J. L. Fernandez-Sanchez. Model-based systems engineering to design collaborative robotics applications. In 2017 IEEE International Systems Engineering Symposium (ISSE), pages 1–6, Oct 2017.
[5]. M. Beetz, D. Beßler, A. Haidu, M. Pomarlan, A. Kaan Bozcuoglu, and G. Bartels. KnowRob 2.0 — A 2nd Generation Knowledge Processing Framework for Cognition-Enabled Robotic Agents. pages 512–519, 05 2018.
[6]. M. Beetz, L. Mosenlechner, and M. Tenorth. CRAM — A Cognitive Robot Abstract Machine for everyday manipulation in human environments. In 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 1012–1017, Oct 2010.

# B

# RoQME, an experience report

In this 2 page document, The RoQME tools are tested and tried to implement for the use with the ROS framework.

## B.1. What is RoQME

RoQME is a set of modelling and code generation tools, to model the non-functional properties and build a metrics provider. This metrics provider acts as a observer for the robotic system, and outputs metrics [0,1] for the non-functional properties. It can be used as a component in the SmartMDSD toolchain, which is an plugin for Eclipse. The SmartMDSD toolchain is a quite intuitive and visual way of system modelling. Although sometimes there are (too) many steps needed to get to something. For example, when you make a new RoQME model, you need to add the context variables by clicking through a very large list with spoilers etc to find the variables you need.

The developers provide a virtual machine image, to avoid all the installation troubles, and have a up and running machine immediately.

Since RoQME and ROS are not compatible (yet), I did not do a full test. For this reason, it can happen that there are some mistakes in my examples. The RoQME are working on a newer version, which is compatible with the latest SmartMDSD version. They can however not give an indication of how long this will take.

## B.2. A RoQME model

For this model, a very simple example is used where the nonfunctional property is modelled as the ratio of forward velocity and maximum velocity Non-functional properties are modelled using the following syntax: **performance**

```
property performance reference 1
```

Context variables are information from the sensors or states of the robot:

```
context v_x : number
```

Context variables can also be used to derive more complex context than just a number.

Observations are the relevant context patterns. These can be (part of) the equations which construct the QoS metrics. In the example of performance, an observation can be:

```
observation o1 : v_x < v_max undermines performance
```

For me, it is not clear yet what undermines exactly means. How much does it lower the performance? There is no documentation about this, and I did not found the code which does this(it happens inside the reasoner I think). It is possible to give a strength to the observation and to undermine or reinforce. The basic syntax for this is:

```
observation NAME : PATTERN EXPRESSION (reinforces | undermines)
PROPERTY [very low | low | high | very high]
```

Pattern expression can be defined using different functions: update(expr), eventWhen(expr), expr1 and | or expr2, period(variable), repeat(n), range(n1,n2), once expr, expr1 while(expr2), and some more

## B.3. QoS Metrics Provider

The QoS metric provider uses the roqme model to model the QoS metrics. A reasoner is included to determine the metrics at run-time.

While normally components in smartsoft can be compiled within Eclipse. For the QoSMetricsProvider, this needs to be done by hand. A script gen_makefile needs to be ran, and the package needs to be built for the QoSMetricsProvider and for the reasoner. The result is a single component which can be included in a bigger system in the SmartMDSD toolchain. Inputs and Outputs can simply be connected to other components. It is a visual and very intuitive method of building a system.

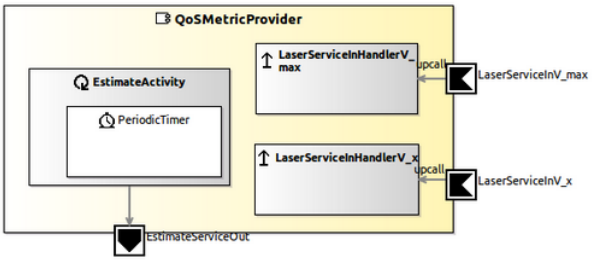



Figure B.1: Screenshot of my model in the SmartMDSD toolchain

## B.4. Communication with ROS

This is something which is already build-in in SmartMDSD(v3.8 and newer), and should work pretty easily. However, RoQME is at this moment only functional in SmartMDSD v3.7(while the newest version of SmartMDSD already v1.12 is). For this reason, I did not test this functionality yet.

A filename.rosinterfacespool file needs to be created, in which one or multiple rostopics can be subscribed to. The file contains for example:

```
RosInterfacesPool {
RosSubscriber _cmd_vel { topicName '\cmd_vel' type 'geometry_msgs.Twist' }
}
```

A component can then be edited, to add the ROS bridge. Instead of a regular input port, a ROS port can be selected.

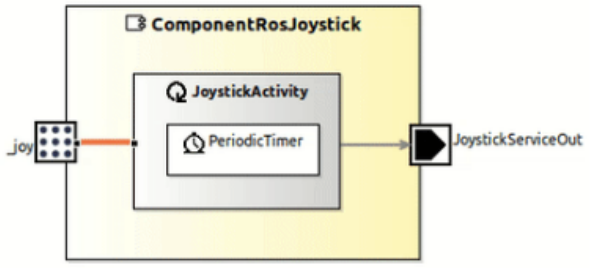



Figure B.2: Picture from the ROS bridge tutorial at wiki.servicerobotik-ulm.de

# C

# Experiment design

To be able to test and validate the developed concepts, experiments in a simulated world are designed. Different environments are simulated, based on the environment in an Albert Heijn retail store.

The first experiments are done in a simple corridor, which is explained in section C.2.

## C.1. Retail environment

The environment is based on the Albert Heijn XL Martinus Nijhoflaan in Delft. The floorplan of the Albert Heijn XL is used to build a realistic gazebo world for experiments in simulation. The floorplan of this retail store is used to design the experiment environments.

## C.2. Environment 1, the simple corridor

The first environment is a simple corridor. On both sides of the corridor are shelves full of products placed. The start position of the robot is at on end on the centerline of the corridor. The goal position is on the centerline at the other end. Obstacles will be spawned inside the corridor at random locations.

Different dependent variables are defined to design the corridor:

**Corridor width ($w$)** is the distance between the shelves.

**Corridor length ($l$)** is the length of the corridor.

**Obstacle clutterness ($C$)** is the average amount of obstacles per square meter that are randomly spawned. This number is used to determine the amount of obstacles to spawn: $n_{obstacles} = w * l * C$. They are however not perfectly uniform distributed spawned, but random.

**Obstacle minimal obstacle ($d_{min}$)** is used to avoid obstacles being too close together and possibly block the corridor.

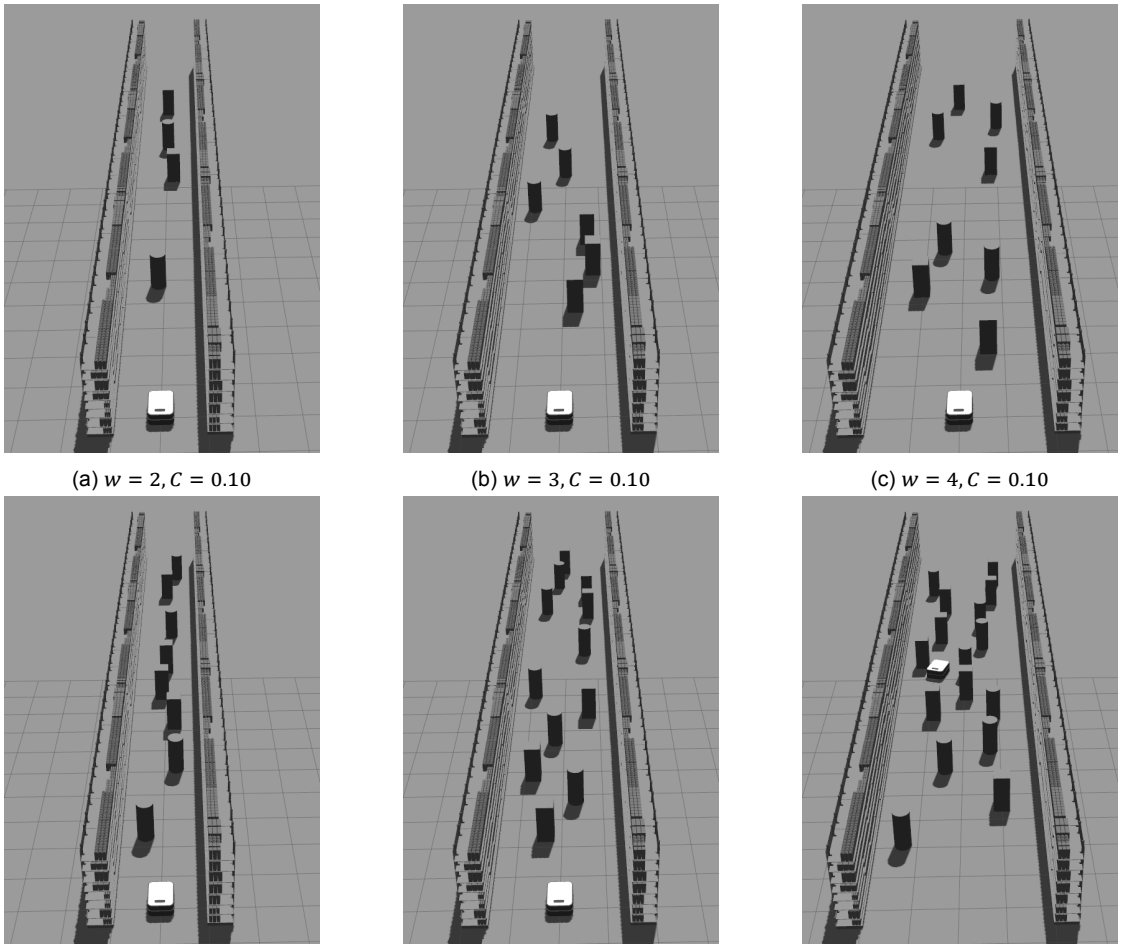
**Obstacle type/size** of the spawned obstacles can be chosen. To simplify the simulation, standard shapes such as cylinders and cubes are used.

### C.2.1. Characteristics from the retail store

From the AHXL Delft floorplan, the dimensions of the corridors are measured, and the smallest and largest possible passages (See Table 4.1). These dimensions are used for determining the values for the environment parameters. This is done to make sure that the experiment is done in a realistic environment.

Table C.1: Dimensions from the AHXL floorplan

| Type | Width [mm] |
|---|---|
| Normal corridor | 2300 |
| Wide corridor | 4200 |
| Narrow corridor | 1800 |
| Smallest passage | 1500 |
| Widest passage | 5000 |



(a) $w = 2, C = 0.10$ (b) $w = 3, C = 0.10$ (c) $w = 4, C = 0.10$

(d) $w = 2, C = 0.25$ (e) $w = 3, C = 0.25$ (f) $w = 4, C = 0.25$

Figure C.1: Examples of different simple corridor environments.

## C.2.2. Experiment 1, Quality and Environment metrics validation

For this experiment, two corridor widths are used to validate the metrics in the different corridors that can be encountered in the retail store.

A single run is done in every environment.

Table C.2: Environment design parameters for the metrics validations

| Environment id | Env1 | Env2 | Env3 | Env4 |
|---|---|---|---|---|
| Corridor width | 3 | 3 | 4 | 4 |
| Clutterness | 0.10 | 0.25 | 0.10 | 0.25 |
| Obstacle minimum distance | 1.4m | 1.4m | 1.4m | 1.4m |

### C.2.3. Experiment 2, collect training data for the training of the quality models

For these experiments, only two corridor widths and two obstacle clutternes amounts are used, resulting in 4 different environments. Only two corridor widths are usede because the results of the small corridor ($w = 2m$) are very similar to the other two widths, and therefore do not contribute to the quality the data.

Every environment is spawned five times, to generate enough data for the training of the models.

Table C.3: Environment design parameters for model training tests.

| Environment id | Env1 | Env2 | Env3 | Env4 |
|---|---|---|---|---|
| Corridor width | 3 | 3 | 4 | 4 |
| Clutterness | 0.25 | 0.10 | 0.25 | 0.10 |

# D

# Default local planner configurations

The parameters of the local planners are analysed and a selection is made of the parameters which are interesting for generation of the different configurations. For these parameters their dependency on the quality attributes is estimated. For the parameters that are selected for the generation of the configurations, different reasonable values are determined.

The default maximum forward velocity in ROS is 0.4. The maximum forward velocity for the boxer robot is higher (1.9). Since the robots in retail stores are used to improve productivity, it is decided to use a higher maximum velocity than the default in ROS.

To improve the reusability, and avoid fine tuning on a single robot model, most parameters are used with their default value. Only the parameters for the speeds and accelerations are changed into the robots limits, and some others to improve the behaviour at the higher speeds.

## D.1. Parameters

### D.1.1. Non default parameters for the DWA

vx_samples and vth_samples are non default to give the local planner enough options, since the speed is higher than default. sim_time heavely influences the performance of the local planner in this experiment. The larger this value, the further away the reference point on the global path is, which is used as a 'local goal' by the local planner. If this reference point is further away, the robot will cut the corners more. This effect can be seen by the length of the green line in figure D.1, where the green line is the local path, and the blue line is the global path. When using the default value for this parameter, the local planner will get stuck a lot in cluttered environment (probably because a higher maximum speed is used than default). Therefore, a second dwa configuration is used, with a lower value for sim_time.

(a) $sim\_time = 1.7$ (b) $sim\_time = 1.0$

Figure D.1: Visualization of the effect of the sim_time parameter.

### D.1.2. Non-default parameters for the TEB method

The default parameters for the TEB configurations are modified to reduce the computational expenses, to be able to run the simulation at normal speed. A different footprint is used, because this is much less computational expensive for this algorithm. The same holds for the parameter enable_homotopy_class_planning, this parameter is set to false since this parameter is too computational expensive.

Table D.1: The non-default parameters for the dwa local planner as used in this research.

| Parameter | Description | Default |
|---|---|---|
| max_vel_trans | The absolute value of the maximum translational velocity for the robot in m/s | 1 |
| min_vel_trans | The absolute value of the minimum translational velocity for the robot in m/s | 0 |
| max_vel_x | The maximum x velocity for the robot in m/s. | 1 |
| min_vel_x | The minimum x velocity for the robot in m/s, negative for backwards motion. | -1 |
| max_vel_y | The maximum y velocity for the robot in m/s | 0 |
| min_vel_y | The minimum y velocity for the robot in m/s | 0 |
| max_vel_theta | The absolute value of the maximum rotational velocity for the robot in rad/s | 1 |
| min_vel_theta | The absolute value of the minimum rotational velocity for the robot in rad/s | -1 |
| acc_lim_theta | The rotational acceleration limit of the robot in radians/sec$^2$ | 1 |
| acc_lim_trans | The rotational acceleration limit of the robot in radians/sec$^2$ | 0.5 |
| acc_lim_x | The x acceleration limit of the robot in meters/sec$^2$ | 0.5 |
| acc_lim_y | The y acceleration limit of the robot in meters/sec$^2$ | 0 |
| vx_samples | The number of samples to use when exploring the x velocity space | 5 |
| vy_samples | The number of samples to use when exploring the y velocity space | 0 |
| vth_samples | The number of samples to use when exploring the theta velocity space | 10 |

Table D.2: The non-default parameters for the teb local planner as used in this research.

| Parameter | Description | Default |
|---|---|---|
| max_vel_x | Maximum translational velocity of the robot in meters/sec | 1 |
| max_vel_x_backwards | Maximum absolute translational velocity of the robot while driving backwards in meters/sec. | 1 |
| max_vel_y | Maximum strafing velocity of the robot (should be zero for non-holonomic robots!) | 0 |
| max_vel_theta | Maximum angular velocity of the robot in radians/sec | 1 |
| acc_lim_theta | Maximum angular acceleration of the robot in radians/sec$^2$ | 1 |
| acc_lim_x | Maximum translational acceleration of the robot in meters/sec$^2$ | 1 |
| acc_lim_y | Maximum strafing acceleration of the robot | 0 |
| footprint_model type | Specify the robot footprint model type used for optimization. The type of the model significantly influences the required computation time. | ”line” |
| line_start | | [-0.1, 0] |
| line_end | | [0.4, 0] |
| enable_homotopy_class_planning | Activate parallel planning in distinctive topologies (requires much more CPU resources, since multiple trajectories are optimized at once) | false |
| max_number_classes | Specify the maximum number of distinctive trajectories taken into account (limits computational effort) | 2 |

# Bibliography


[1] Gregor Klančar et al. “Autonomous Guided Vehicles”. In: *Wheeled Mobile Robotics*. Elsevier, 2017, pp. 387–418. DOI: 10.1016/B978-0-12-804204-5.00007-X. URL: https://linkinghub.elsevier.com/retrieve/pii/B978012804204500007X.

[2] *Meet Amazon’s busiest employee – the Kiva robot*. URL: https://www.cnet.com/news/meet-amazons-busiest-employee-the-kiva-robot/.

[3] Lingqi Zeng and Gary M. Bone. “Mobile Robot Navigation for Moving Obstacles with Unpredictable Direction Changes, Including Humans”. In: *Advanced Robotics* 26.16 (Nov. 2012), pp. 1841–1862. ISSN: 0169-1864. DOI: 10.1080/01691864.2012.703166. URL: https://www.tandfonline.com/doi/full/10.1080/01691864.2012.703166.

[4] *Josie Pepper – the humanoid robot with artificial intelligence - Munich Airport*. URL: https://www.munich-airport.com/hi-i-m-josie-pepper-3613413.

[5] *Amazon heeft al 30.000 logistieke robots in dienst (+ video) | Flows*. URL: https://www.flows.be/nl/logistics/amazon-heeft-al-30000-logistieke-robots-dienst-video.

[6] Roland Siegwart and Illah R Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Vol. 2. 1. 2004. ISBN: 026219502X. DOI: 10.1016/s1474-6670(17)54619-4.

[7] N. Sariff and N. Buniyamin. “An overview of autonomous mobile robot path planning algorithms”. In: *SCOReD 2006 - Proceedings of 2006 4th Student Conference on Research and Development ”Towards Enhancing Research Excellence in the Region”*. 2006, pp. 183–188. ISBN: 1424405270. DOI: 10.1109/SCORED.2006.4339335.

[8] Thi Thoa Mac et al. “Heuristic approaches in robot path planning: A survey”. In: *Robotics and Autonomous Systems* (2016). ISSN: 09218890. DOI: 10.1016/j.robot.2016.08.001.

[9] Luca Gherardi and Nico Hochgeschwender. “RRA: Models and tools for robotics run-time adaptation”. In: *IEEE International Conference on Intelligent Robots and Systems*. Vol. 2015-Decem. Institute of Electrical and Electronics Engineers Inc., Dec. 2015, pp. 1777–1784. ISBN: 9781479999941. DOI: 10.1109/IROS.2015.7353608.

[10] C Hernandez Corbato. “Model-based Self-awareness Patterns for Autonomy”. PhD thesis. 2013.

[11] Davide Brugali. “Non-functional requirements in robotic systems: Challenges and state of the art”. In: *2019 IEEE International Conference on Real-Time Computing and Robotics, RCAR 2019*. Vol. 2019-Augus. Institute of Electrical and Electronics Engineers Inc., Aug. 2019, pp. 743–748. ISBN: 9781728137261. DOI: 10.1109/RCAR47638.2019.9044033.

[12] Juan Francisco Inglés-Romero et al. “Dealing with Run-Time Variability in Service Robotics: Towards a DSL for Non-Functional Properties”. In: (Mar. 2013). URL: http://arxiv.org/abs/1303.4296.

[13] Javier Berrocal, Jose Garcia-Alonso, and Juan Hernández. *RoQME: Dealing with Non-Functional Properties through Global Robot QoS Metrics*. Tech. rep. 2018. URL: www.uidee.com.

[14] Carlos Hernández, Julita Bermejo-Alonso, and Ricardo Sanz. “A self-adaptation framework based on functional knowledge for augmented autonomy in robots”. In: *Integrated Computer-Aided Engineering* 25.2 (2018), pp. 157–172. ISSN: 18758835. DOI: 10.3233/ICA-180565.

[15] Gamini Dissanayake et al. “A review of recent developments in Simultaneous Localization and Mapping”. In: *2011 6th International Conference on Industrial and Information Systems, ICIIS 2011 - Conference Proceedings*. 2011, pp. 477–482. ISBN: 9781457700354. DOI: 10.1109/ICIINFS.2011.6038117.

[16] Miroslav Kulich, Viktor Kozák, and Libor Přeučil. “Comparison of local planning algorithms for mobile robots”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9055. Springer Verlag, 2015, pp. 196–208. ISBN: 9783319223827. DOI: 10.1007/978-3-319-22383-4{_}15.

[17] P Raja and S Pugazhenthi. “Optimal path planning of mobile robots: A review”. In: *International Journal of Physical Sciences* 7.9 (2012), pp. 1314–1320. DOI: 10.5897/IJPS11.1745. URL: http://www.academicjournals.org/IJPS.

[18] M. G. Mohanan and Ambuja Salgoankar. *A survey of robotic motion planning in dynamic environments*. 2018. DOI: 10.1016/j.robot.2017.10.011.

[19] Aykut Özdemir and Volkan Sezer. “Follow the gap with dynamic window approach”. In: *International Journal of Semantic Computing* 12.1 (2018), pp. 43–57. ISSN: 17937108. DOI: 10.1142/S1793351X18400032.

[20] J.F. Sanchez. “Implementation and comparison in local planners for Ackermann vehicles”. In: (2018).

[21] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. “The dynamic window approach to collision avoidance”. In: *IEEE Robotics and Automation Magazine* 4.1 (Mar. 1997), pp. 23–33. ISSN: 10709932. DOI: 10.1109/100.580977.

[22] Christoph Rösmann et al. “Trajectory modification considering dynamic constraints of autonomous robots”. In: *Robotik 2012* (2012), pp. 74–79.

[23] S. Quinlan and O. Khatib. “Elastic bands: connecting path planning and control”. In: *[1993] Proceedings IEEE International Conference on Robotics and Automation*. IEEE Comput. Soc. Press, pp. 802–807. ISBN: 0-8186-3450-2. DOI: 10.1109/ROBOT.1993.291936. URL: http://ieeexplore.ieee.org/document/291936/.

[24] Paolo Arcaini, Elvinia Riccobene, and Patrizia Scandurra. “Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation”. In: *Proceedings - 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015*. Institute of Electrical and Electronics Engineers Inc., Aug. 2015, pp. 13–23. ISBN: 9781479919345. DOI: 10.1109/SEAMS.2015.10.

[25] Christian Krupitzer et al. “A survey on engineering approaches for self-adaptive systems”. In: *Pervasive and Mobile Computing* 17 (2015), pp. 184–206. ISSN: 1574-1192. DOI: https://doi.org/10.1016/j.pmcj.2014.09.009. URL: http://www.sciencedirect.com/science/article/pii/S157411921400162X.

[26] Matthias Rohr et al. “A classification scheme for self-adaptation research”. In: *Proceedings of the International Conference on Self-Organization and Autonomous Systems In Computing and Communications (SOAS’2006)* (2006), p. 5. URL: http://se.informatik.uni-oldenburg.de:30000/273/.

[27] Davide Brugali et al. “Model-based development of QoS-aware reconfigurable autonomous robotic systems”. In: *Proceedings - 2nd IEEE International Conference on Robotic Computing, IRC 2018*. Vol. 2018-Janua. Institute of Electrical and Electronics Engineers Inc., Apr. 2018, pp. 129–136. ISBN: 9781538646519. DOI: 10.1109/IRC.2018.00027.

[28] Didac Gil De La Iglesia and Danny Weyns. “MAPE-K formal templates to rigorously design behaviors for self-adaptive systems”. In: *ACM Transactions on Autonomous and Adaptive Systems* 10.3 (Aug. 2015), pp. 1–31. ISSN: 15564703. DOI: 10.1145/2724719. URL: http://dl.acm.org/citation.cfm?doid=2819320.2724719.

[29] John C. Georgas and Richard N. Taylor. “Towards a knowledge-based approach to architectural adaptation management”. In: *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. 2004, pp. 59–63. ISBN: 1581139896. DOI: 10.1145/1075405.1075417.

[30] Mogens Blanke et al. *Diagnosis and Fault-Tolerant Control*. 2006, p. 685. ISBN: 9783540356523.

[31] Betty H.C. Cheng et al. “Software engineering for self-adaptive systems: A research roadmap”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 5525 LNCS. 2009, pp. 1–26. ISBN: 3642021603. DOI: 10.1007/978-3-642-02161-9{_}1.

[32] Alex Lotz et al. “Managing run-time variability in robotics software by modeling functional and non-functional behavior”. In: *Lecture Notes in Business Information Processing*. Vol. 147 LNBIP. Springer Verlag, 2013, pp. 441–455. ISBN: 9783642384837. DOI: 10.1007/978-3-642-38484-4{_}31.

[33] N. D. Muñoz, J. A. Valencia, and N. Londoño. “Evaluation of navigation of an autonomous mobile robot”. In: *Performance Metrics for Intelligent Systems (PerMIS) Workshop* 1 (2007), pp. 15–21. DOI: 10.1145/1660877.1660878.

[34] Aaron Steinfeld et al. “Common metrics for human-robot interaction”. In: *HRI 2006: Proceedings of the 2006 ACM Conference on Human-Robot Interaction*. Vol. 2006. Association for Computing Machinery, 2006, pp. 33–40. ISBN: 1595932941. DOI: 10.1145/1121241.1121249.

[35] Woojin Chung et al. “Safe navigation of a mobile robot considering visibility of environment”. In: *IEEE Transactions on Industrial Electronics* 56.10 (2009), pp. 3941–3950. ISSN: 02780046. DOI: 10.1109/TIE.2009.2025293.

[36] Carlos Hernandez Corbato et al. “MROS: Runtime Adaptation For Robot Control Architectures”. In: (Oct. 2020). URL: http://arxiv.org/abs/2010.09145.

[37] Cristina Vicente-Chicote et al. “A component-based and model-driven approach to deal with non-functional properties through global QoS metrics”. In: *CEUR Workshop Proceedings*. Vol. 2245. CEUR-WS, 2018, pp. 40–45.

[38] David T WestWick and Robert E Kearney. *Identification of nonlinear physiological systems*. Vol 7. John Wiley & Sons, 2003.

[39] *Difference Between Supervised, Unsupervised, & Reinforcement Learning | NVIDIA Blog*. URL: https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/.

[40] Fabian Pedregosa FABIANPEDREGOSA et al. *Scikit-learn: Machine Learning in Python Gaël Varoquaux Bertrand Thirion Vincent Dubourg Alexandre Passos PEDREGOSA, VAROQUAUX, GRAMFORT ET AL. Matthieu Perrot*. Tech. rep. 2011, pp. 2825–2830. URL: http://scikit-learn.sourceforge.net..

[41] *7 Steps to Machine Learning: How to Prepare for an Automated Future | by Dr Mark van Rijmenam | DataSeries | Medium*. URL: https://medium.com/dataseries/7-steps-to-machine-learning-how-to-prepare-for-an-automated-future-78c7918cb35d.

[42] Giuseppe Bonaccorso. “Machine learning algorithms”. In: *Packt Publishing Ltd* (2017).

[43] Jeffrey O. Kephart and David M. Chess. “The vision of autonomic computing”. In: *Computer* 36.1 (Jan. 2003). ISSN: 00189162. DOI: 10.1109/MC.2003.1160055.

[44] Evren Sirin et al. “Pellet: A practical OWL-DL reasoner”. In: *Web Semantics* 5.2 (June 2007), pp. 51–53. ISSN: 15708268. DOI: 10.1016/j.websem.2007.03.004.

[45] *Lifecycle and System Modes | micro-ROS*. URL: https://micro-ros.github.io/docs/concepts/client_library/system_modes/.