Elias Treis (462941)

Development of a Benchmarking Platform for Autonomous Navigation Approaches

Master Thesis
to achieve the university degree of
Master of Science

submitted to

Technische Universität Berlin

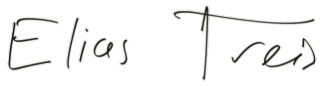
Supervisor: M.Sc. Linh Kästner
First Examiner: Prof. Dr.-Ing. Jens Lambrecht
Second Examiner: Prof. Dr.-Ing. Sebastian Möller

Industry Grade Networks and Clouds

April 01, 2022

Affidavit

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.
Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.



Berlin: March 30, 2022

Abstract

This work develops a benchmarking platform for comparing robot navigation approaches. It allows the creation of custom scenarios, running of the scenarios in 2D and 3D environments, and evaluating robot performance in the simulation. This platform is unique:

- Firstly, in its size, as it contains the largest collection *Gazebo* worlds, robots, and local planners,
- Secondly, in its capabilities as it supports different modes of usage, and the training of custom rl-navigation agents.

With over 5400 simulation runs in diverse environments this work addresses in a second part two research questions (utilizing the benchmarking platform):

- Firstly, how simulating robots in 2D-*Flatland* environments impacts robot performance compared to more precise and computationally expensive 3D-*Gazebo* environments.
- Secondly, whether the usage of an emergency brake system with mobile robots has a positive impact on navigation safety.

Kurzfassung

In dieser Arbeit wird eine Benchmarking-Plattform für den Vergleich und Benchmarking von Roboter- Navigations Ansätzen entwickelt. Diese Plattform ermöglicht die Erstellung von benutzerdefinierten Szenarien, die Ausführung der Szenarien in 2D- und 3D-Simulationsumgebungen und die Auswertung der Roboterleistung in der Simulation. Diese Plattform zeichnet sich durch mehrere Alleinstellungsmerkmale aus:

- Erstens durch ihren Umfang, da sie eine große Sammlung von Gazebo-Welten und simulierten Robotern, sowie einer Vielzahl von Motion-Planning ansätzen beinhaltet,
- Zweitens durch ihre unterschiedlichen Nutzungsmöglichkeiten sowie die Möglichkeit, eigene RL-Agents zu trainieren.

Auf die Platform zurückgreifend, behandelt diese Arbeit in einem zweiten Teil zwei Forschungsfragen:

- Erstens, wie sich das Simulieren von Robotern in 2D-*Flatland* Umgebungen im Vergleich zu 3D-*Gazebo* Umgebungen im Bezug auf die Leistungsdaten unterscheidet.
- Zweitens, ob sich das Verwenden eines sicherheitsstop-Mechanismus positiv auf die Navigationssicherheit auswirkt.

Contents

List of Abbreviations

AGV	Autonomous Ground Vehicle
AMCL	Adaptive Monte Carlo Localization
AI	Artificial Intelligence
cmd_vel	Command Velocity
DRL	Deep Reinforcement Learning
GPS	Global Positioning System
DWA	Dynamic Window Approach
ROS	Robot Operating System
RL	Reinforcement Learning
RGB	Red, Green, Blue
TEB	Timed Elastic Band
TF	Transform

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The field of robot navigation has been the subject of intense research in recent years. This research however focuses primarily on developing new and improved navigation concepts for specific use cases. For comparing recent progress, there have been several small-scale efforts to create benchmarking platforms (primarily to compare specific approaches to other commonly used approaches of this kind). There is therefore a need for a more comprehensive benchmarking platform.

- existinge components and totols that are indpendent
  This need can be met in a unique fashion by the *ignc* chair, as past research projects have developed all components necessary for this task. This includes the in size and functionality unmatched 2D and 3D *arena* platforms with a large selection of different planning approaches and robot types. As well as the custom scenario-development with *arena-tools* and simulation-evaluation with *arena-evaluation*.

This work unites these independent repositories into an interconnected benchmarking platform enabling a coherent workflow: from creating custom scenarios and worlds with *arena-tools*, simulating the scenarios in 2D and 3D environments to evaluating the robot performance via different measures with *arena-evaluation*.

This platform is utilized in a second step to answer two research questions. The first research question examines how the usage of a 2D platform (with a higher degree of abstraction) impacts the accuracy of navigation-approach evaluation compared to a more sophisticated 3D simulation. This information is used to explain when using a more computationally intensive 3D simulation is justified by higher result accuracy.

The second research question examines whether the usage of an emergency-break system has a positive impact on collision avoidance and to which degree its performance is influenced by environmental factors.

1.2 Objective

This work can be broken down into three major objectives: the creation of the benchmarking platform, the answering of the research questions, and the creation of a large data set.

For the creation of the benchmarking platform, this work aims to create well-documented interfaces between *arena-tools* and the simulation platforms *arena-rosnav* and *arena-rosnav-3D*. This will allow users to use *arena-tools* to create custom 2D worlds and scenarios. To simulate planners and robots in both 2D and 3D environments, planning approaches and robots that are only available in the 2D repository are to be made available in the 3D repository and vice versa. In addition, the implementation of new components such as planners and robots will be documented. The *arena-evaluation* repository is to be expanded to include simulation-specific factors such as simulation speed. Performance measures are to be expanded to more accurately document robot performance.

To answer the research questions, the above-mentioned benchmarking platform will be used to create scenarios and worlds at different difficulty levels. Data will be collected with different robot models in both the 2D and 3D simulations. To answer the second research question, the emergency braking system must be enabled on all robots and common issues of interaction with the navigation stack must be addressed before collecting data in the predictively more accurate 3D simulation. The collected data for both research questions will be analyzed by examining the effects of the parameters in question on robot performance.

To create the large data set needed to build a model that can predict thresholds for robot performance without running a simulation, the *random* task mode is to be extended so that it can be used for *arena-evaluation*. In addition, the 3D repository's random world generator is to be extended to generate random worlds in a more controllable manner. In addition, the *arena-evaluation* repository will be extended to record not only active simulation data such as laser scan data but also environmental data such as the number and speed of dynamic obstacles or the occupancy map of the world generated by the random world generator.

2 Related Works

This section summarizes current research in robot simulation, benchmarking metrics, and benchmarking platforms.

A. Robot Simulation

Simulating mobile robots in virtual environments under realistic conditions has become an important practice in the field of robot navigation. There are several different simulators that perform very different functions. The *Gazebo* simulator[1] is designed to simulate dynamic 3D multi-robot environments in realistic virtual environments. It features realistic physics engines, sensor simulations, and a graphical interface[13]. *Gazebo* is the most widely used 3D robot simulator (closely followed by *V-Rep*[2])[31], mainly because of its well-established ros user interface[26][4][21].

For certain use cases, such as training reinforcement agents, simpler platforms (with a higher level of abstraction) are used. Here, the *Flatland* simulator[3] has been suggested by previous research as it is lightweight, easy to use, and highly customizable [3][8]. A comprehensive comparison between different 2D simulators, or a comparison between the *Flatland* simulator and *Gazebo* has so far not been conducted.

B. Benchmarking-Metrics

For a unified evaluation Jian Wen et al. [36] proposed several metrics to comprehensively compare the planner performance, these include safety metrics, efficiency metrics, and smoothing metrics. Safety metrics are used to quantify the proximity of robots and obstacles in different planning approaches. Efficiency metrics describe the time required to achieve a given goal. Smoothness metrics evaluate the smoothness of the navigation path. Additionally, these metrics are demonstrated by evaluating two popular local planners in environments with different levels of difficulty.

Daniel Perille et al. [25] provides a set of metrics and environments to evaluate the motion performance of wheeled robots in (static) obstacle-occupied, planar environments. The metrics focus primarily on using the robot's laser scan data to predict path difficulty. A created dataset is publicly available and can be used to benchmark other navigation approaches. A series of (non-realistic) *Gazebo*

[1] http://gazebosim.org/

[2] https://www.coppeliarobotics.com/

[3] https://flatland-simulator.readthedocs.io/en/latest/

worlds were created at various levels of difficulty.
In their foundational work, Alexandre Lampe and Raja Chatila [16] present a set of theoretical measures for predicting the performance of mobile robots in simulated environments. These include the speed, distance, time, and success rate of the robots in reaching a given goal. In addition, they propose measures to quantify the complexity of the simulated environment (global, i.e., the complexity of the overall map, and second, local, the complexity of the map near the robot).
I. Rano and J. Mingue [28] propose a set of methods to evaluate the performance of mobile robots under different environmental conditions. These metrics primarily quantify robot performance in terms of robustness, optimality, safety, etc., but also the difficulty of a given environment. However, these theoretical metrics are not implemented but are merely proposed metrics to quantify robot performance in specific environments.

C. Benchmarking-Platforms

Regarding the benchmarking of motion planning approaches, Jian Wen et al. [9] created the benchmarking platform "Bench-MR" to compare the performance of four-wheeled, non-holonomic mobile robots, and provide a set of navigation scenarios and implementations of performance metrics, as well as a set of visualization techniques to evaluate robot performance. However, the use of integrated platforms and metrics is strongly tied to the proposed benchmarking scenarios.
Mark Moll et al. [18] proposed several benchmarks and implemented them within the OMPL package[4]. These metrics focus more generally on motion planning problems and, with the integrated benchmarking platform, provide a comprehensive overview of the performance of the implemented algorithms. The included algorithms focus mainly on moving systems such as robotic arms and comparison of steering algorithms.
Servant [33] has introduced a benchmarking framework for path planning algorithms specifically for robotic systems without kinematic constraints. The "Moving AI" path-finding benchmark provides many navigation scenarios in different grid-based environments, such as city grids, which are publicly available and can be used for further research. In addition to the environments, a set of performance metrics is introduced to allow the benchmarking of algorithms in standardized grid-based (non-real-world) environments.

[4]https://ompl.kavrakilab.org/index.html

3 Fundamentals

This section covers some theoretical fundamentals relevant to the work of this thesis. In particular, the *Gazebo* and *Flatland* simulation platforms, the ROS navigation stack, and the motion planning algorithms are briefly introduced. In addition, the types of mobile robot bases and their differences are explained, and a general introduction to robot safety and environment complexity is given.

3.1 Gazebo Simulator

Gazebo[1] is the most popular 3D robot simulation platform due to its extensive documentation[7] and community, a wide range of features, and well-established Ros interface. *Gazebo* is a standalone 3D simulator designed to simulate robots in environments with realistic physics as well as collaborative scenarios, such as multi-robot scenarios or human-robot scenarios, etc.

Figure 3.1 provides a rough draft of the conceptual structure of the *Gazebo* simulator. In general, it consists of several elements:

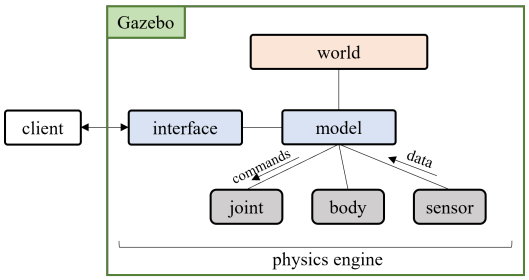


Figure 3.1: The conceptual design of the *Gazebo* simulator

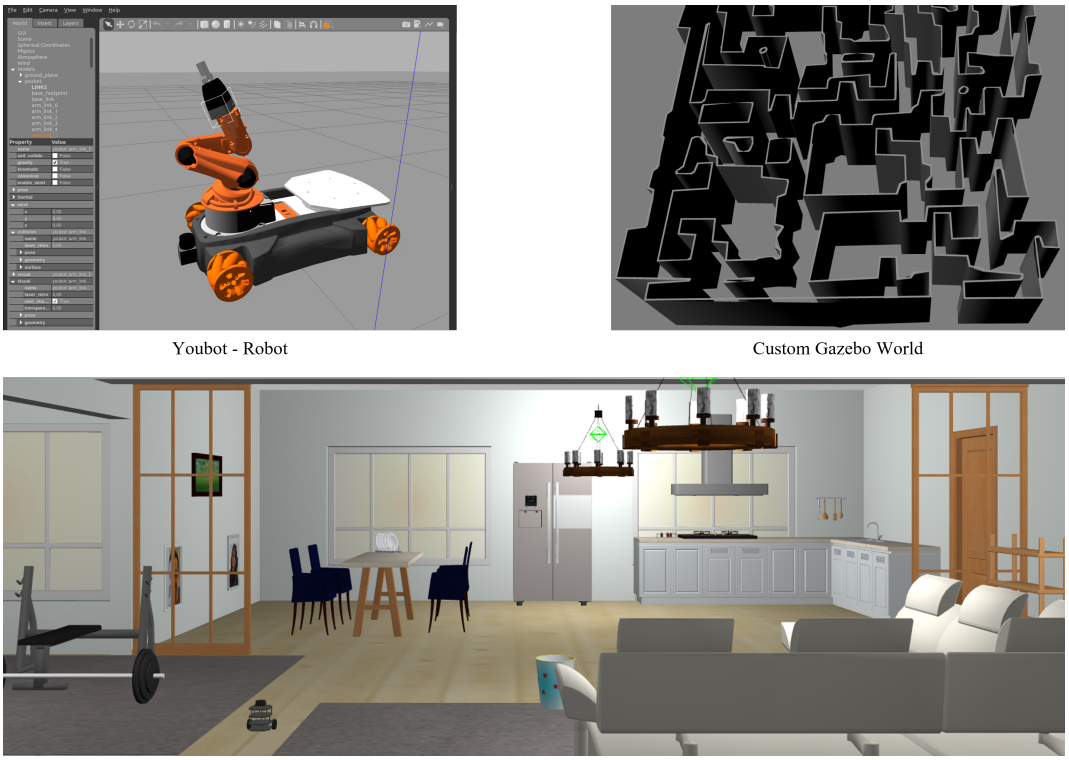
Gazebo worlds, which are objects are written in *sdf*[30] format, that describe a scene (i.e. static and dynamic obstacles), static obstacles, as well as robots, are part of the *model* class (as opposed to the *world* class) and can even be added/spawned during a running simulation, while world objects have to be defined before starting the simulation. For virtual worlds, one can either use pre-built worlds provided by the Gazebo community (by far the largest collection can be found in the *arena-rosnav-3D*[2] repository) or create one's custom worlds by e.g. creating mesh objects (e.g. using blender[2]). Robots and other models can either be loaded from the extensive collection of the *Gazebo* community or created by the user (see Figure 3.2 for example of *Gazebo*-objects). Simulated robots function by connecting static elements such as the robot body

[1] http://gazebosim.org/
[2] https://github.com/ignc-research/arena-rosnav-3D

and wheels with links and joints. By defining the robot joints, the movement between the parts can be defined (such as possible directions of movement (trans-x, trans-y ...), maximum acceleration ...). It is possible to add different sensors to the robot models, such as lidar or odom sensors, which publish information about the model.

Newer versions of *Gazebo* provide a class called *actors* in addition to the world and *model* objects. These are animated mesh objects such as walking humans that enable the simulation of dynamic obstacles and thus more realistic human-robot collaboration scenarios.

It is possible to interact with *Gazebo* in different ways. *Gazebo* consists of two components, firstly the *gz server*, which is used to run the simulation, and secondly the *gz client*, which provides a graphical interface to the simulation. *Gazebo* communicates similarly to ros using topics and services. By using the *gazebo_ros_pkgs* packages, it is possible to additionally interact with *Gazebo* via ros messages and services. Another way to interact with *Gazebo* is offered via plugins that add additional functionalities to the simulator, such as laser beam visualization.



Youbot - Robot

Custom Gazebo World

Figure 3.2: The Gazebo youbot model and an example of custom and pre-build Gazebo worlds

The use of *Gazebo*, especially for the development testing of robot concepts, has

several advantages. Firstly alterations in the simulation can usually be done with much less effort, secondly, the *Gazebo* logging functionality allows for easy data collection. It is also possible to increase the simulation speed to test in faster than real-time. The simulation time is dependent on several factors, firstly the available computational power and secondly on the complexity of the scenario. Meaning, the number and level of detail of the models included (surfaces, lines, etc.), the number of sensors and plug-ins, the number of interactions between simulation components (e.g., the number of contact points)[24].

3.2 Flatland Simulator

The *Flatland* simulator[3] is a lightweight 2D robot simulator (see some graphical examples in Figure 3.3). It integrates directly with ROS and uses the *Box2D* physics engine. Because of its simple customizability, it is a popular simulator for training rl-agents in ROS[3][8]. The simulator is divided into four ROS packages: *flatland-server, flatland-plugins, flatland-viz,* and *flatland-messages.*

The *Flatland-server* contains the core functionality of Flatland, including simulation environment, models, plugin interfaces, and ROS services. Unlike *Gazebo*, the entire *Flatland* server runs on a single ROS node in a single thread (allowing easier parallelization). The functionalities are divided into several main areas. The first is the 'simulation manager', which manages the state of the simulation (e.g., simulation speed). The simulated world includes all models and layers. The models, i.e. predefined objects like obstacles or robots. And layers, which include the static environment in the simulation (e.g. walls). Models like robot models are defined in a separate file and include the definition of basic aspects like size or color.

The *Flatland-plugin* package contains a number of helpful plugins, such as the *diff-drive* plugin for mobile robots, that add functionality to the *Flatland* 2D simulation.

The *Flatland-visualization* package enables the visualization of the simulation process with RVIZ. Additionally, the user can interact with the simulation, e.g. by publishing navigation goals with RVIZ GUI.

The *Flatland-msgs* package provides a collection of ROS messages and services to interact with the *Flatland* simulator.

It is important to mention that the *Flatland* simulator has been adapted

[3]https://flatland-simulator.readthedocs.io/en/latest/index.html

to the purposes of the Arena suite, e.g. by integration of *Pedsim*[4] (used for realistic dynamic obstacle simulation).

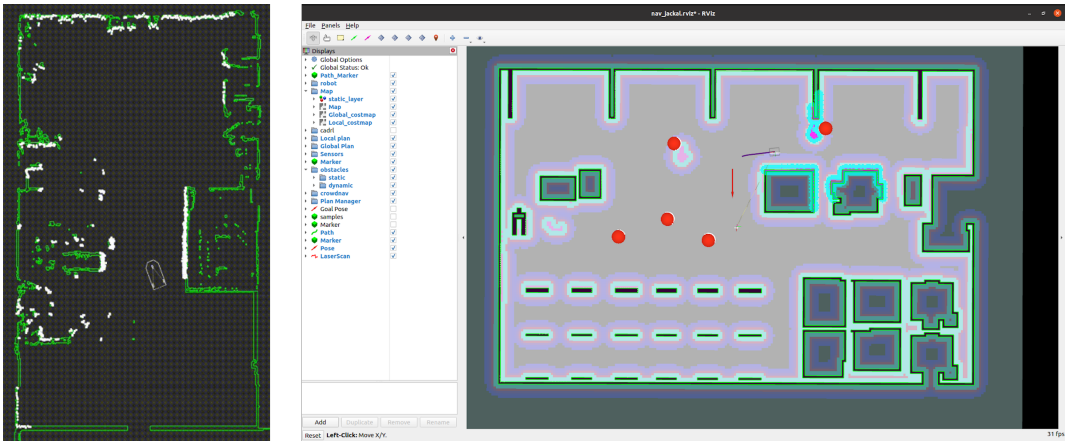


Figure 3.3: An examples of Flatland-simulation runs, visualized with RVIZ (modified from [1])

3.3 RVIZ

Unlike *Gazebo* and *Flatland*, *RVIZ* is not a standalone simulator, but a ROS visualization tool[11]. It can be used to visualize ROS topics and publish service requests and messages, providing a useful overview of the simulation process and data. Compared to other arena components, it is relatively lightweight and has therefore become an integral part of the arena suite, especially for monitoring the simulation process. To encode the relevant ROS data, the user has to define an *RVIZ*-settings file where the relevant messages, message types, visualization technique, etc. are specified.

3.4 ROS Navigation Stack

The ROS *navigation stack* [20] is an integral part of robot navigation with local planning algorithms. It takes information from odometry and sensor streams such as *odometry lidar* and outputs velocity commands that are sent to a mobile base. Figure 3.4 shows a rough conceptual design of the navigation stack, which consists of various components called nodes that exchange information by sending data over the network using specific message topics and formats. In the following sections, we will briefly introduce the components and notes shown in the Figure.

[4]https://github.com/srl-freiburg/pedsim_ros

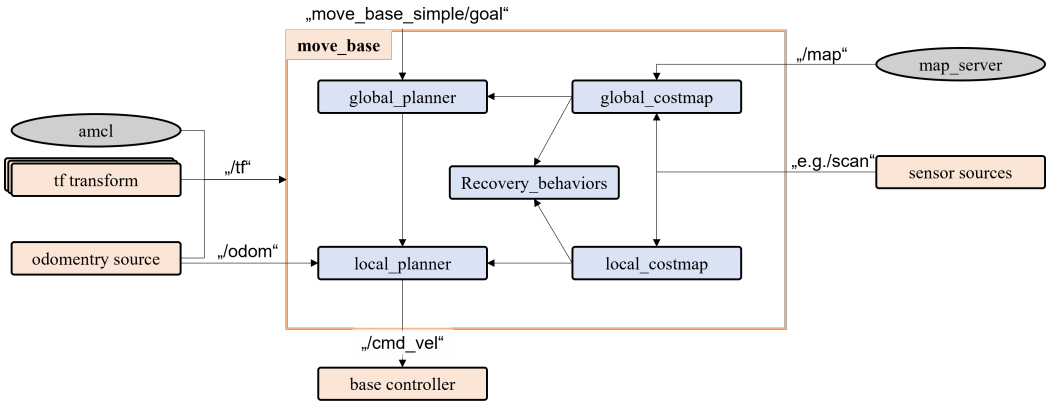


Figure 3.4: The conceptual design of the move base package

3.4.1 The Map-Server Package

The *map-server* package[5] is a ROS node that offers map data as a ROS service and provides utilities for creating such maps. In a typical ROS simulation pipeline, the first step is to create a map of the simulated environment by moving the robot around in the environment and using the incoming sensor data to create a map representation of the environment (see an example map in Figure 3.5). These maps consist of map metadata as well as a constructed image of the scene. The image consists of two main values, a white pixel describing an empty space and a black pixel describing an occupied space. Any other value, usually gray, represents an unknown (not yet mapped) area. The map server then provides the ability to publish this information to other nodes in a format called *OccupancyGrid*. In this way, we obtain a common framework, often called a map frame, in which we can describe the position of entities in the environment.

[5] http://wiki.ros.org/map_server

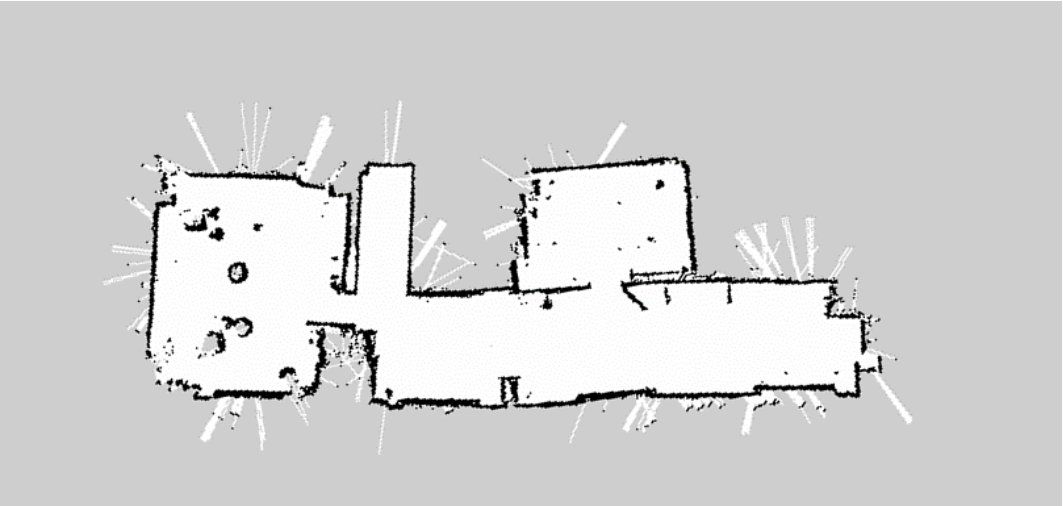


Figure 3.5: Realistic 2D Occupancy Grid Map of the ignc-lab

3.4.2 The Transform Configuration

The *tf* package[6] allows the user to track multiple coordinate frames over time. When working with different entities such as robots and maps, the data has different origins. For laser data, the origin point is at the position of the laser scanner, while odometry data (information about the position of the robot) has the origin of the odometry sensor (base link) and so on. This means that the information extracted from the sensor must be transformed for other robot links to correctly detect and avoid obstacles. The *tf* package reads the configuration of a robot and manages the relationship between the coordinate frames of the different links of the robot and between the robot and the world. In this way, we can transform the obtained data to know the position of the obstacles in the desired coordinate frames.

3.4.3 The Sensor Information

Robots can be equipped with various sensors. Very common is the use of laser scanners, as they are needed for many navigation approaches. They emit laser beams at a predefined interval that measures the proximity of objects within the laser range. Important parameters defining the laser are, therefore:

- The laser *range*: the radius of the scan circle.
- The number of *laser beams* is the number of samples taken in a scan circle
- The *angle* [”min”, ”max”]: describes the start and endpoint of the scan circle
- The *origin*: defines the position of the laser scanner on the robot

[6]http://wiki.ros.org/tf

- The *step size*: which describes the degree to which a laser scan is evaluated, can be calculated as follows:

$$\frac{|angle\_max - angle\_min|}{num.\ of\ laser\ beams}$$

A graphical representation of these parameters can be found in Figure 3.6.

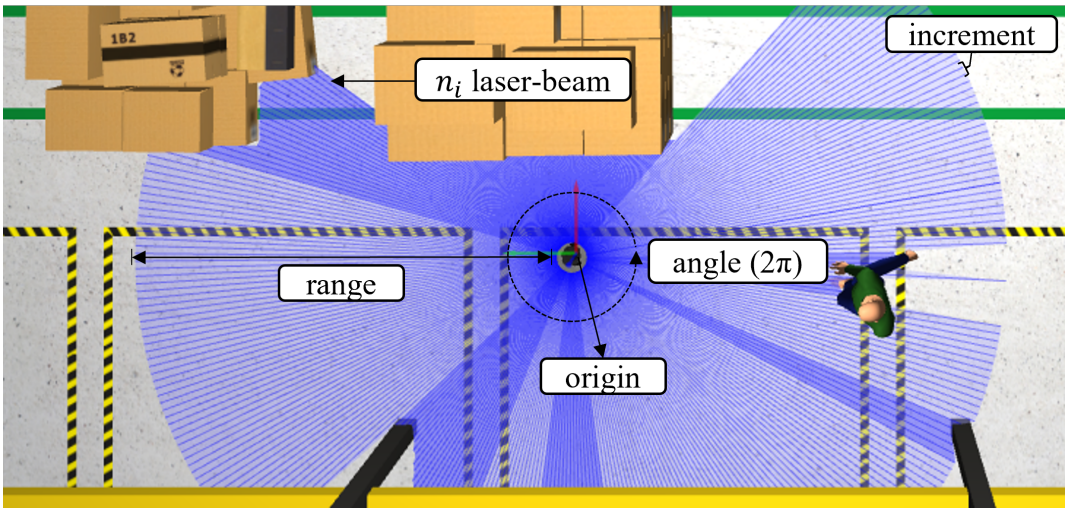


Figure 3.6: Visual representation of important laser sensor parameters

Other commonly used sensors include *RGB-D* cameras, which produce a constant input stream of images, or *GPS* sensors, which provide more accurate localization of the robot (however, due to the idealized conditions in robot simulations, these are usually not required).

3.4.4 The Odometry Information

Once a robot starts its autonomous navigation process from a given location, we want to track its position and orientation along the path it takes. This data is called *odometry* data and is usually based on a combination of inputs from the robot's measurement devices. The wheels have a special encoder that counts how many times the wheel has turned left or right. Opposed to this is an inertial measurement unit that uses a gyroscope to measure the orientation of the robot. With this information, we can estimate the change in the robot's position as well as its speed.

3.4.5 The Amcl package / Robot Localization

Localizing a robot in a given map is another important task required for robot navigation. While we can use odometry information to estimate the position of

the robot relative to a starting point, we are also interested in estimating the current position of the robot relative to a common frame, usually the map of the environment. There are several approaches to this, with adaptive Monte Carlo localization (*amcl*)[7] being the most widely used. This involves comparing the local environment detected by the lidar sensor with the environment published by the map server [5]. This type of localization is needed mainly when the *odometry* data is not necessarily reliable. However, in the case of the simulated worlds in *Flatland* and *Gazebo*, this is (mostly) not the case. Therefore, it is sufficient to publish the position of the robot relative to the map image once. (It is also possible to simulate inaccurate *odometry* information as it occurs in real-world scenarios. However, this is not yet included in the arena suite).

3.4.6 The Move-Base Package

The *move-base* package[8] attempts to move a mobile robot to an navigation goal. To accomplish this task, it consists of various components that are needed to perform the navigation task [14]:

- The *local* and *global* 2D cost-maps are the topics that contain the information representing the projection of the obstacles on a 2D plane (the ground), as well as a safety inflation radius, an area around the obstacles that guarantees that the robot will not collide with any object, regardless of its orientation. These projections have a cost associated with them, and the robot's goal is to reach the navigation goal by creating a path with the lowest possible cost. While the global cost map represents the entire environment (or a large portion of it), the local cost-map is generally a scrolling window that moves in the global cost map relative to the robot's current position
- The *global* planner uses the current position of the robot and the navigation goal and predicts a trajectory with the lower cost for the *global* cost-map. The *local* planner uses the *local cost map* to predict the immediate robot trajectory. Using the local cost map has several advantages. Since it includes not only the information from the static map but also from sensors such as lidar, it, therefore, includes dynamically changing elements such as dynamic obstacles or other robots. The local planner is thus responsible for creating the actual trajectory, using the global plan as input and deviating from it as needed based on changes in the local cost map. See section 3.6 for an introduction to different local planning approaches.
- The *recovery* behavior is a set of actions that are activated when the system is unable to create a valid navigation path. It is a multi-stage process with increasing aggressiveness if the previous stages were unsuccessful. It includes resetting the cost map and finally aborting the navigation [27].

[7] http://wiki.ros.org/amcl
[8] http://wiki.ros.org/move_base

See Figure 3.7 for a visualization of these concepts.

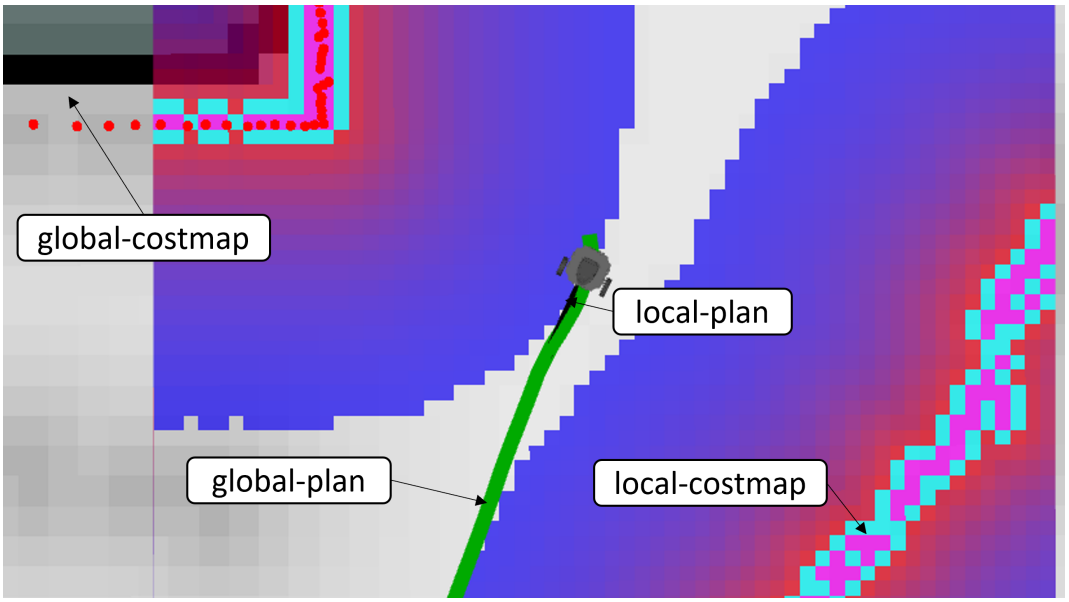


Figure 3.7: Local, global plan and costmap on the example of the burger-robot

3.4.7 The Base-Controller Package

The *base-controller* is an element that subscribes to the "*cmd_vel*" (command-velocity) topic and converts this into motor commands for the robot. In the case of the arena suite, this task is taken over by the respective simulator and depends on the robot and platform used.

3.5 Robot Types

The *arena-rosnav-3D* and *arena-rosnav* benchmarking repositories contain a number of different *agv* robot types. These differ in several factors. All are equipped with at least a lidar sensor. However, the specific sensor configurations differ between models. Also, different navigation approaches are used. First, *differential drive* robots, second, *holonomic* robots, and third, *car-like* robots. The differences between these robots are explained in the following sections. To distinguish the robot models consistently, the robot coordinate frames must be introduced.

The degrees of freedom regarding motion can be described as shown in Figure 3.8 with:

- The forward motion on the x-axis
- The sideways motion on the y-axis
- The vertical motion on the z-axis
- The rotational motion *theta* (θ)

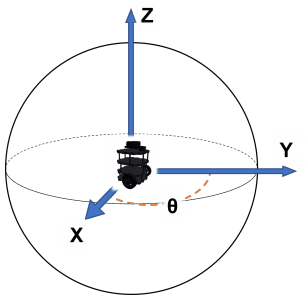


Figure 3.8: The robot coordinate frames

3.5.1 Differential Drive Robots

Differential drive robots are robots like the t*urtlebot3-burger/waffle.pi*[9] that have two wheels that allow individual motion/rotation control. The *burger* robot shown in Figure 3.8 has two front wheels with individual motors and a third support wheel that can move in all directions but does not have an individual motor. These robots can therefore only move in the x and *theta* directions.

3.5.2 Car like Robots

Car-like robots such as the *Jackal* robot[10] have four wheels and function like the differential drive robots (with two individually steered wheels). The special design of the robot with four wheels does not allow the robot to drive all the paths available to the differential drive robot (the range of motion is limited compared to the differential drive robot). However, it is possible to drive *car-like* robots with the differential drive plugin, as shown in previous research [19][10].

3.5.3 Holonomic-robots

Holonomic robots such as the *Youbot*[11] or the *Ridgeback*[12] robot can move instantly in x, y and *theta* directions through a round wheel design and individual motors at each wheel. They are also referred to as *omnidirectional*. This type of robot is not yet supported by the *Flatland* Simulator, but can be used in the *Gazebo* Simulator. The action-space of these robots is compared to the differential drive robot much larger due to the additional velocity dimension. See a visualization of the action-space in Figure 3.9.

[9] https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/
[10] https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/
[11] https://www.locomotec.com/kuka-youbot
[12] https://clearpathrobotics.com/ridgeback-indoor-robot-platform/

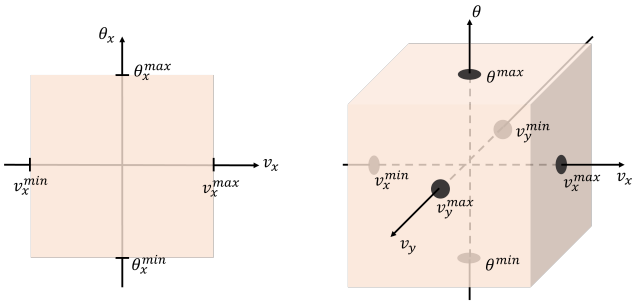


Figure 3.9: The action-space of differential-drive robots compared to holonomic robots

3.6 Local Planner and RL-Based Planner

3.6.1 DWA

The Dynamic Window Approach (DWA) algorithm proposed by [6] is a commonly used local planner as part of the conventional planner collection in the *arena-rosnav-3D* repository. Its basic algorithmic structure can be described as follows:

1. Create a discrete sample of the robot's control space (dx, dy, dtheta).
2. For each distinct option perform a forward simulation, to predict the outcome of applying the control configurations.
3. Rank the performance of each alternative by different factors (such as proximity to obstacles and proximity to the goal, discard illegal trajectories).
4. Select the highest scoring option and send it to move-base.
5. Rinse and repeat.

There are several adjustable parameters of the algorithm and its simulation environment that allow the algorithm to be adapted to the environment [37]. Due to its generic structure, it has been found that it is not suitable (in its basic form) for car-like robots [29].

3.6.2 TEB

The Timed Elastic Band (TEB) algorithm is an algorithm that is executed during run time to optimize the trajectory, originally predicted by the global planner, based on several factors such as execution time and distance from obstacles. The optimal trajectory is defined as a multi-objective optimization problem with several constraints such as velocity and acceleration limits. The user can define the bounds of the specific algorithm with different parameters [35] depending on the specific situation. In contrast to dwa, the algorithm can be used with all kinds of mobile robots, it however tends to get stuck in local minima [34].

3.6.3 Rosnav

The rosnav-based navigation approach is a fundamentally different navigation approach than traditional approaches such as *teb* and *dwa*. Instead of using the entire navigation stack, the *arena-rosnav*s training platform trains an *rl* agent. It takes the robot's position and laser scan data as input to a neural network that predicts the best possible command-velocity path based on incoming observation data.

3.7 Robot Savety

Robot safety measures are of significant importance for using robots in industrial human-robot-collaboration scenarios. In the German context, robotic applications are primarily regulated by the norms EN ISO 12100 and EN ISO 10218. For a safe operation, different safety measures are being used, such as safety zones or emergency stop functions.

The emergency stop function works by equipping the mobile robot with additional sensors such as lidar to monitor the proximity to obstacles in the environment. The basic idea is shown in Figure 3.10. When an obstacle dangerously approaches the robot, the robot first reduces its speed (zone 1). When an obstacle reaches the inner safety zone (safety zone 2), the robot performs an emergency brake (it sets its speed to zero) to avoid a collision with the obstacle.

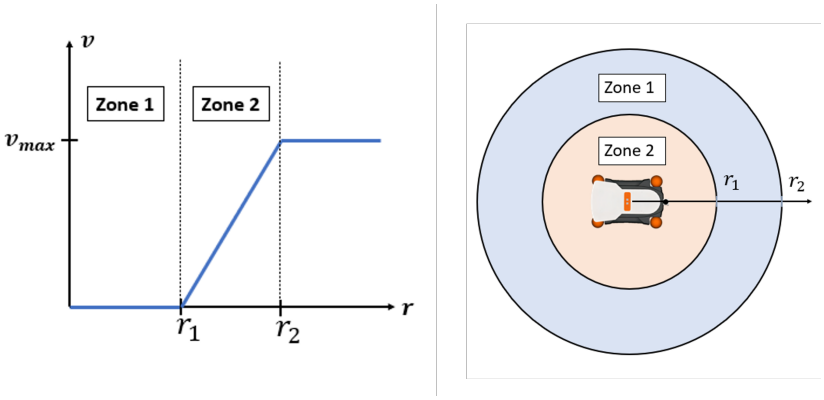


Figure 3.10: The two safety-zones of the emergency brake system and the impact of obstacle distance on the command velocity

The emergency brake safety mechanism is of particular interest to the field of robot navigation since the task of obstacle avoidance is usually performed by global and local planners. The combination of the two approaches to obstacle detection (also referred to as the multi-level approach [15]) is of particular interest to the field of robot navigation, especially since the emergency brake

system reduces the degrees of freedom of the classical navigation approach by reducing the scope of action of the local planner.

There are various approaches [32] for the practical implementation of the multi-level safety mechanism in the navigation stack. The most common is a simpler implementation of the emergency stop system that includes only the safety zone 2 mechanisms. The stopping behavior is achieved by either publishing the navigation goal at the robot's current position or canceling the navigation goal with *actionlib*[13]. The goal position then gets set, published when the safety zone has cleared.

In practice, the emergency brake system has several disadvantages. Firstly, unintentional stops can occur, e.g. due to damaged distance sensors. Another common problem is obstacles that are permanently located in the safety zone (zone 2), e.g. because the robot has moved too close to a static obstacle. This is especially the case with classical local planning algorithms (such as *dwa*) since the goal of optimizing an objective function can lead to trajectories that are very close to static obstacles, e.g., to avoid a collision with a dynamic obstacle. Human intervention is required to free the robot from the impasse.

3.8 Robot Performance Metrics

Quantifying robot performance is important in the field of robot navigation to compare different algorithms and determine the reliability of an approach under certain conditions. Robot performance must be viewed relative to one or more higher-level goals, such as:

- *collision* probability,
- *path length*
- *time to goal* (time needed to reach a specific goal)
- *curvature / roughness* of the trajectory
- *reliability* (percentage of the situation were the robot was able to create a viable trajectory path)

The performance of the algorithm regarding these metrics is next to the algorithm also dependent on external factors. These environmental factors are the subject of intensive research, as discussed in Section 2. Since these metrics are not always intuitive, we will briefly discuss the metrics included in *arena-rosnav-3D*.

- The *occupancy ratio* measures the ratio of occupied cells to total number of cells

$$\frac{cells_{occupied}}{cells_{total}}$$

[13] http://wiki.ros.org/actionlib

- *Entropy* measures the degree of disorder in the positioning of obstacles in a given map [36]. That is, it measures whether the obstacles are distributed at regular intervals or more randomly. See a visualization of the entropy measure in Figure 3.11.

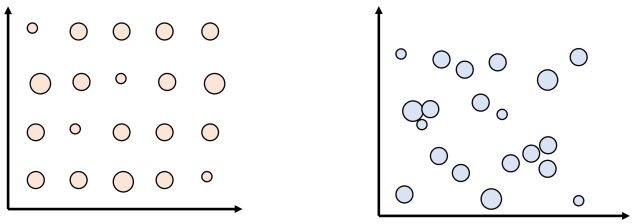


Figure 3.11: Entropy in the obstacle distribution, with a representation of low entropy in the left image and high entropy in the right image.

- *Map size* is a parameter which measures the total size of the map in *width* times *height*.
- The *hallway width* measures the minimal distance between obstacles which results in the free space available for the robot to drive through.
- The *free space angle*measures at every given point the angle of not with obstacles occupied areas. See Figure 3.12.

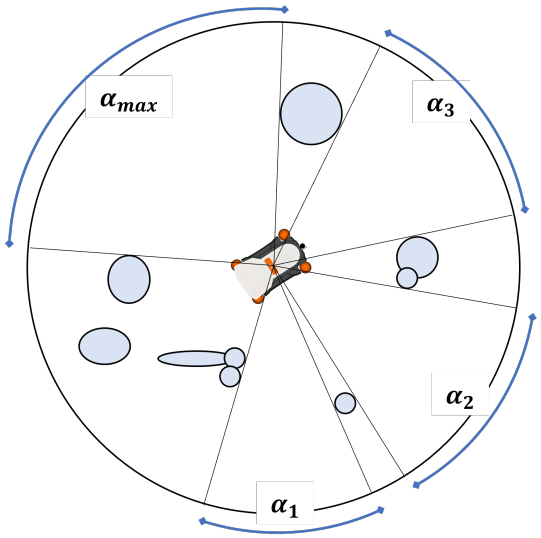


Figure 3.12: The free-space angles of the robots observation-space

Some of these parameters gain their special significance in combination with other factors and are therefore combined in some cases to find correlations in higher dimensions.

In addition to factors from the static environment, the robot's performance is

also affected by factors from its dynamic environment (these factors are usually defined in the *arena-rosnav* scenarios), such as:

- *Number* and *speed* of dynamic obstacles interacting with the robot trajectory.
- *Speed* and *size* of the robot model
- *Size* of the dynamic obstacles

4 Conceptual Design

The following section intends to give a high-level overview of the systems introduced to the arena suite by featuring its requirements and system design. In particular, the interaction of the relevant *arena-rosnav-3D* modules, the implementation of the *emergency-brake* system and the newly introduced *data generator* mode will be discussed.

4.1 The Arena-Benchmark Platform

The arena benchmarking suite (see its rough design structure in Figure 4.1) consists of the 2D simulator *Flatland* with *arena-rosnav,* the 3D simulator *Gazebo* with *arena-rosnav-3D*. Both are similarly integrated into the arena support structure.

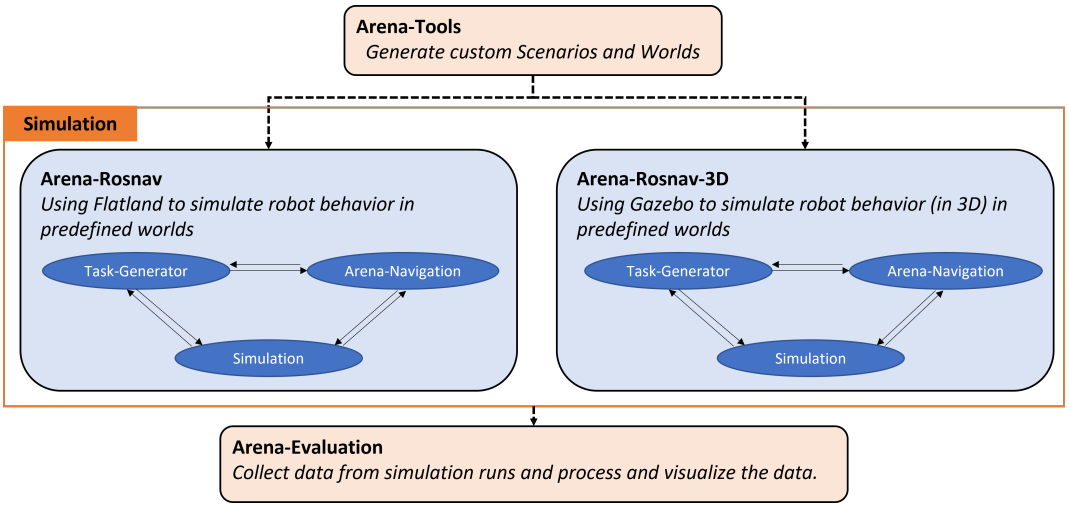


Figure 4.1: A rough system design of the arena-suite and the interplay between the different repositories

As part of the arena support structure, the *task-manager* is responsible for managing the simulation, setting, and managing obstacles and robots. The *arena-bringup* module is responsible for starting the simulation and the simulation supporting components like the *task-manager* and the navigation components.

To compare the 2D and 3D simulators in uniform environments (worlds) and

scenarios, *arena-tools* can be used to create scenarios (both in pedsim format, in case pedsim is used for obstacle management, and in classic *Flatland* format). *Arena-tools* can also be used to create custom environment maps and worlds that can be used in both 2D and 3D simulations. *Arena-tools* must be integrated into the arena environment to fully support both 2D and 3D environments.

To evaluate the performance of the planner, robots, and worlds, the *arena-evaluation* module/node has been created. This node subscribes to information published by the simulation and utilizes this data by evaluating it regarding several metrics. To use *arena-evaluation* as a stand-alone *arena*-component it must be compatible with both the 2D and 3D simulation in *'scenario'* mode.

In the scope of this work, the previously stand-alone *arena* components, such as the *arena-rosnav* repository, have been unified in the arena suite to enable smooth usability of the entire arena suite.
A more detailed description of the individual modules and sub-modules of the arena-suite can be found in section 5.1.

4.2 The Emergency Brake System

The *emergency-brake* system is implemented in a multi-stage design. The robot has two safety zones, zone 1 and 2. When the laser scanner detects an object within safety zone 1, the robot reduces its target speed. The velocity is decreased following a linear function, by decreasing the velocity, the closer the object gets to zone 2. If an obstacle is detected in zone 2, the actual emergency stop is triggered by setting the command velocity to zero. The safety zone concept has been visualized in Figure 3.9, the system design of the emergency-stop mechanism can be found in Figure 4.2.

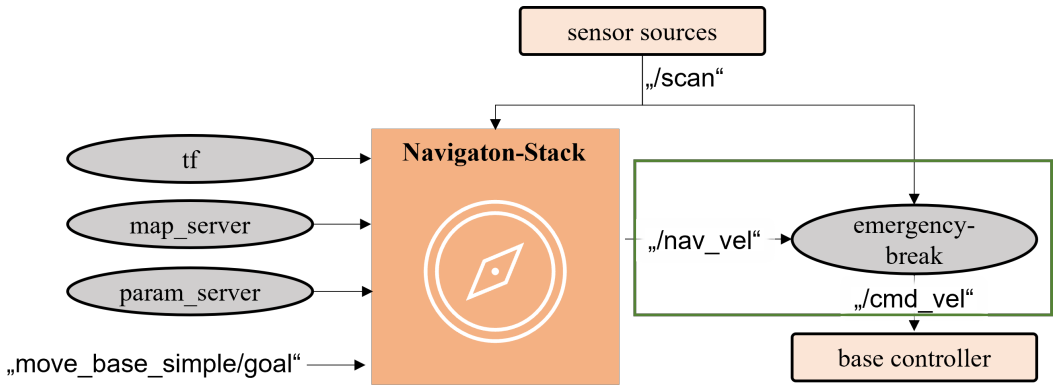


Figure 4.2: The system design of the emergency-brake system

The module follows hereby several requirements:

- The module should be available in the 3D repository on all robots (it could easily be implemented in the 2D repository, but has not been done so far to keep the repository as clean as possible)
- The emergency system does not result in a permanent shutdown; as soon as the safety zone is cleared of obstacles, the robot continues on its original path.
- The emergency system overrides the navigation stack, even goal resets or recovery behaviors cannot override the emergency brake commands.

A more detailed description of the specific modules can be found in section 5.2.

4.3 The Data Generator Mode

The data generator mode is an extension of the *arena-rosnav-3D* repository that allows collecting large amounts of data without the need for pre-defined scenarios, worlds (or robots (this is only implemented to a limited extent)).

The module is however primarily designed to create a large data set that can be used to predict robot performance in a given environment or scenario without the need for research-intensive simulation runs. A design of the entire performance prediction pipeline is depicted in Figure 4.3.

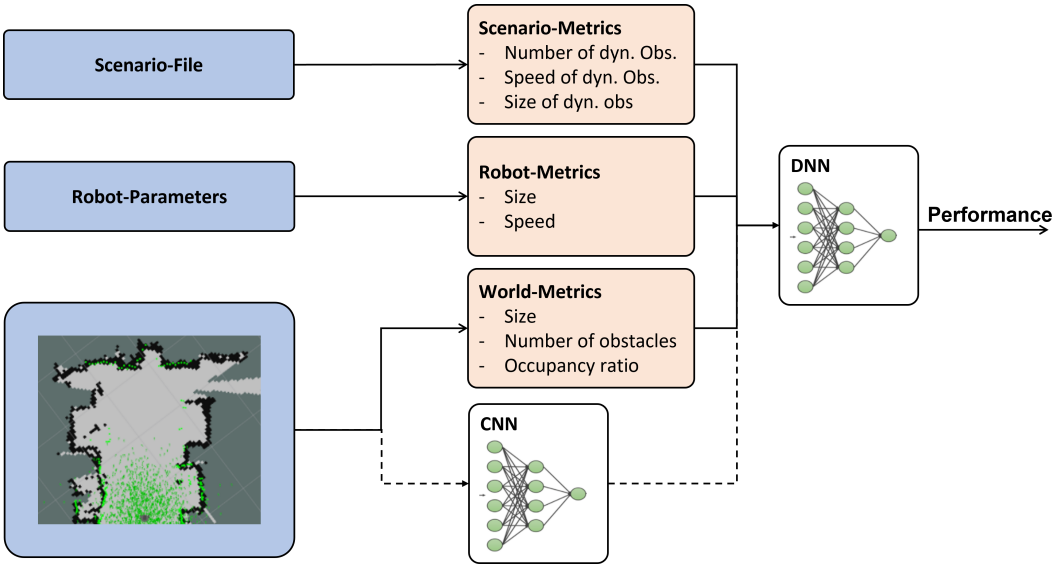


Figure 4.3: The system design for creating a pipeline for predicting robot performance based on performance measurements

The goal of this work is not to create the complete performance prediction pipeline, but to implement the data creation pipeline. A draft of the system design can be seen in Figure 4.4 and is discussed in the following section.

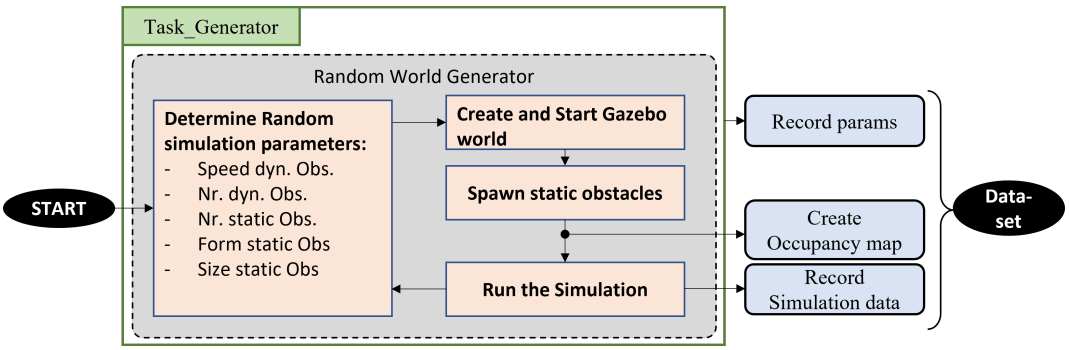


Figure 4.4: The system design of the data-generator mode

The main contribution of the *data-generator* mode is an extension of the *task-generator* node. The *task-generator* is started in *'random'* mode and creates a world randomly using the *outdoor* world generator. The occupancy map of the created world is automatically generated by the *gazebo-ros-2d-map-plugin*[1].

In addition, a random scenario is created (determination, number, and speed of dynamic obstacles from a predefined range). The defined scenario conditions (metadata) are stored in a separate document to enable scenario-based research. The *arena-evaluation* module is used to record the data from the simulation runs and to pre-process the data from the simulation runs to evaluate the data according to different metrics. A more detailed description of the individual modules can be found in section 5.3.

[1]https://github.com/marinaKollmitz/gazebo_ros_2Dmap_plugin

5 Implementation

The following section provides a detailed overview of the implementation of the three main contributions within this thesis. These include:

- The implementation of the different components of the *arena-suite* in section 5.1
- The implementation of the *emergency-brake* system in section 5.2
- The implementation of the *data-generator* platform in section 5.3

5.1 The Arena-Benchmark Platform

The arena suite consists of four main repositories with a large number of support repositories. See Figure 5.1 for an overview of the repositories used and their interaction. The design of these repositories has been the subject of other research projects/reports (and will therefore not be the primary focus of this work). The main contribution of this work is to combine the individual repositories into a large-scale benchmarking platform that includes benchmarking preparation (with *arena-tools*), benchmarking in 2D and 3D environments (with *arena-rosnav* and *arena-rosnav-3D*), and benchmarking evaluation (with *arena-evaluation*).

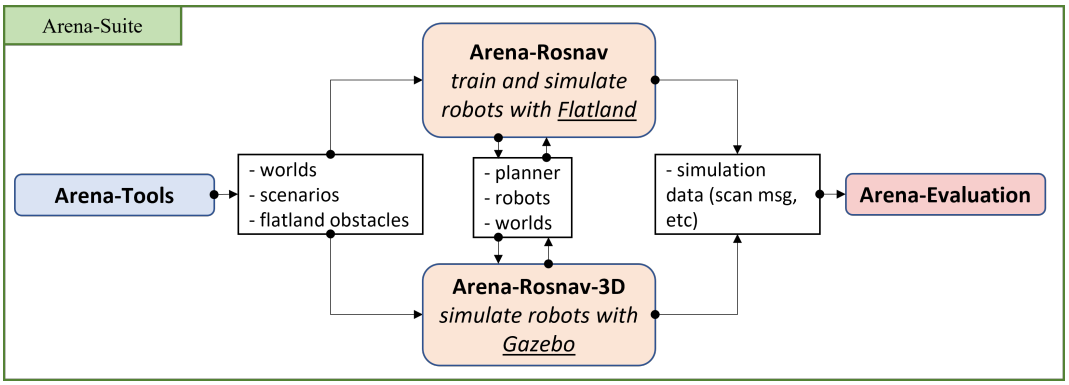


Figure 5.1: Overview of the major components of arena-suits and their interaction

The following section thus focuses on the description of the implemented interfaces between the platform components.

5.1.1 Arena-Tools

Arena-tools performs three main functions:

- The creation of custom *occupancy maps* (2D) in different difficulty tiers / feature classes
- The creation of custom benchmarking scenarios in *Pedsim* format.
- (The creation of custom *Flatland* models) (Note: This feature should theoretically work, but has not yet been tested, as it has not yet been shown that the creation of more detailed models has a significant impact on the performance of the motion planning algorithm)

The occupancy maps created by *arena-tools* are stored directly in the corresponding directory in the 2d and 3d simulation repository. However, for use within the *Gazebo* simulation, an additional step must be taken. Using Blender, the 2d occupancy map must first be converted to a *mesh* object and this *mesh* object must then be included in a *Gazebo* World file.

The custom benchmarking scenarios created by *arena-tools* are written in *Pedsim* format. *Pedsim* was originally used for obstacle management in *arena-rosnav* and *arena-rosnav-3D*. Due to various limitations of the pedsim-simulator, the default method for simulating obstacles in both repositories has been changed. For the *arena-rosnav* repository, scenarios can be converted to the now default *arena*-obstacle manager using the *ped_to_arena.py*[1] script. For the *arena-rosnav-3D* repository, the original pedsim scenarios do not need to be converted. This step is done by the *world-generator.py*[2] script in the *arena-rosnav-3D* repository.

5.1.2 Arena-Rosnav and Arena-Rosnav-3D

The simulation repositories *arena-rosnav* and *arena-rosnav-3D* have a similar structure (the interaction of their core components is shown in Figure 5.1). The main difference between the repositories is the different simulation engines and the resulting differences in the management of the simulation objects (robots, obstacles, etc.).

To be able to simulate identical scenarios in both 2D and 3D environments, extensive documentation was created to transfer concepts and models between the simulation environments (since there are significant differences between the two simulation worlds, it was not possible to automate these processes). The documentation includes:

[1] https://github.com/ignc-research/arena-tools/blob/main/utils/ped_to_arena.py

[2] https://github.com/ignc-research/arena-rosnav-3D/blob/main/task_generator/scripts/generate_world.py

- How to install both repositories side by side[3].
- How to include robot models in both environments[4].
- How to include navigation approaches in both environments[5].

5.1.3 Arena-Evaluation

The *arena-evaluation* repository is responsible for recording and evaluating the simulation process for various specified metrics. Data is recorded via a data recorder node that is started at the beginning of the simulation and subscribes to relevant *odometry* and performance data (e.g., robot speed). As part of this work, the *arena-evaluation* repository was extended to more accurately capture the simulation process by taking into account factors such as simulation/*real-time* and laser settings on the robot models. Furthermore, several additional metrics such as world complexity metrics and variance evaluation metrics were included. An overview of the data processed by *arena-evaluation* and the metrics calculated can be found in Figure 5.2.

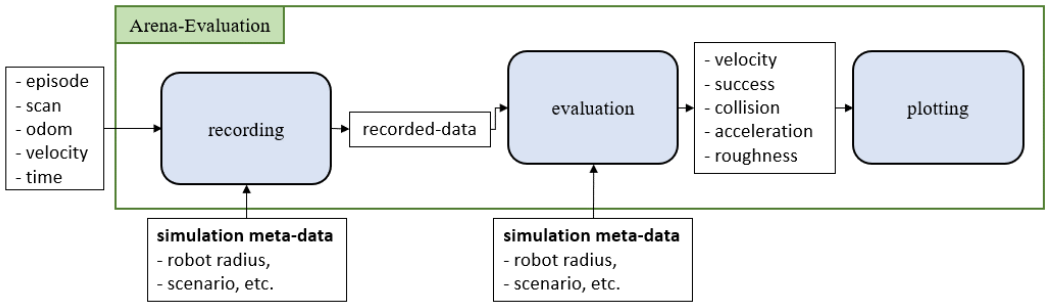


Figure 5.2: The basic structure of the arena-evaluation repository

5.2 The Emergency Brake System

To emulate the safety behavior of AGVs in real-world applications, the emergency brake note was introduced, which adjusts the behavior of the robot based on the current laser scan measurements. The feature entails the introduction of several virtual proximity zones around the robot, which affect the command speed of the robot.

[3]https://github.com/ignc-research/arena-rosnav-3D/blob/main/docs/Installation.mdadd-arena-rosnav-next-to-arena-rosnav-3d

[4]https://github.com/ignc-research/arena-rosnav-3D/blob/main/docs/Miscellaneous.mdhow-to-include-additional-robot-models

[5]https://github.com/ignc-research/arena-rosnav-3D/blob/main/docs/Usage.mdhow-to-add-new-navigation-approaches-in-gazebo-and-flatland

The node is integrated by subscribing to the speed published by the navigation stack under the topic *"/nav_vel"*. The emergency brake node, in turn, republishes the velocity under the topic *"/cmd_vel",* which is subscribed to by the *base-controller* to be converted into robot control commands. When an object is detected within the two proximity zones, the velocity is changed before it is republished, following the rules outlined in section 4.2, as shown in Figure 3.10.

However, a major challenge of the *emergency-brake* system is that the navigation algorithms used in the simulation are not designed to account for proximity zones and emergency-brake. The trajectories are calculated by optimizing an objective function. Under certain conditions, the solution found by the planning algorithm may be to drive close to obstacles. However, in the worst case, this would lead to a permanent standstill if the second approach zone was violated.

An additional difficulty is an effect on the action range of the planner. Because of the proximity zones, many actions such as closely passing static obstacles are not possible (as it would violate the second proximity zone). And passing obstacles inside the first zone of proximity is less attractive because it affects the speed of the robot).

To address this issue, the (virtual) robot radius within the cost map was extended to the actual robot radius + virtual zone 2. This can be seen in Figure 5.3. This forces the conventional planner to consider the virtual zone. The impact of this navigation action is evaluated in the following section. For rl-based planners such as rosnav, this approach is not possible because the motion planner is trained on the specific robot radius.

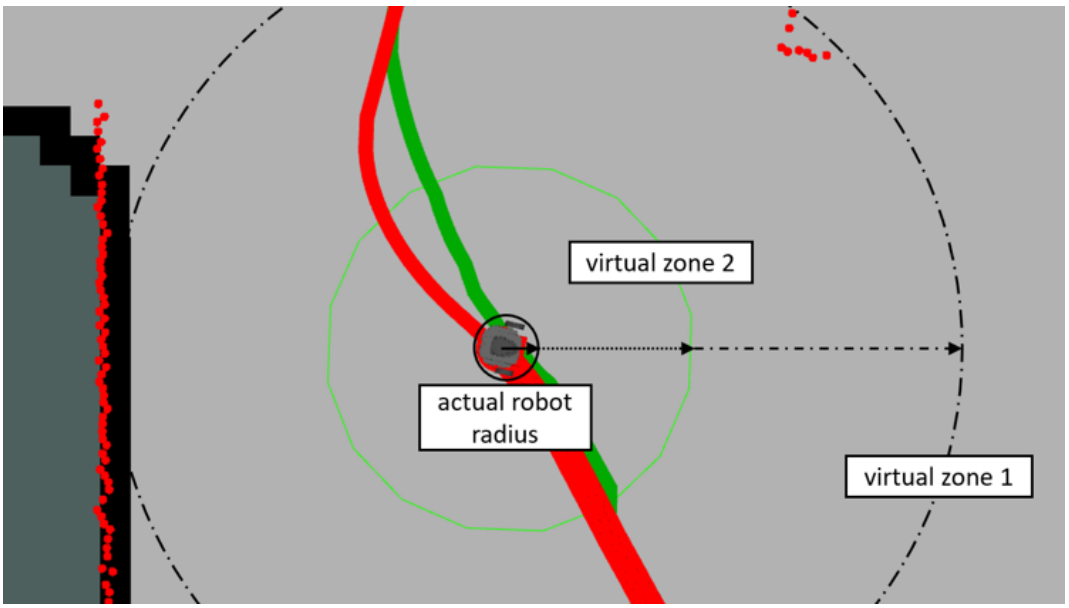


Figure 5.3: The burger robot in simulation with activated emergency stop and enlarged footprint

In the current configuration, the *emergency-brake* system can be started by setting the corresponding parameter in the launch file to *true*. It is implemented only in the *arena-rosnav-3D* repository so far. However, it could easily be transferred to its 2D counterpart. The radius of proximity zone 1 is set to $1m$ from the laser scan origin, while the radius of proximity zone 2 is set to $0.5m$ from the laser scan origin.

5.3 The Data Generator Mode

The data generator mode builds on the random task mode and the *'random'* world generator of the *arena-rosnav-3D* repository and is located in its own branch of the *arena-rosnav-3D* repository[6]. The basic idea of the design is to define the range between which random worlds should be generated before launch. It also records not only the robot's performance data but also specific simulation designs such as the number of static obstacles and the speed of dynamic obstacles. The mode is designed to allow testing of robot concepts in hundreds of different scenarios, enabling broader testing and prediction of robot concepts as well as training of predictive AIs on large data sets.

The data generator mode is started by selecting the random task mode and the outside mode when starting the simulation. In addition, several parameters can be set before the start, such as:

[6]https://github.com/ignc-research/arena-rosnav-3D/tree/data-generator

- (Max) number of static obstacles
- (Max) number of dynamic obstacles
- (Max) speed of dynamic obstacles

After the simulation is started, it goes through the following steps:

1. Determine the specific parameters (e.g. number of static obstacles) for this simulation run from the possible range defined at the launch.
2. Create a *Gazebo* world with the parameters defined in the previous step.
3. Start the *Gazebo* simulation.
4. Use the *gazebo_ros_2Dmap_plugin* to create a 2D occupancy map of the *Gazebo* world
5. Starts the robot and sets the start and goal positions
6. The robot travels to the navigation goal, the robot's performance is recorded by the *arena-evaluation* node.

The simulation is set up to run indefinitely but can be stopped at any time. To create a large data set for predicting global performance, the following steps have to be taken:

1. Evaluate the recorded simulation data, to distill performance metrics
2. Merge the simulation data, by its meta-data (e.g. number of obstacles)

6 Evaluation

To demonstrate the unique value and gain a better understanding of the components of the arena-suite, extensive amounts of data were collected to answer the following research questions:

Research Question	**Formulation**	**Section**
RQ1.0	Is there a significant difference between simulating robots in 2D *Flatland* environments and 3D *Gazebo* environments - are the additional resources for more realistic 3D environments justified by the results achieved?	6.1.2
RQ1.1	Do the simulation results (in terms of mean and variance) differ between the *robot*-types?	6.1.2
RQ1.2	Do the simulation results (in terms of mean and variance) differ between the different *planner* types?	6.1.2
RQ1.3	Do simulation results (in terms of mean and variance) differ between different *worlds*?	6.1.2
RQ2.0	Does the use of an emergency-brake system have a positive effect on navigation safety?	6.2.2
RQ2.1	Within which safety tier perform our simulated robots by robot-type and planner?	6.2.2

Table 6.1: Overview of the research questions addressed in this work

6.1 Gazebo - Flatland comparison

6.1.1 Experiment Design

Since the arena suite supports a large number of robot types (planners, etc.), only a small subset of their features was considered for this work, each representing functionalities of a specific class. For each unique combination of environment parameters (robots, planners, obstacles, worlds), a simulation scenario was prepared and run with a total of 30 resets. (30 resets were performed to be able to assume a standard distribution for the specific performance parameters). An overview of the simulation runs performed can be found in Table 6.2.

Simulator	Planner	World	Robot and Obstacles					
			burger	jackal	youbot	burger	jackal	youbot
			5 Obs			10 Obs		
Gazebo	dwa	small-wh	30	30	30	30	30	30
Gazebo	teb	small-wh	30	30	30	30	30	30
Gazebo	rosnav	small-wh	30	30	30	30	30	30
Flatland	dwa	small-wh	30	30	30	30	30	30
Flatland	teb	small-wh	30	30	30	30	30	30
Flatland	rosnav	small-wh	30	30	30	30	30	30
Gazebo	dwa	map2	30	30	30	30	30	30
Gazebo	teb	map2	30	30	30	30	30	30
Gazebo	rosnav	map2	30	30	30	30	30	30
Flatland	dwa	map2	30	30	30	30	30	30
Flatland	teb	map2	30	30	30	30	30	30
Flatland	rosnav	map2	30	30	30	30	30	30
Gazebo	dwa	map5	30	30	30	30	30	30
Gazebo	teb	map5	30	30	30	30	30	30
Gazebo	rosnav	map5	30	30	30	30	30	30
Flatland	dwa	map5	30	30	30	30	30	30
Flatland	teb	map5	30	30	30	30	30	30
Flatland	rosnav	map5	30	30	30	30	30	30

Table 6.2: Overview of the simulation runs performed to evaluate the difference between the Gazebo and the Flatland simulator

Environmental parameters

The three robot types (see Figure 6.1) were selected for their different characteristics. Their specifications can be found in Table 6.3.

turtlebot3-burger jackal youbot

Figure 6.1: Robot models used in the simulation runs[22][23][17]

Robot	Speed (v_x) [m/s]	Speed (v_y) [m/s]	Speed (θ_y) [rad/s]	Laser-range [m]	Holonomic
Burger	0.22	0.0	2.84	0.113	FALSE
Jackal	2.0	0.0	4.0	0.267	FALSE
Youbot	0.8	0.8	1.2	0.347	TRUE

Table 6.3: Specifications of robots used in the simulation runs

The planners were chosen because the local planners *dwa* and *teb* are very commonly used and therefore provide a good basis for comparing conventional

navigation approaches with *rosnav* navigation based on *rl*. The worlds (see Figure 6.2) were chosen because they represent robot environments with different levels of difficulty (see Table 6.4). The number of dynamic obstacles was chosen to increase the (dynamic) occupancy of the navigation map. The trajectory of the dynamic obstacles was chosen to intersect possible navigation routes of the robot.

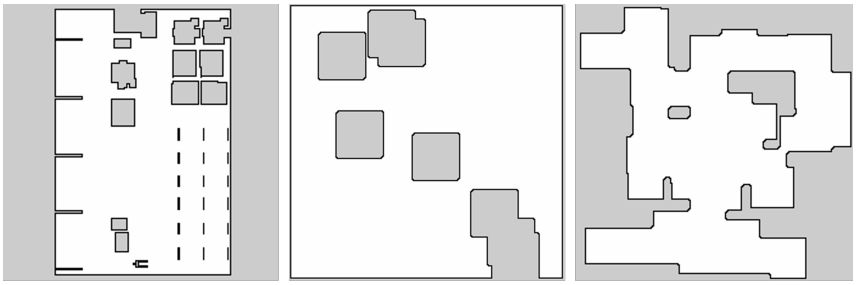


Figure 6.2: World models used in the simulation runs

World	Size [m²]	OccRatio	Entrophy	Num of Obstacles
small-warehouse	6250	0.032	0.265	30
map2	6250	0.030	0.244	5
map5	6250	0.027	0.233	3

Table 6.4: Specifications of worlds used in the simulation runs

Performance Parameters

The performance of a robot is being measured by several parameters. These are:

- **Success**: A binary classifier that determines whether the robot has reached the navigation goal within the specified time frame and below the collision threshold
- **Collisions**: Measures the number of collisions of the robot with obstacles by checking whether obstacles were registered by the range laser in the immediate proximity of the obstacle.
- **Path length**: Measurement of the number of meters the robot needs to reach the navigation goal.
- **Normalized Curvature**: Is a measure to quantify the degree of trajectory change.
- **Roughness**: A value to quantify the trajectory smoothness
- **Angle-over-length**: An alternative measure of smoothness
- **Velocity**: The velocity of the robot [m/s]

- **Jerk**: The jerk of the robot [m/s^3]

Data-Cleansing
The *arena-evaluation* package is capable of accurately recording simulation data, but in running hundreds of simulation runs on multiple computers, some episode records were corrupted (e.g., recording a simulation speed that was greater than the physically possible speed of the robot). (There are several uncontrollable reasons for these outliers, such as command processing timeouts or collisions). To account for this, the data was reduced to include only reasonable results. It was also ensured that the data points were comparable by using the same number of data points for each unique combination of environmental parameters.

6.1.2 Results

RQ 1.0 Comparing Gazebo and Flatland

This section focuses on answering the following research question:

"*Is there a significant difference between simulating robots in 2D Flatland environments and 3D Gazebo environments - are the additional resources for more realistic 3D environments justified by the results achieved?*"

Observation data
In the following Figures 6.3 to 6.5 a comparison of the performance parameter will be made. In this section, the total difference in performance parameters is depicted (ignoring the influencing factors like worlds, robots, etc).

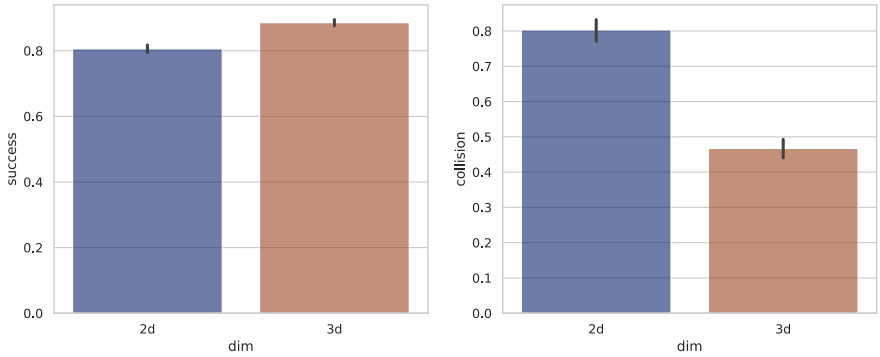


Figure 6.3: Success and collision probability in a 68 percent confidence interval

When looking at Figure 6.3 it can be observed that running robots in a 3D simulation has a slightly (but significant) higher likely-hoot of reaching the navigation goal. This could be due to the significantly lower collision probability when simulating robots in 3D environments. In the following sections, we will

investigate whether this is due to other specific influencing factors (such as more detailed worlds, etc.). Another reason that can be expected to be an influencing factor for the higher collision probability is the different ways of simulating dynamic obstacles. In *Gazebo* simulations, dynamic obstacles are managed with the *actor* concepts. In this case, the *actors* move at a constant speed along a predefined trajectory. *Flatland* provides its dynamic obstacle manager, where the obstacle speed depends on the length of the obstacle's trajectory. This leads to higher obstacle speeds and a larger occupied area in comparable scenarios.

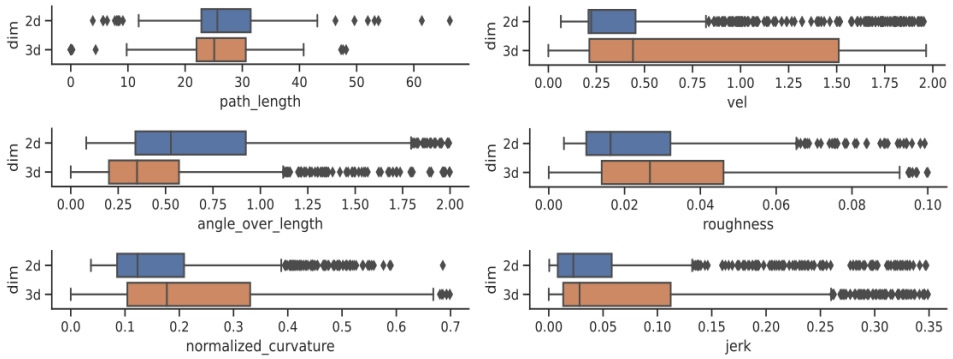


Figure 6.4: Average and variance of additional performance parameters

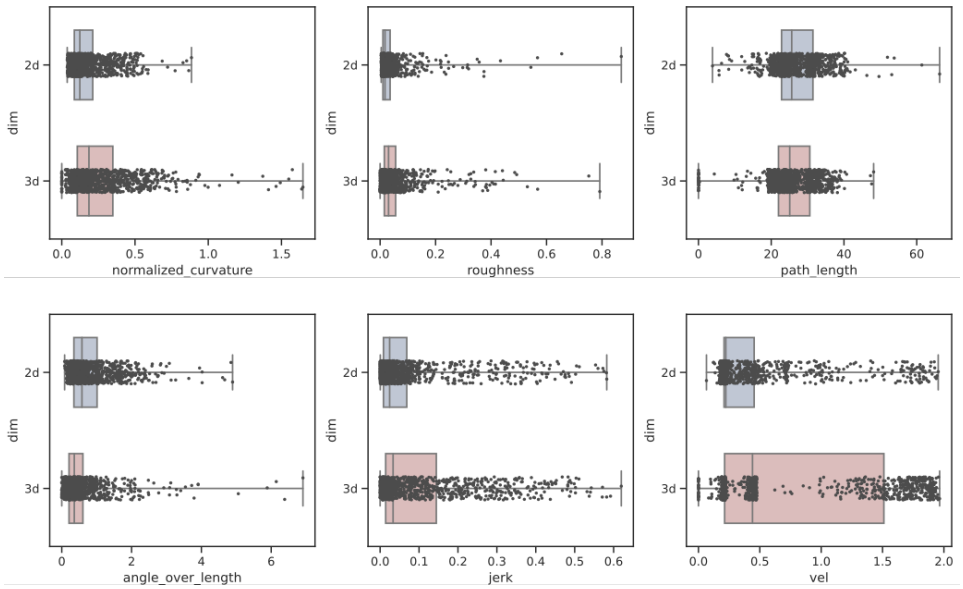


Figure 6.5: Distribution and spreading of performance parameters

Looking at the other performance indicators, a strong *curvature*, as well as a larger variance, can be observed in the *Gazebo* simulations. This is in line with

an increased *jerk* (average and variance) as well as a larger velocity. Regarding *path-length*, it can be observed that simulating robots in 3D environments results in slightly shorter paths with a smaller variance. These factors might be due to the more sophisticated 3D environment, which allows for more precise steering of the robot.

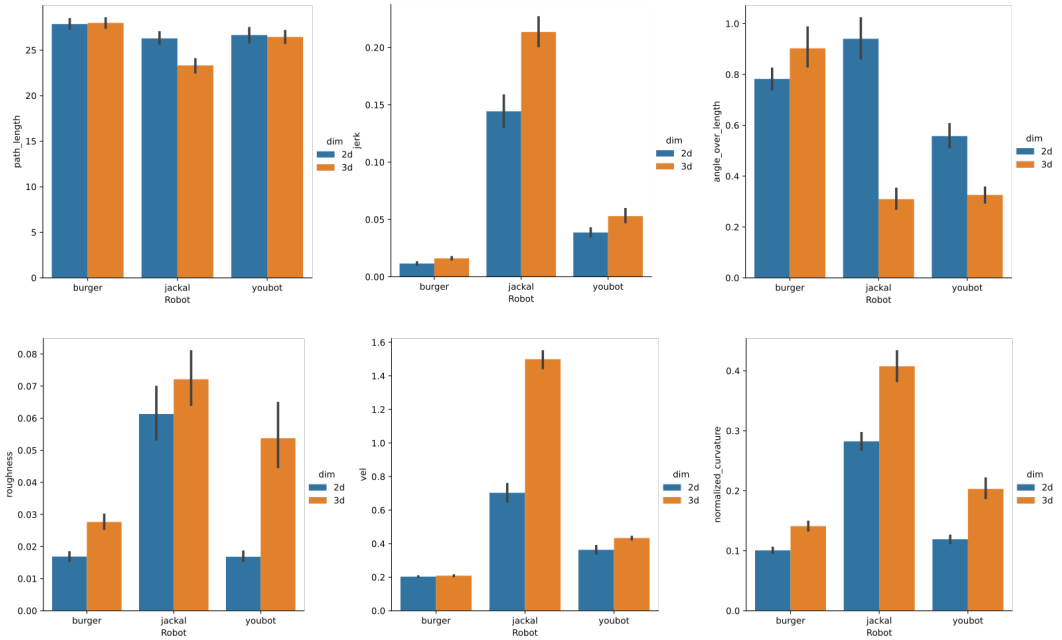
Interpretation
In summary, there are significant differences between simulating robots in 2D *Flatland* environments compared to 3D *Gazebo* environments. Whether these differences justify the use of additional resources will be answered in Section 6.1.3, as the findings of the sub-research questions are substantial, for answering this question comprehensively.

RQ 1.1 Comparing Gazebo and Flatland influence of robot types

This section focuses on answering the following research question:

"*Do the simulation results (in terms of mean and variance) differ between the robot-types?*"

Observation data
In the following Figures 6.6, 6.7 we show the average performance and the distribution between robot types.



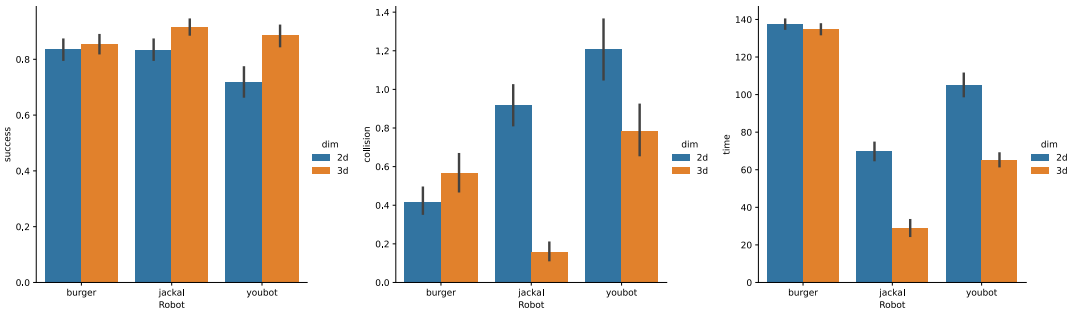


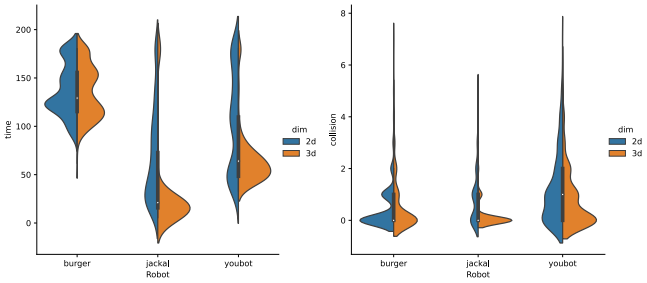
Figure 6.6: Mean and variance of robot performance indicator

Looking at the performance indicator in Figure 6.6, not only are the differences between the robot types notable, but also between the 2D and 3D simulations. Comparing our three robot types, the slower differential-dive-based *burger* robot, with the fast car-like *jackal* and the holonomic *youbot*, we can make the following observations:

- The roughness of the Holonomic-Youbot's trajectory is in between the *burger* and the *youbot*. The larger action space did not result in smoother trajectories (or shorter navigation paths)
- The trajectory length of the fast *jackal* robot (as well as the collision probability) is significantly lower compared to its slower counterparts.

When comparing the 2D and the 3D simulation we can make the following observations:

- The higher the robot speed, the more the 2D and 3D simulations diverge. In Flatland, the *jackal* is used at half the average speed compared to the 3D environment (which in turn affects the other factors such as *jerk*).
- In the 2D environment, the holonomic *youbot* has the highest collision probability, followed by the *jackal*. In this respect, there is a strong difference with its 3D counterpart



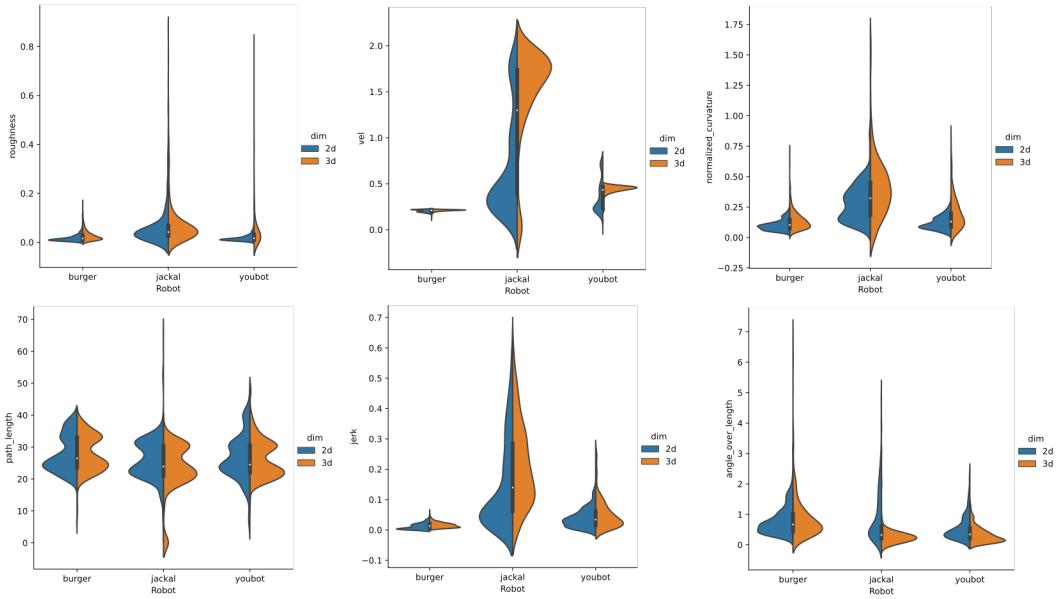


Figure 6.7: Distribution and variance of robot performance indicator

When comparing the distribution of values between robot types, there are some notable observations (besides the different average values discussed in the previous section comparison):

- The time to reach the navigation goal has a much larger variance for the faster robots *jackal* and *youbot*.
- The larger action space of the holonomic *youbot* does not lead to a lower variance (higher predictability)
- The greater speed of the *jackal* leads to higher parameter variance (except for parameters such as *angle-over-length*).

When comparing the 2D and the 3D simulation we can make the following observations:

- The 2D simulation shows a much larger variance in *time-to-goal* compared to the 3D simulation, but this has no significant effect on *path-length* .
- The speed and *jerk* of the car-like *jackal* robot are significantly differently distributed in 2D and 3D environments (although one can expect the variance to be similar).

To observe the impact of the planner on the individual robot performance, Figures 6.8 **:** 6.13 show the impact of the planner on robot performance.

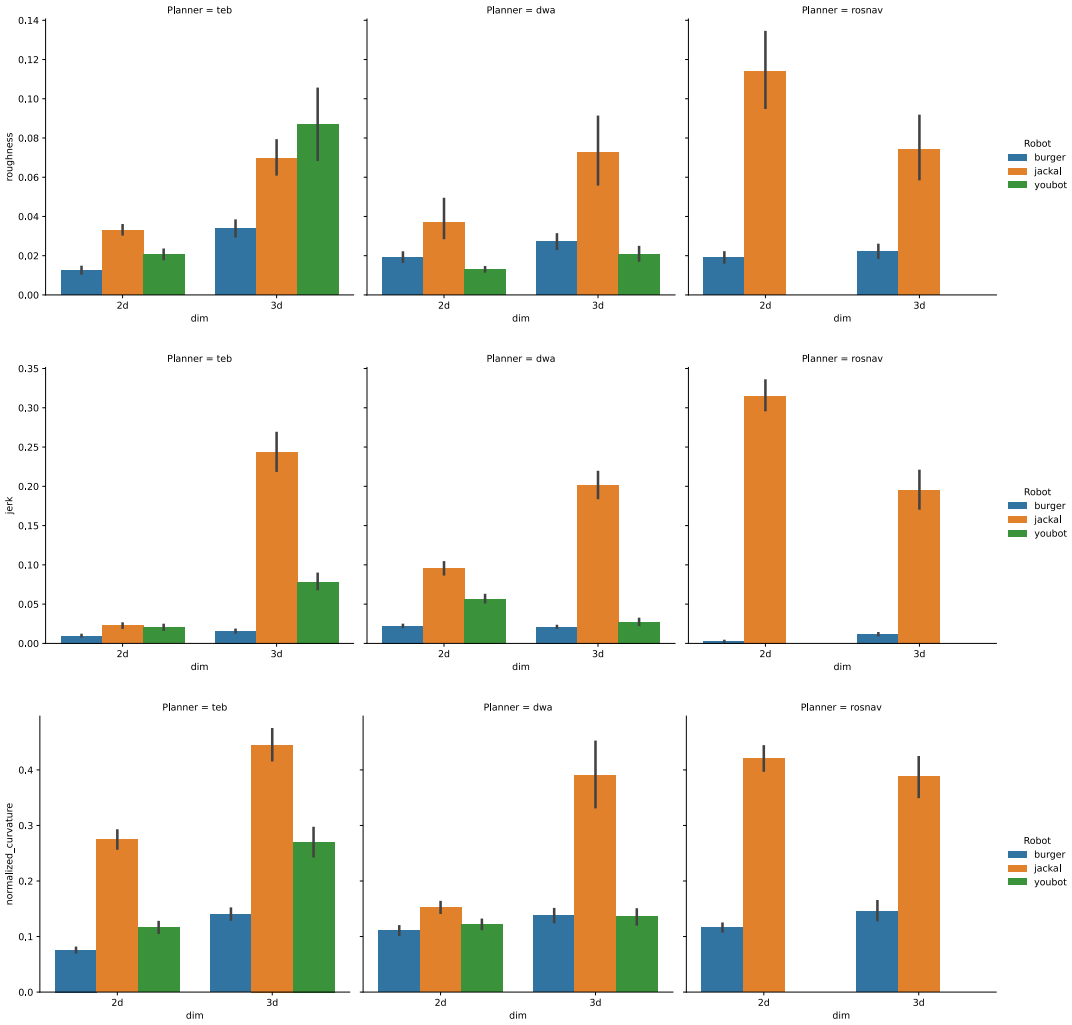


Figure 6.8: Mean and variance of performance parameters by planner and robot type (part1)

In Figure 6.9 the values for *youbot* with *rosnav* are not included. Since it produced inadequate performance and therefore cannot currently provide conclusive information about the planner's performance.

When comparing the average values, *youbot* consistently produces large values. However, it can be seen that there are significant differences between planner performances, even between those of the conventional planner. It is noticeable that teb's local planner produces relatively low *jerk* values in the 2D simulation. When comparing the *jerk* values, it is also noticeable that the *rosnav* planner determines very low *jerk* values for the *burger* robot, while the *jerk* value of the *jackal* is significantly higher.

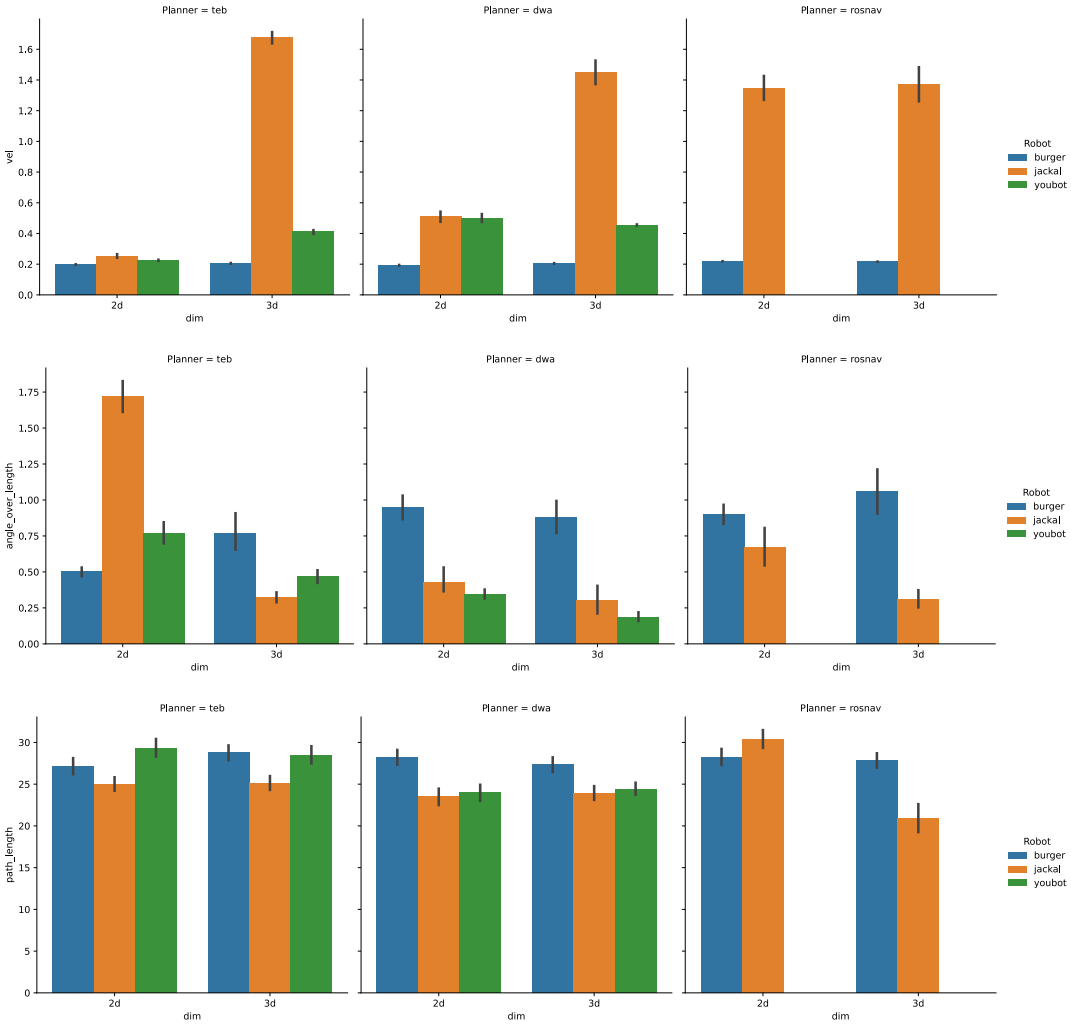


Figure 6.9: Mean and variance of performance parameters by planner and robot type (part2)

Comparing the speed of the robot between the planners, it can be seen in Figure 6.9 that the conventional planners in the 2D simulation tend to use much lower speeds than their 3D counterparts. (In Figure 6.11, it can be seen that larger values are also used, although to a lesser extent.)

When comparing the values for *angle-over-length*, it is noticeable that the values for the *jackal* robot are lower. This can be explained by the different models used in the 2D and 3D simulation since in the 2D world the car-like behavior is approximated by the differential drive plugin.

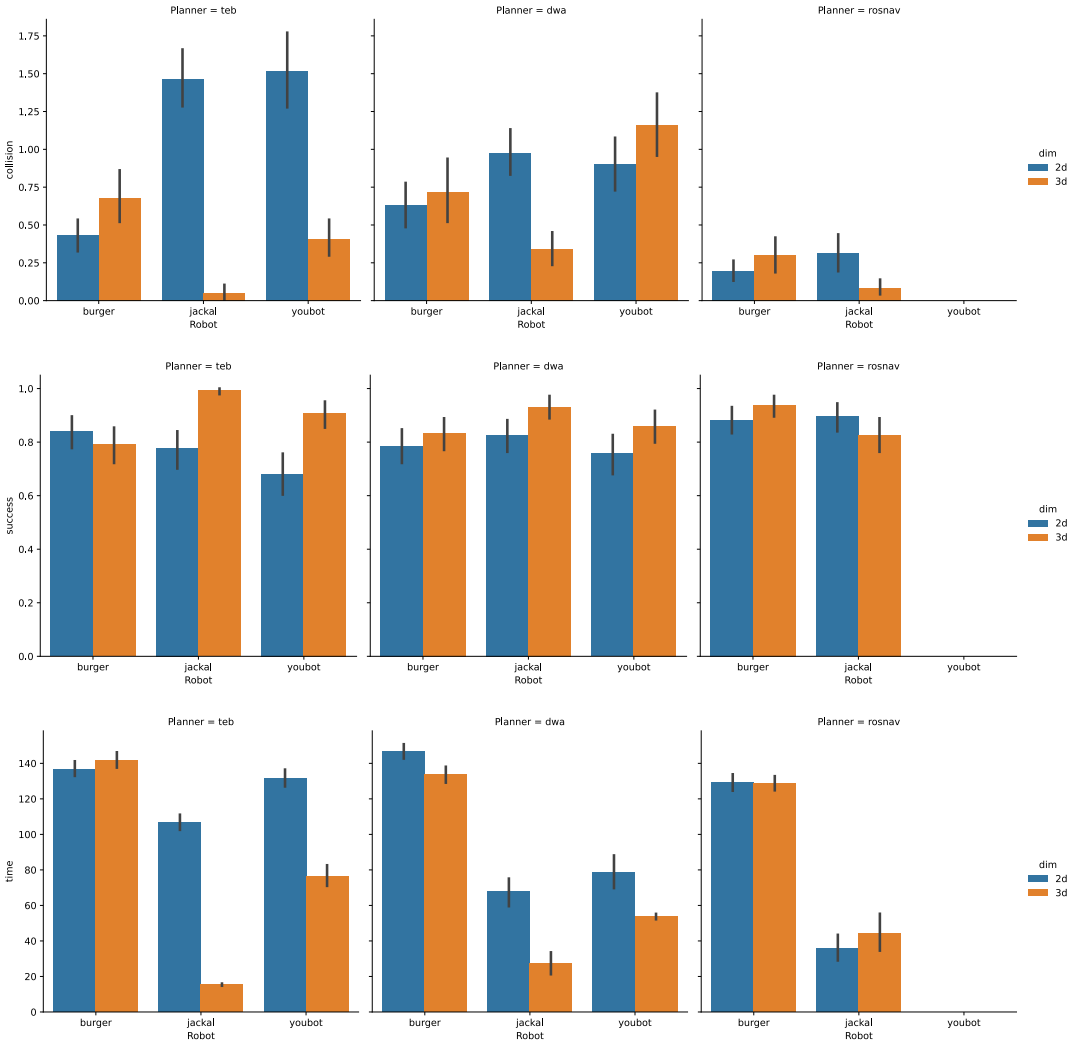


Figure 6.10: Mean and variance of performance parameters by planner and robot type (part3)

Figure 6.10 uses a different approach to comparing the 2D and 3D simulations by using color as a separator. This provides several insights. It can be seen that the *teb* algorithms perform significantly better in terms of collisions in the 3D simulation (except when used with the *burger* robot). In terms of collisions, the *rosnav* agent performs significantly better than its conventional counterparts.

The parameter time to goal shows a clear difference between the 2D and 3D simulation with the conventional planner. In contrast, the *rosnav* planner performs similarly well in the 2D and 3D simulation. The *jackal* robot even performs slightly better in the 3D simulation, although it was originally trained in the 2D simulation.

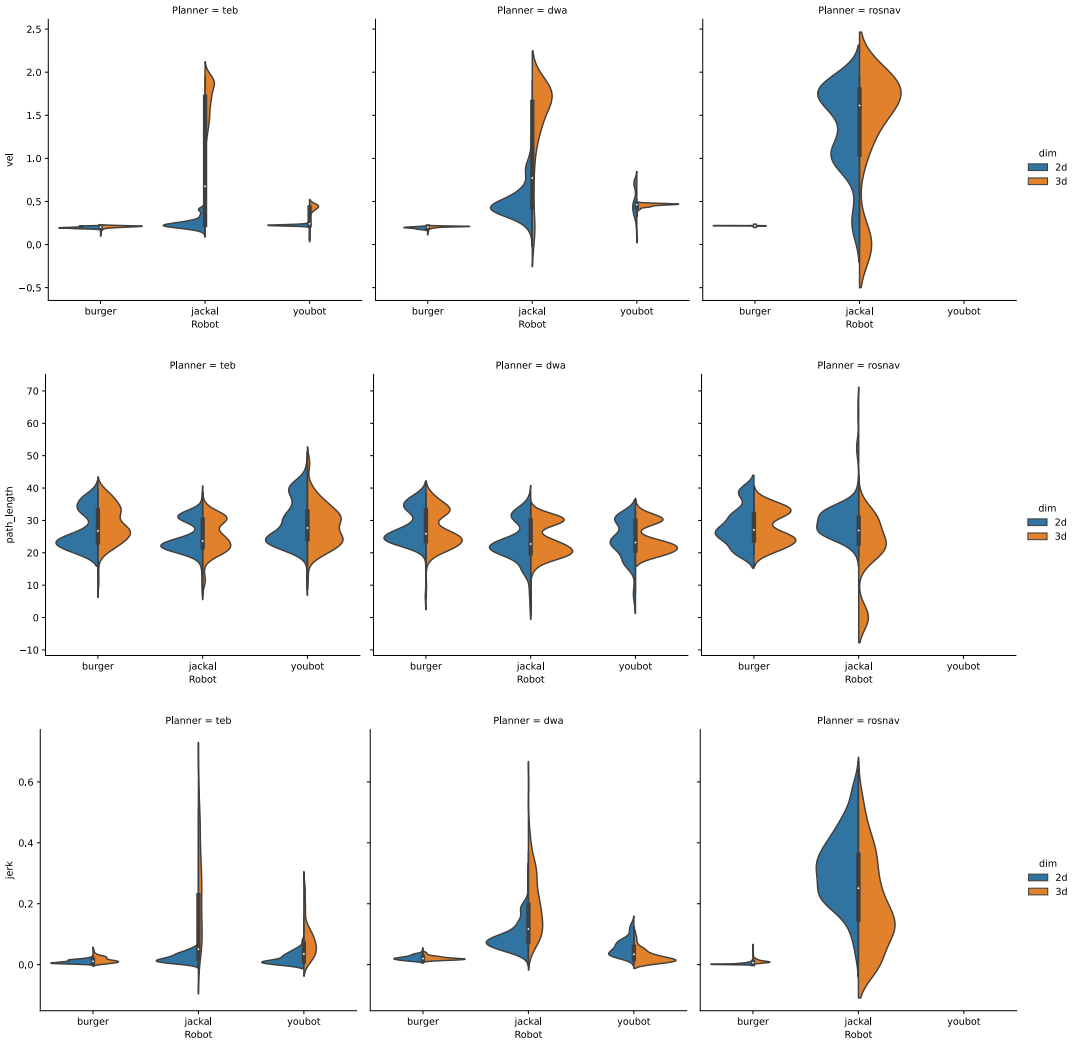


Figure 6.11: Distribution of performance parameters by planner and robot type (part 1)

When looking at the distribution of values, large differences can be seen between the 2D and 3D simulations. It can be observed that the 2D simulation tends to have a lower variance when considering each robot type individually.

It can also be observed that the *burger* robot tends to induce parameters with much lower variance, or the parameter variance seems to increase with the base speed of the robot.

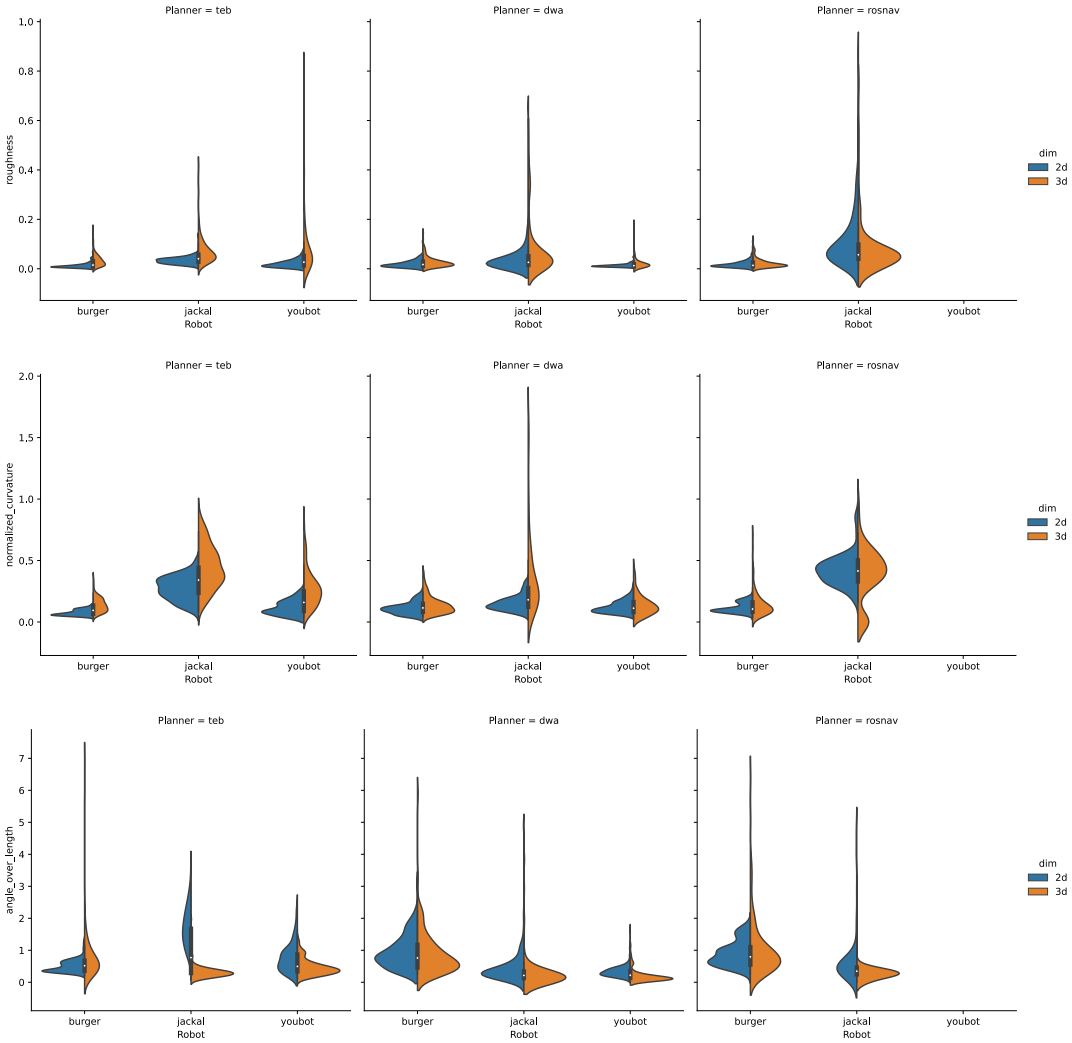


Figure 6.12: Distribution of performance parameters by planner and robot type (part 2)

With the distribution of values in Figure 6.12, the trend observed in the previous figure continues. Even though the average value tends to be similar in the 2D and 3D simulations, the distribution of values tends to be significantly different. With a larger variance in the 3D simulation.

Looking at the distribution values between the 2D and 3D simulations, it can be observed significant differences in the distribution of the values. While the 3D simulation tends to follow a normal distribution of values, the 2D simulation has a much more uneven distribution of values.

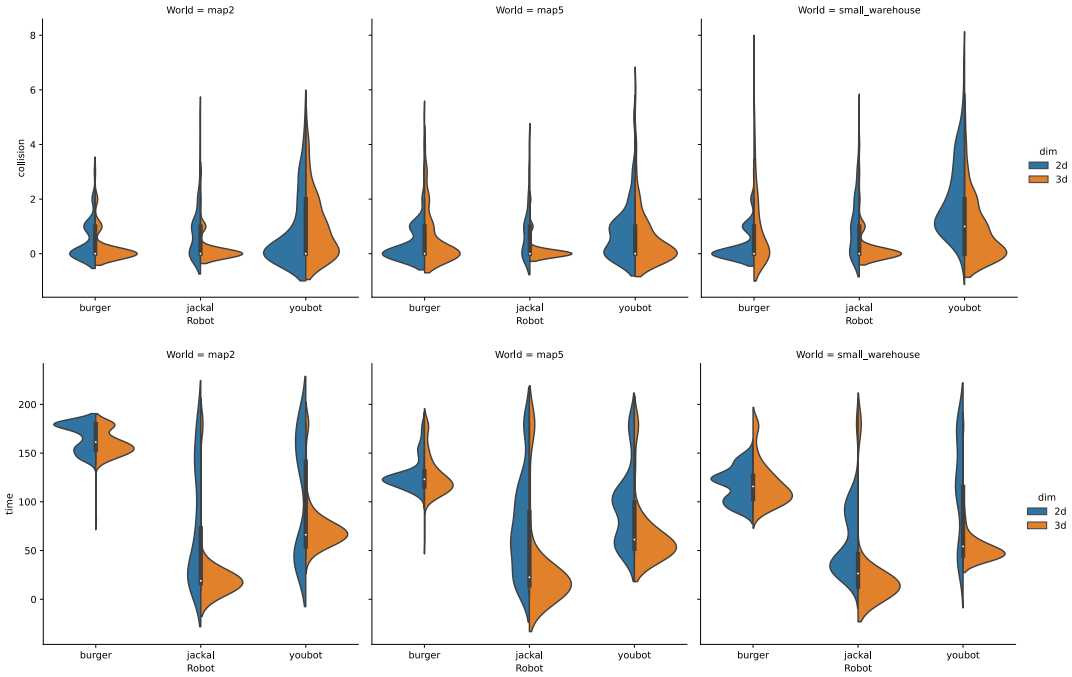


Figure 6.13: Distribution of performance parameters by planner and robot type (part 3)

Looking at the collision distribution in Figure 6.13, it can be observed that the 3D simulation tends to have a lower average value for the distribution as well as a lower spread of values. It can also be observed that the distribution of collision values between robots tends to be similar regardless of the planner.

When comparing the time to reach the goal, the *burger* robot tends to perform equally well in all planners, with the *rosnav* planner providing the best simulation results. For the other robots, the parameter variance in the 2D simulation is significantly greater than in the 3D simulation.

To observe the impact of the map on the individual robot performance, Figures 6.14 to 6.19 show the impact of the planner on robot performance.

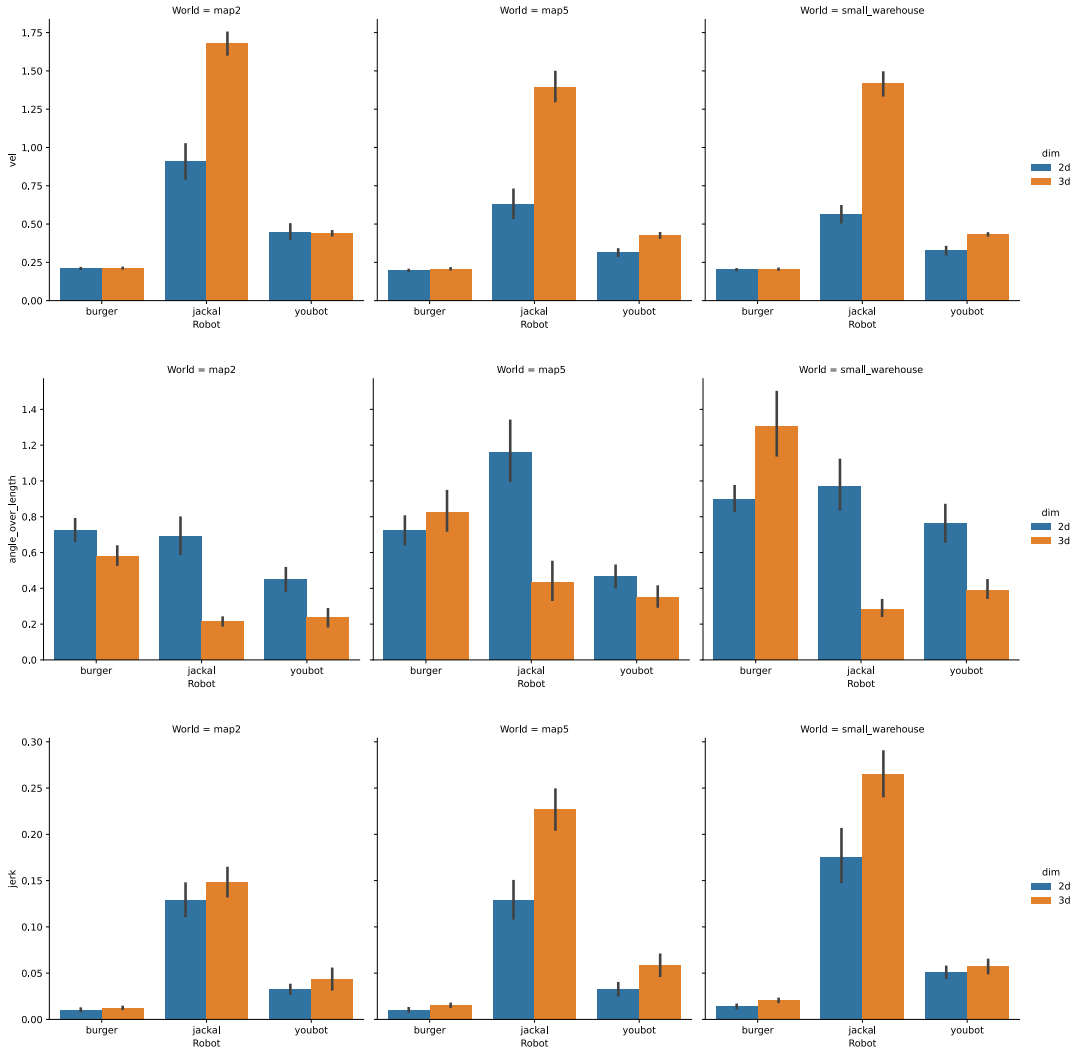


Figure 6.14: Mean and variance of performance parameters by map and robot type (part 1)

When looking at the robot command velocity, it can be seen that the more spacious *map2* leads to an overall higher robot speed (for faster robots), as higher occupancy leads to a lower average robot speed.
It can also be observed that the higher occupancy of the *map5* and *small-warehouse* maps lead to less smoothness of the trajectory. This effect is particularly pronounced for the faster *jackal* robot.

When comparing the 2D and 3D simulations, clear differences between the 2D and 3D simulations become apparent. These can be observed especially for worlds with higher occupancy.

A significant difference between the pre-build *small-warehouse* world and the *arena-tools* worlds can not be observed.

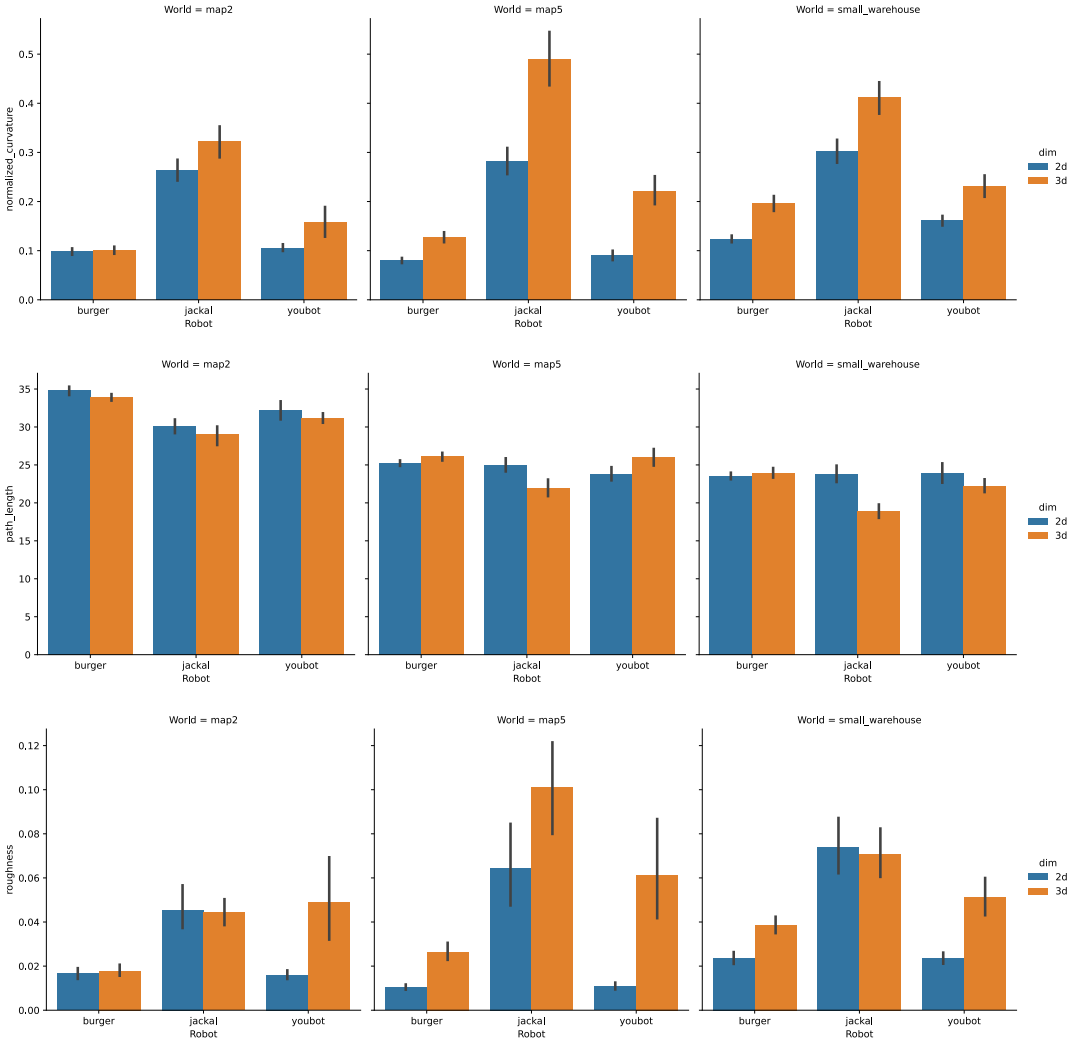


Figure 6.15: Mean and variance of performance parameters by map and robot type (part 2)

The trend observed in the previous Figure can also be found in Figure 6.15 with the maps of higher occupancy resulting in lower path smoothness.
When looking at the paths, there are significant differences between the worlds. This could also be because the best possible path from the start to the goal has different lengths on the different maps. It can also be observed that the average *path-length* of the *jackal* robot is significantly shorter in the 3D simulation than in the 2D simulation.

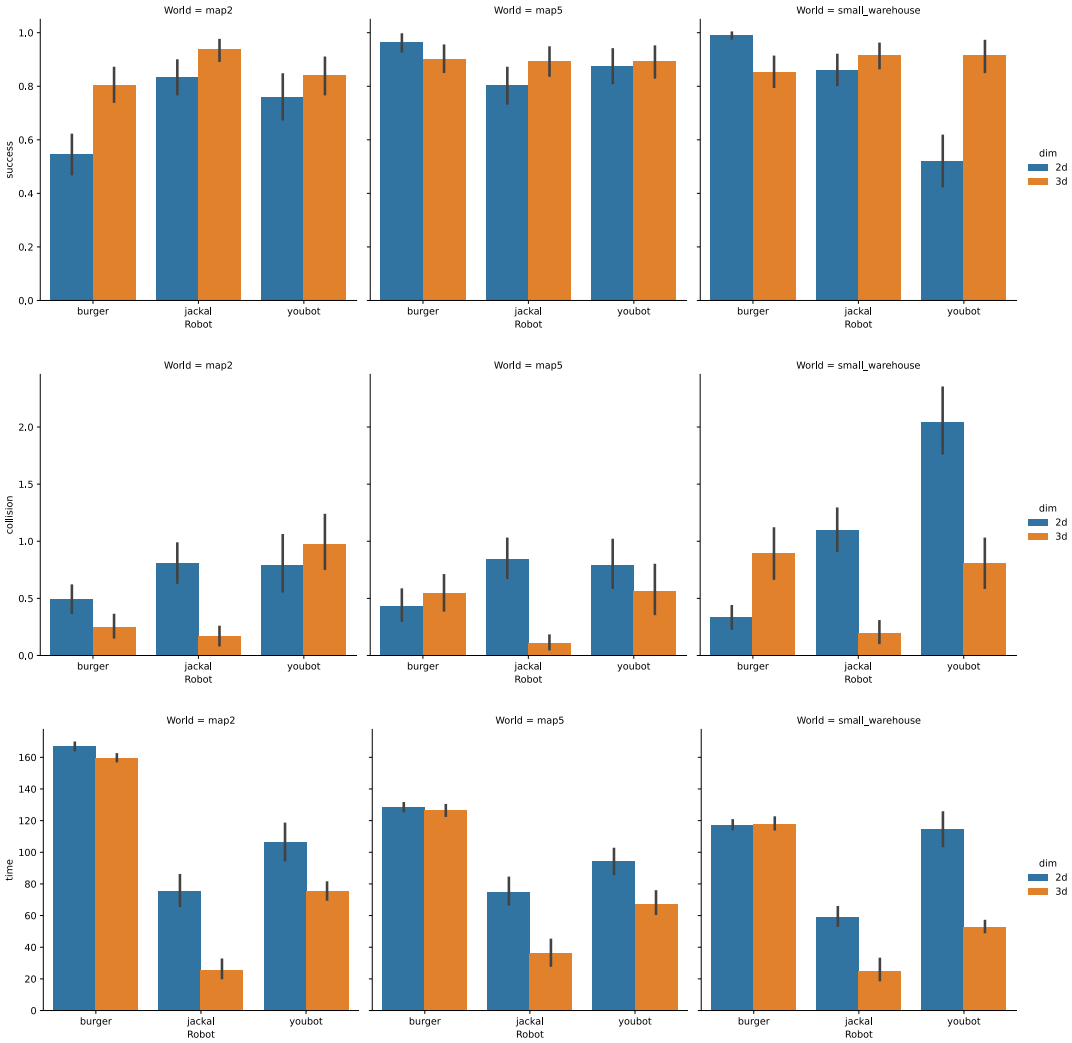


Figure 6.16: Mean and variance of performance parameters by map and robot type (part 3)

When observing the *success* of the robot it can be observed that the two robots performed significantly differently in the 2D simulation compared to the 3D simulation. The *burger* robot performed significantly worse than the *jackal* robot in *map2* in the *small-warehouse* world (although it should be noted that the *youbot* robot is the largest robot in terms of size). The low success rate of the *youbot* robot is correlated with a disproportionately high number of collisions, and the low success rate of the *burger* robot is correlated with the large *time-to-goal* time of the *burger* robot.

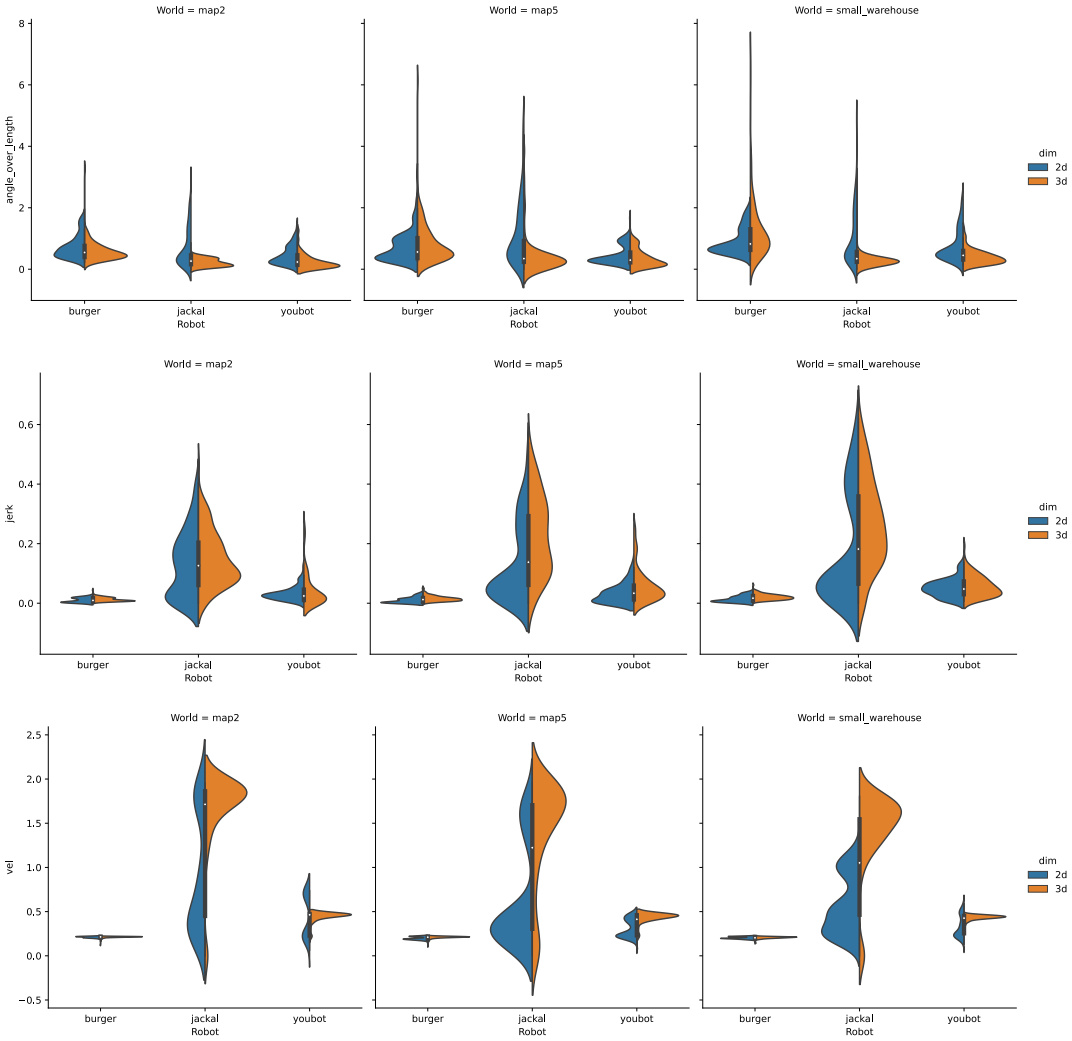


Figure 6.17: Distribution of performance parameters by map and robot type (part 1)

Comparing the distribution of the performance parameters in Figure 6.17, some differences can be observed:

- The variance of the *angle-over-length* parameter appears to be larger for the 2D simulation than for the 3D simulation.
- The *jerk* is similarly distributed in both simulation options, with the distribution following less of a normal distribution in the 2D simulation.
- Velocity shows greater variance with the *youbot* in the 2D simulation compared to its 3D counterpart.

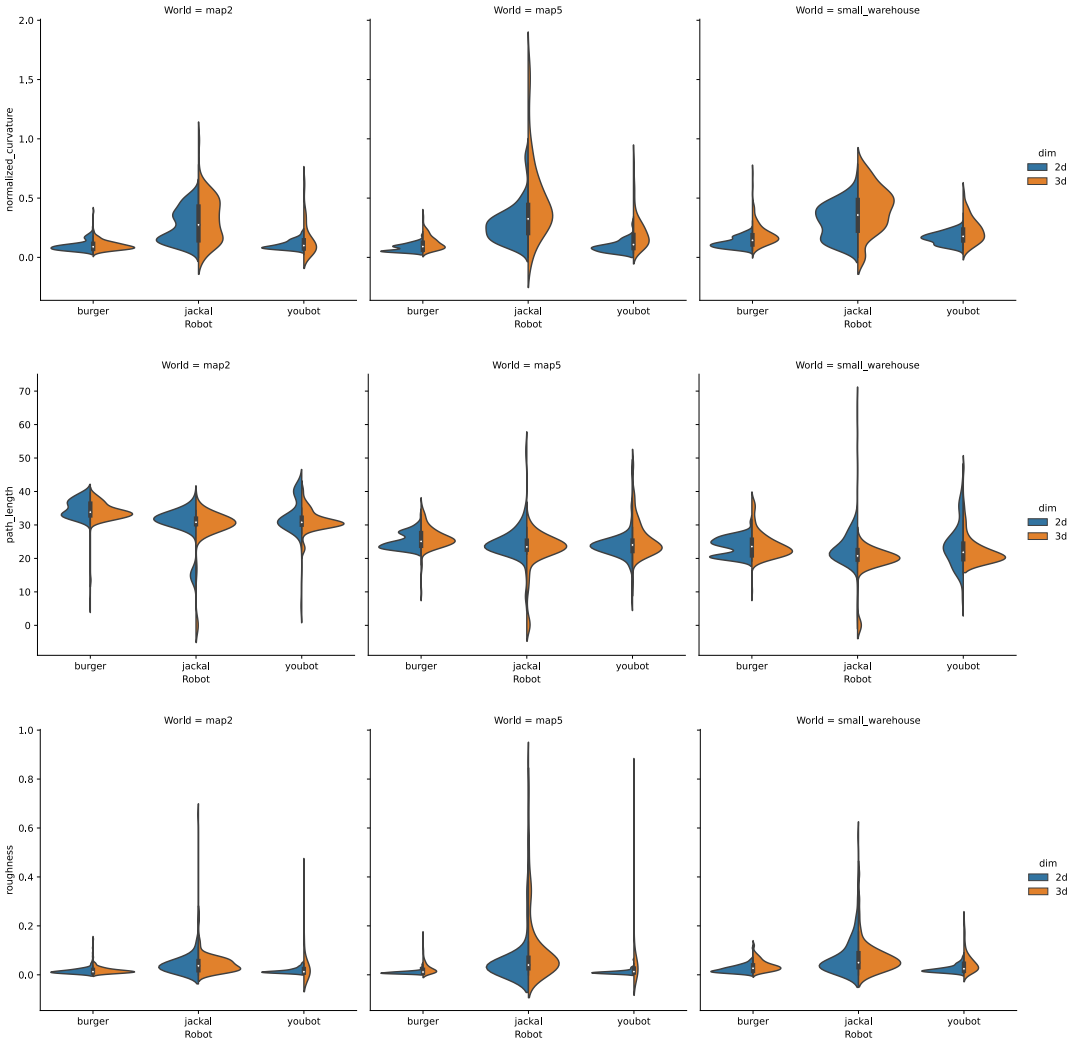


Figure 6.18: Distribution of performance parameters by map and robot type (part 2)

When looking at the Figure 6.18 several factors can be observed:

- The *path-length* tends to follow less of a normal distribution in the 2D simulation but has several peaks. These are independent of the maps used (note, however, that in this case the 5-person and 10-person scenarios are aggregated)
- The holonomic *youbot* robot has a much lower variance in 2D simulation across all maps compared to the other robots.

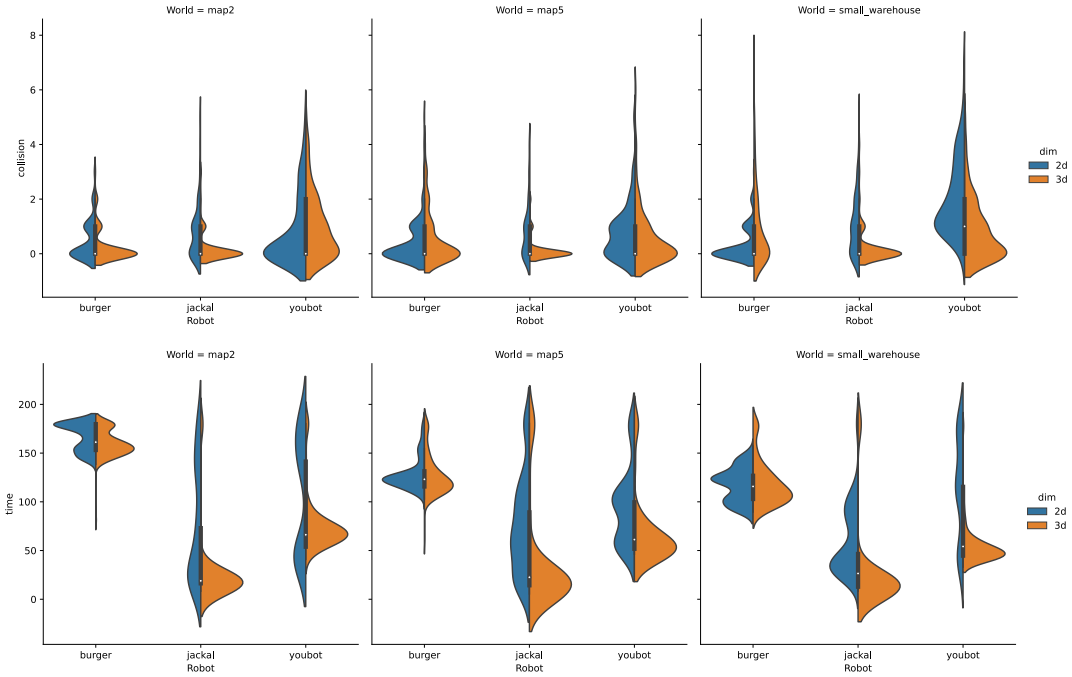


Figure 6.19: Distribution of performance parameters by map and robot type (part 3)

Comparing the performance parameters in Figure 6.19, we can observe:

- That not only the average number of collisions but also the variance of the *jackal* robot collisions is much lower,
- That the *time-to-goal* is distributed relatively consistently, with median values varying depending on the world environment.

Interpretation

In summary, it can be observed, that there are stark differences between robot types. These can in part be explained by the different action spaces of the robot. With the car-like robot *jackal* driving much smoother trajectory, or the slower *burger* robot resulting in more consistent (lower variance) performance parameters (e.g. *path-length*). Another significant influencing factor is the planner and (to a lesser extend) the simulated worlds. Much of the variance must however be attributed to the different simulation environments. The much more simple, differential drive-based, 2D robot simulation, is insufficient in simulating robot behavior more accurately. Also, the different obstacle manager used in the *Flatland*, compared to *Gazebo*, seems to have a significant (negative) impact on simulating slower robots. It can also be observed, that the data seems to be not as equally distributed in the 2D simulation. With the available data, it is not possible to explain this difference sufficiently. It is however possible that the more generic design of the 2D simulator tends to use a more discrete action space (as the simulator has much fewer moving parts, compared to the 3D counterpart).

RQ 1.2 Comparing Gazebo and Flatland influence of planner types

This section focuses on answering the following research question:

"*Do the simulation results (in terms of mean and variance) differ between the different planner types?*"

Observation data

The Figures 6.20 and 6.21 show the impact of the planner on robot performance.

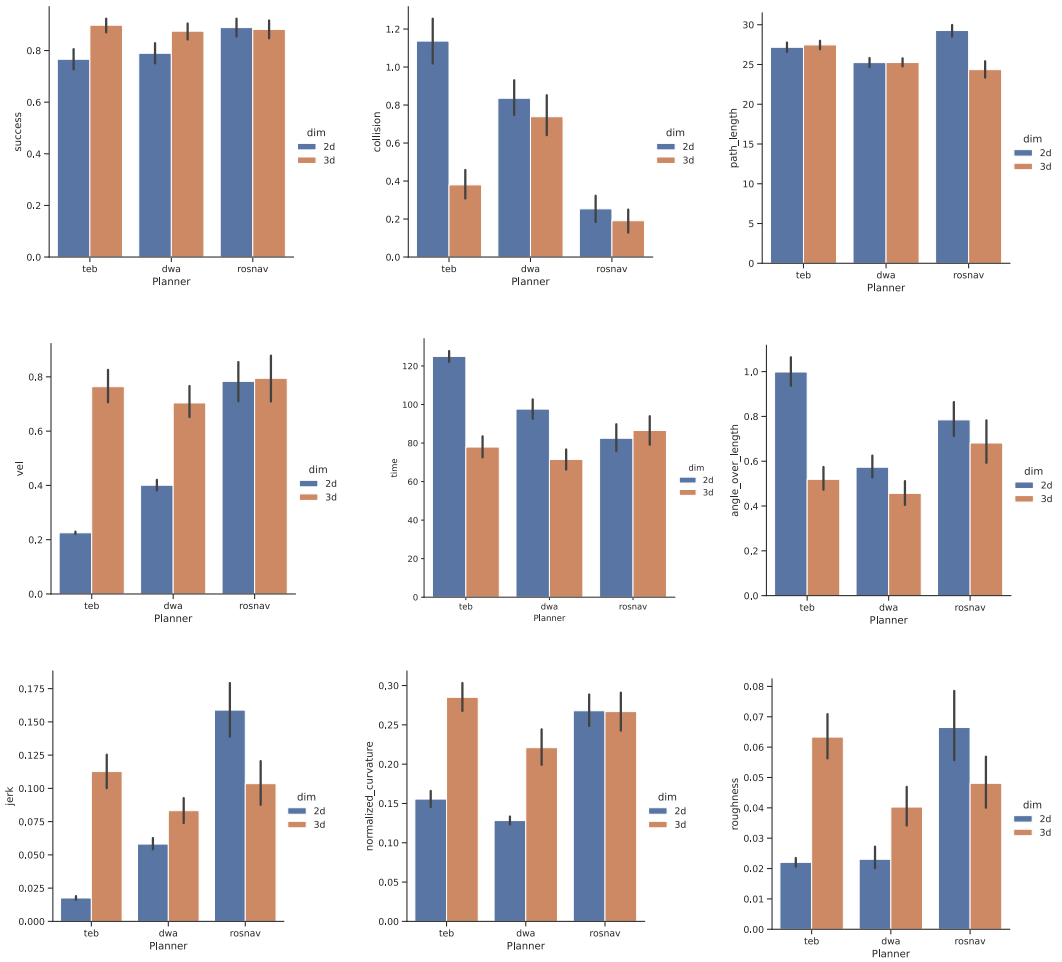


Figure 6.20: Mean and variance of performance parameters by planner type

When comparing the differences between the planner types, significant differences can be observed:

- The *rosnav* planner induces significantly fewer collisions than the conventional planner
- The *dwa* local planner creates much smoother trajectories compared to *rosnav* and *teb*

When comparing the average values between the 2D and 3D simulation planner tend to perform very differently:

- The *teb* local planner performs in the 2D simulation significantly worse than in the 3D simulation. Having a lower success rate, the highest collision probability, and curvatures as well as the lowest velocity.
- The *dwa* local planner also shows clear differences between the 2D and 3D simulation. However, this effect is less pronounced compared to its 3D counterpart.
- The *rosnav* agent tends to perform similarly in both 2D and 3D simulation, with significant differences in *jerk* and *roughness*.

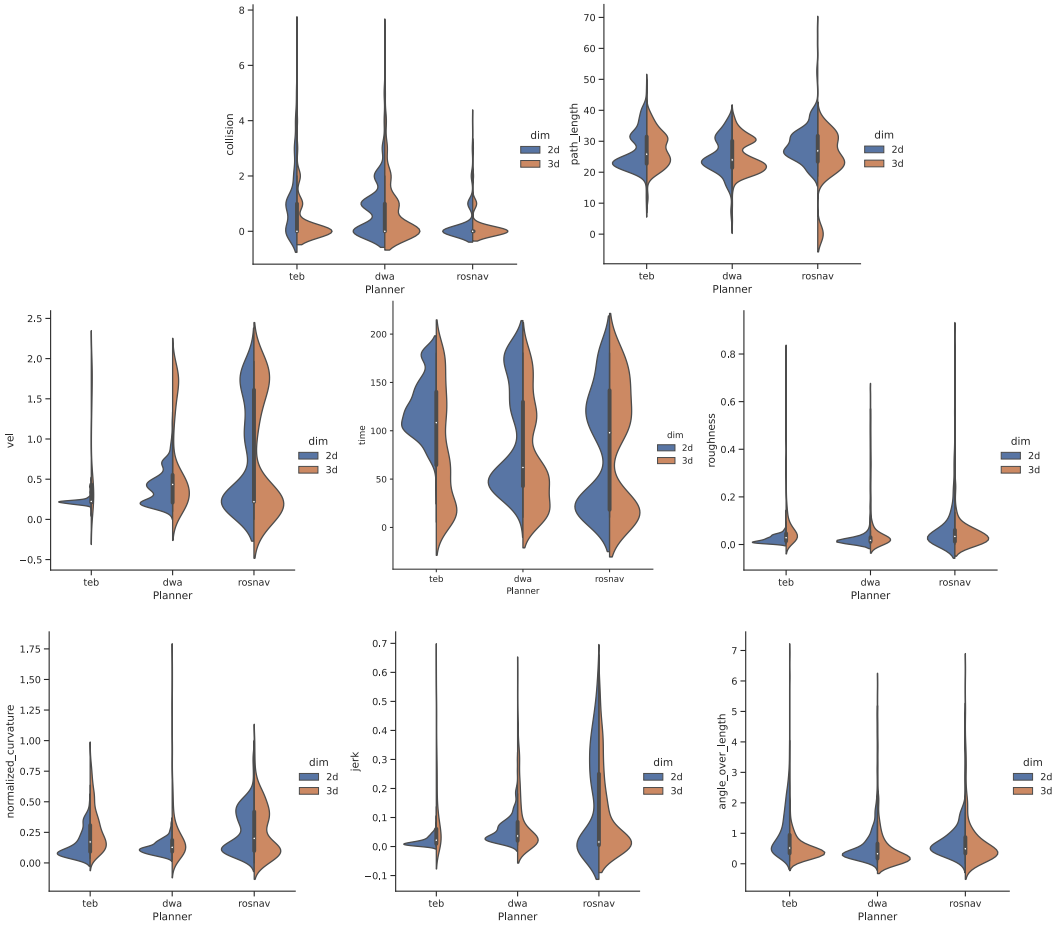


Figure 6.21: Distribution of performance parameters by planner type

When comparing the value distribution it can be observed that:

- The number of collisions seems to have a wider distribution in the 2D simulation.

- The 3D simulation follows more of a normal distribution of values, while the 2D simulation has more peaks.
- The *rosnav* agent tends to use a larger area of its solution space (it often has significantly higher variance).

The following Figures show the influence of world-types in planner performance. The influence of the robot on planner performance can be found in Figures 6.14 : 6.21.

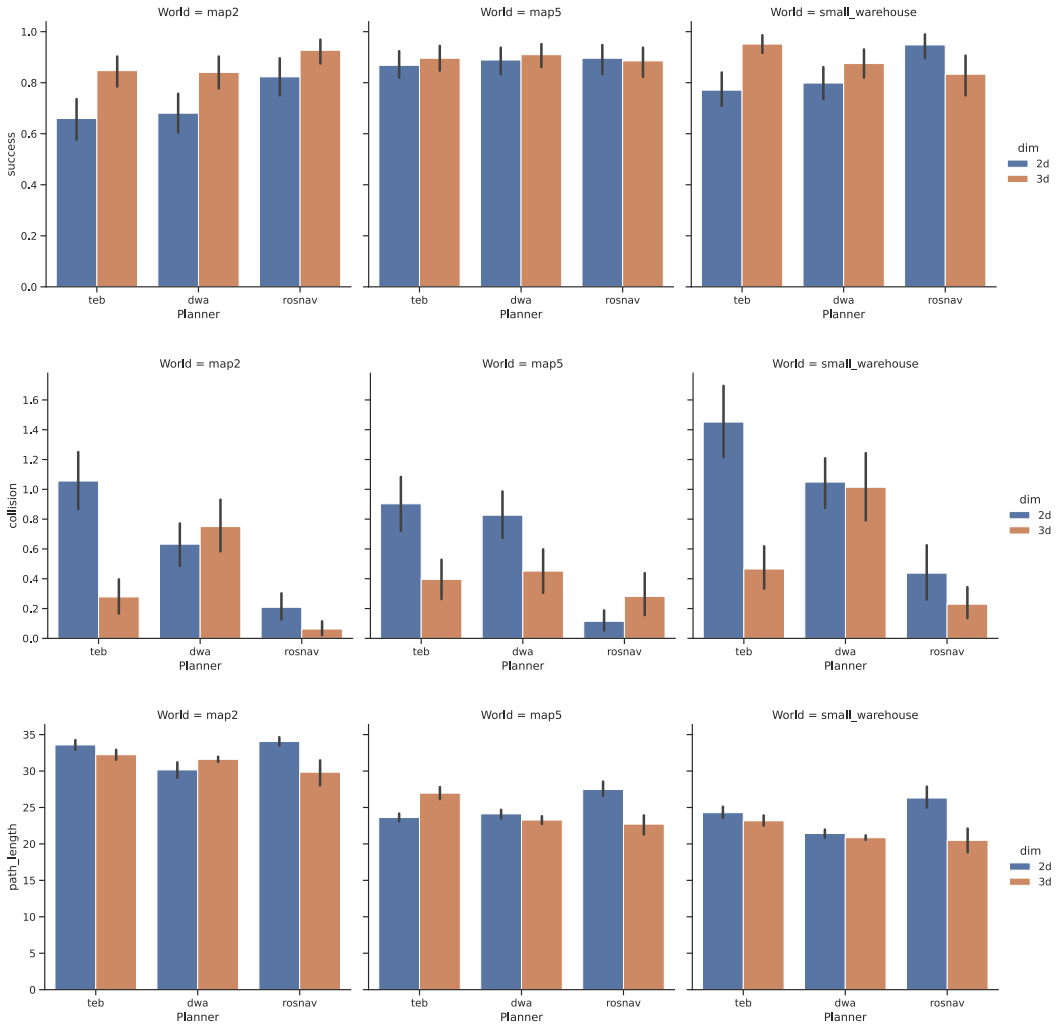


Figure 6.22: Mean and variance of performance parameters by planner and map type (part 1)

In Figure 6.22, it can be seen that there are strong differences in navigation success between the 2D and 3D repositories in the different worlds, regardless of the planner. In terms of *collisions*, the 2D and 3D repositories provide different simulation results. Both planner and map have a significant impact on planner

performance. It can be observed that *map2* leads to a longer *path-length* in all maps.

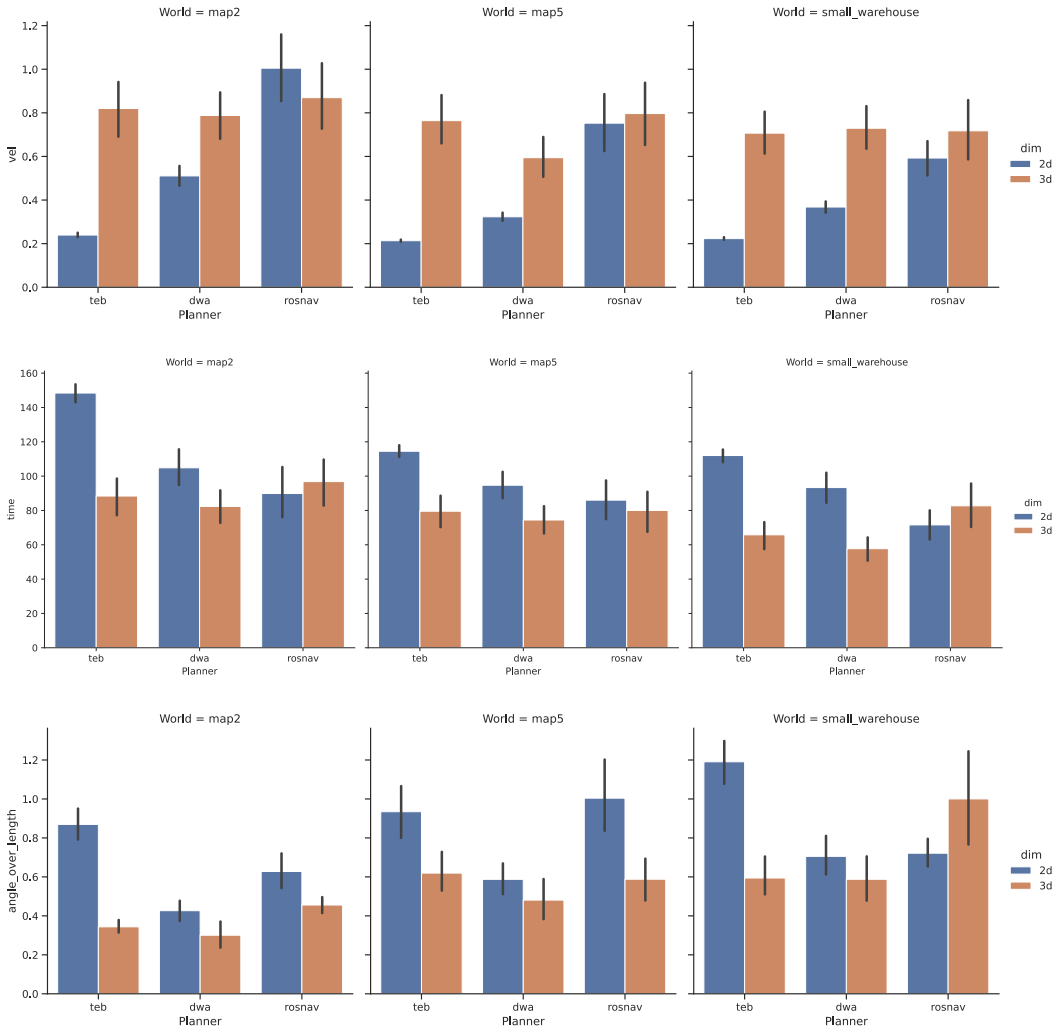


Figure 6.23: Mean and variance of performance parameters by planner and map type (part 2)

Comparing the velocities in Figure 6.23, it can be observed that robots hat a higher average velocity in *map2* compared to the other maps. It can be observed that the distribution of time and *angle-over-length* shows a similar pattern between the worlds with major differences being primarily between the planner and the 2D and 3D simulation.

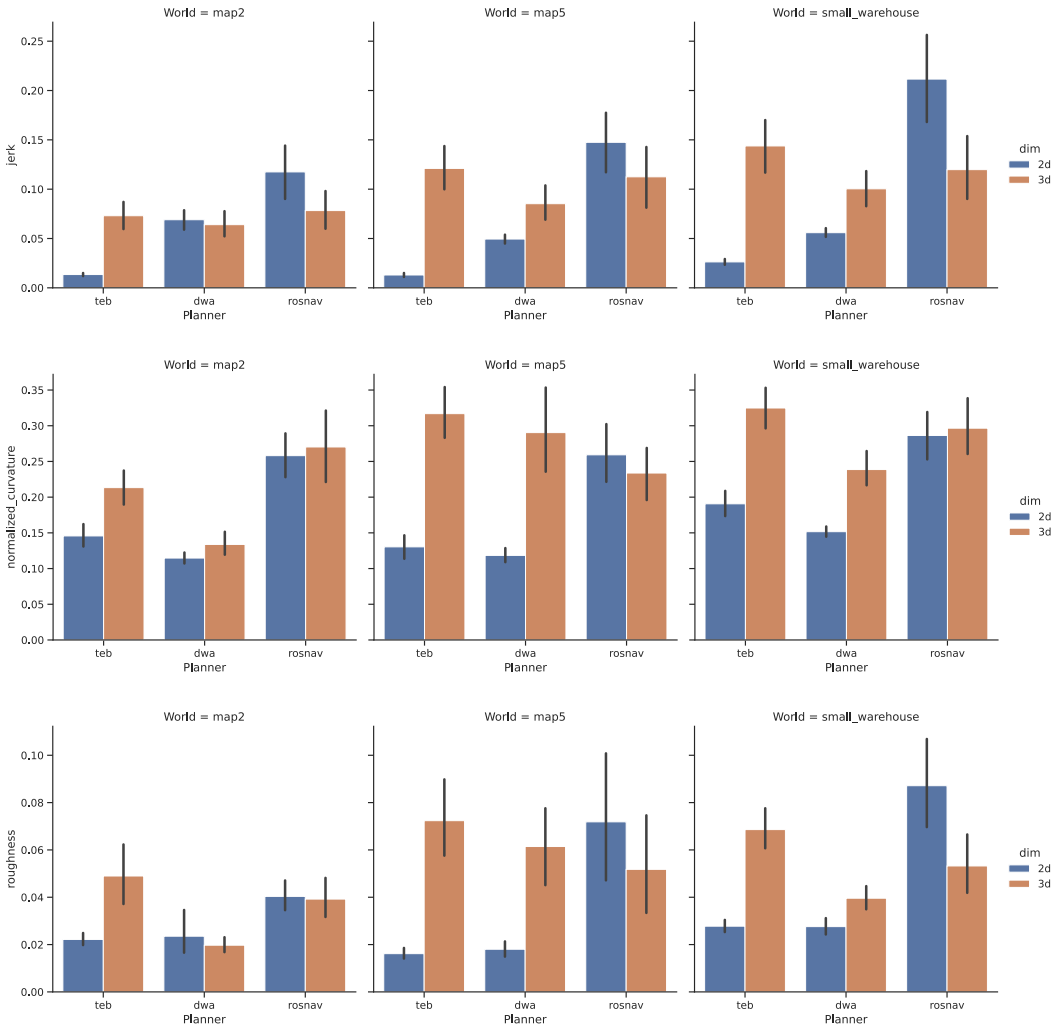


Figure 6.24: Mean and variance of performance parameters by planner and map type (part 3)

Comparing *jerk*, *normalized curvature*, and *roughness* in Figure 6.24, we observe that *map2* tends to invoke relatively low parameter values. The planners follow a similar distribution in *map2* as in *map5*, with *teb* performing significantly better in *map5*.

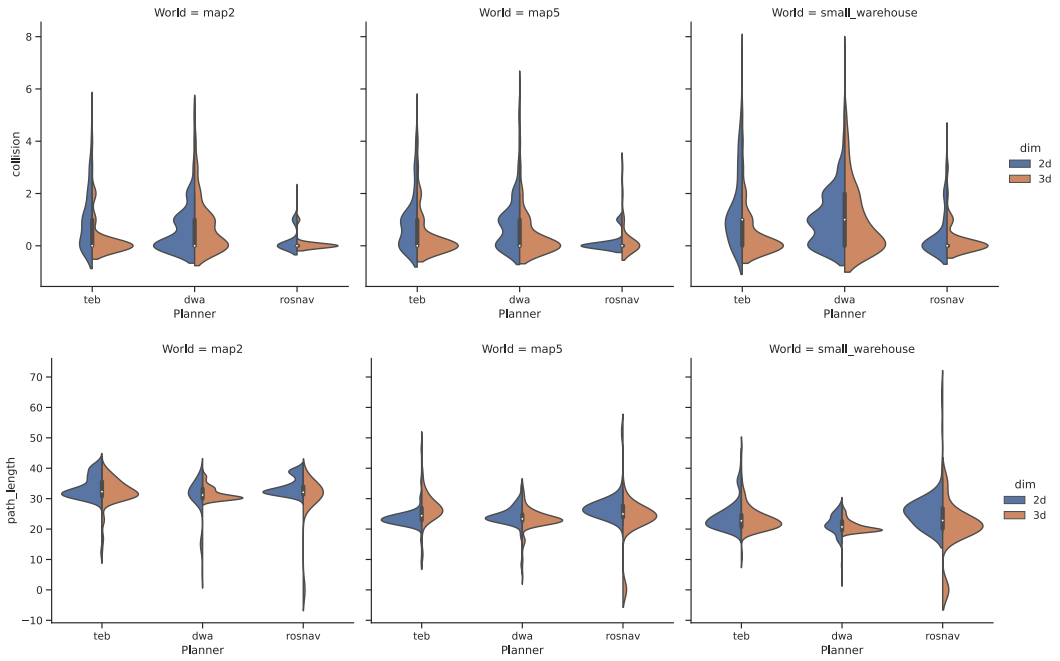


Figure 6.25: Distribution of performance parameters by planner and map type (part 1)

Comparing the distribution of collisions in Figure 6.24, it can be observed that *dwa* procures several collisions especially in the *small warehouse* map, with two collision runs even more frequent than single collision runs. In general, the planners follow the same trend, with the 2D simulation producing more collisions per instance.

In terms of *path-length*, *dwa* in the 3D simulation is characterized by a remarkably low variance. It can also be observed that the 2D simulation has does not follow a normal distribution within the same world file.

The zero values in the *path-length* variable indicate the *path-length* at timeout and show that the *rosnav* planner did not work reliably a few times, especially in the *small-warehouse* map, and did not navigate at all.

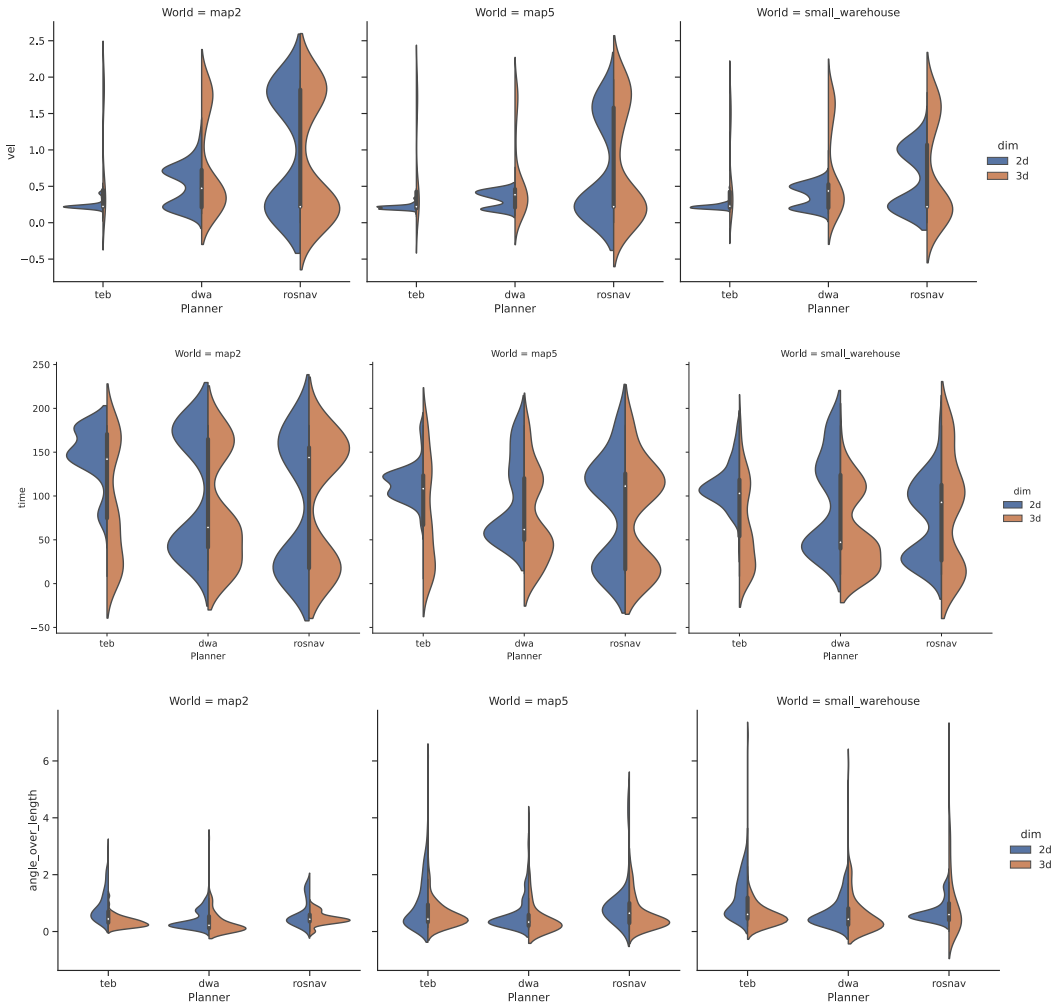


Figure 6.26: Distribution of performance parameters by planner and map type (part 2)

In Figure 6.26 it can be observed that the *time to goal* with *teb* and *dwa* follows two curves. (Probably due to the two different scenarios). The negative values shown are due to the visualization since the data set does not contain negative values. The *angle-over-length* parameter shows significant differences between the planner with the lowest distribution in the least populated *map2*.

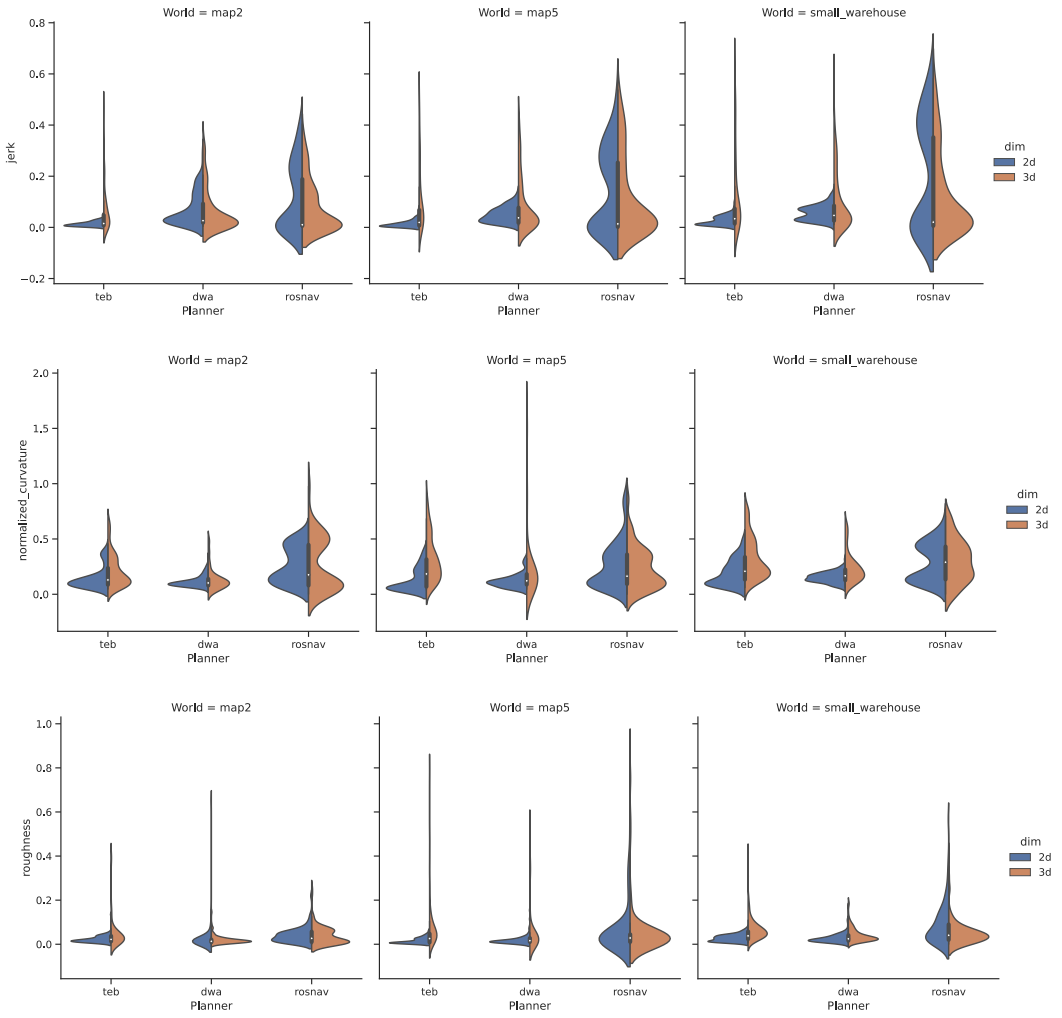


Figure 6.27: Distribution of performance parameters by planner and map type (part 3)

Comparing *jerk*, *normalized curvature*, and *roughness* in Figure 6.27 It can be observed that the *small-warehouse* map tends to produce a slightly larger variance with *teb* and *dwa* in the 2D simulation. In terms of curvature, it is observed that *dwa* tends to produce a small curvature variance across all simulated worlds.

Interpretation

In summary, it can be said that there are strong differences between the planner types in terms of robot performance. In particular, the planners *teb* and *dwa* provide completely different performance results. This difference can be explained by the fact that the 2D and 3D simulation require a different structure of the navigation stack. However, differences between the 2D and 3D simulation can also be observed for the *rosnav* planner, although the *rosnav* planner does not rely on the navigation stack. These differences could be partly due to the different obstacle managers (note, however, that in this case a constant

difference between the 2D and 3D simulation is observed for all planners). The differences are likely due to the aggregation of smaller factors, such as certain planners performing better with certain robots, or certain planners exhibiting different behavior in more populated areas.

RQ 1.3 Comparing Gazebo and Flatland influence of world types

This section focuses on answering the following research question:

"*Do simulation results (in terms of mean and variance) differ between different worlds?*"

Observation data

The following Figures 6.28 : 6.33 we show the average performance and the distribution between the map types, by number of obstacles.

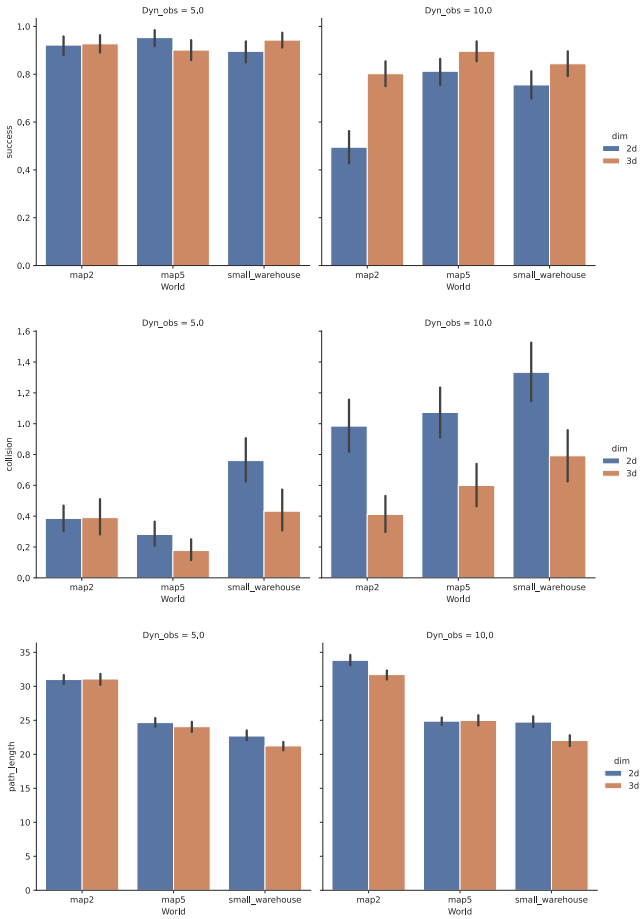


Figure 6.28: Mean and variance of performance parameters by map type and number of obstacles (part 1)

Comparing *success*, *collisions*, and *path length* in Figure 6.28, it can be

observed that *success* and *collisions* are significantly different in 10-person scenarios than in 5-person scenarios, particularly affecting the 2D simulator and *map2*. *Path-length*, on the other hand, is generally independent of map type, with *map2* leading to longer paths.

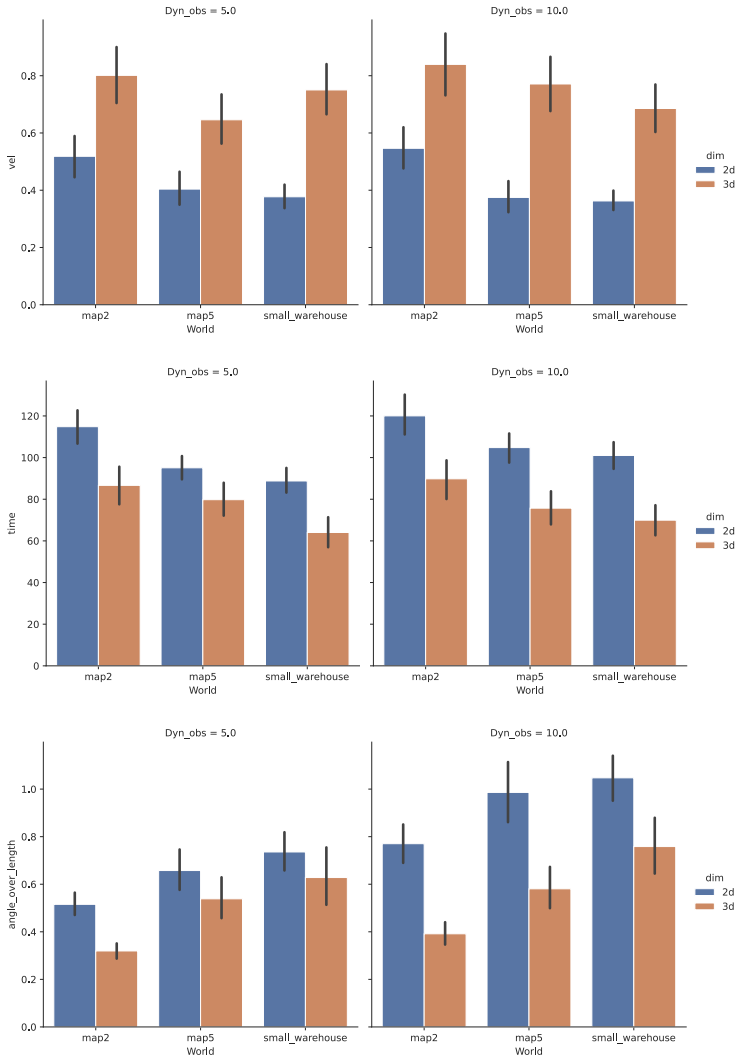


Figure 6.29: Mean and variance of performance parameters by map type and number of obstacles (part 2)

When comparing *speed*, *time*, and *angle-over-length* in Figure 6.29, it can be observed that the difference between the 2D and 3D simulation tends to be larger than the difference between the 5- and 10-person scenarios or the different worlds.

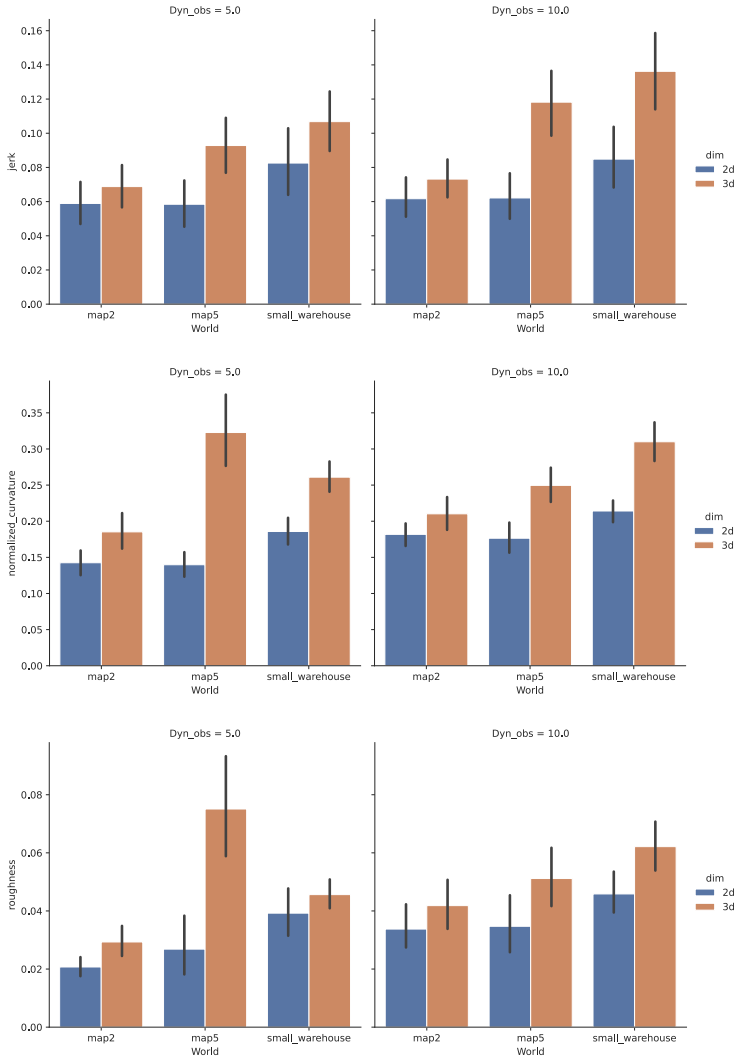


Figure 6.30: Mean and variance of performance parameters by map type and number of obstacles (part 3)

When comparing the *jerk*, *normalized curvature*, and *roughness* in Figure 6.29, it can be observed, as in the previous figure, that the difference between the 2D and 3D simulations tends to be larger than the difference between worlds or scenarios. The 2D simulation consistently shows lower values for the smoothness parameters. It can also be observed that *roughness* and *curvature* in *map5* are higher in the 5-person scenario than in the 10-person scenario.

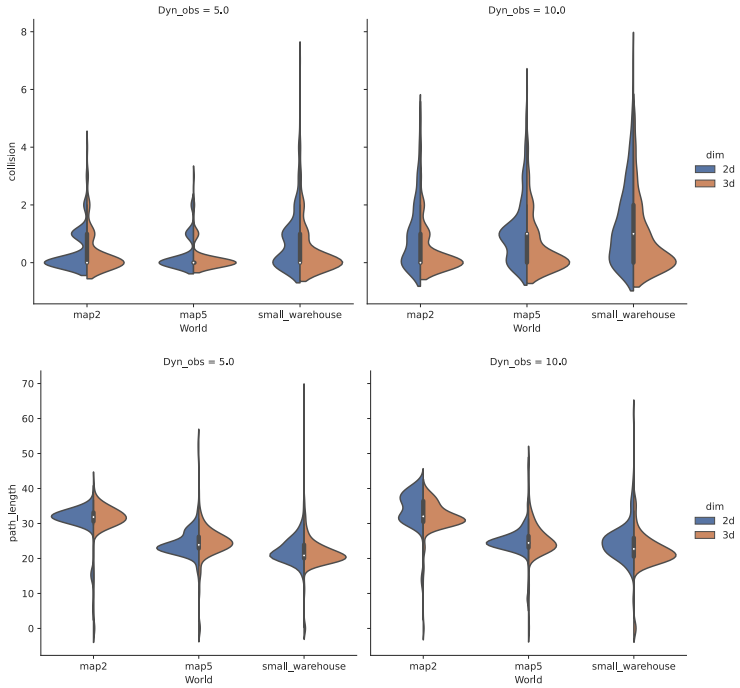


Figure 6.31: Distribution of performance parameters by map and number of obstacles (part 1)

Comparing *collisions* and *path length* in Figure 6.31, it can be observed that multiple collisions are more likely in the 10-person scenario than in the 5-person scenario. This effect is particularly pronounced in the 2D simulation. When comparing the path length, it can be observed that the distribution of values in the 2D simulation does not follow a normal distribution, even though the data is already filtered by the number of obstacles.

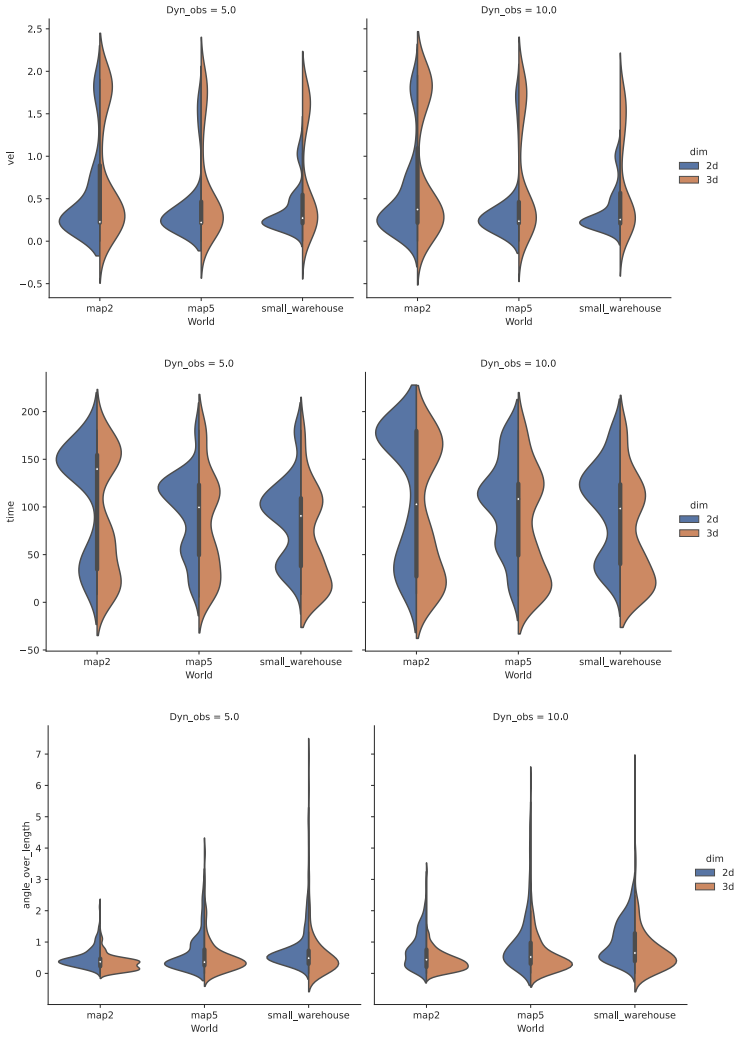


Figure 6.32: Distribution of performance parameters by map and number of obstacles (part 2)

Comparing *velocity*, *time* and *angle-over-length* in Figure 6.32 it can be observed, that the average *velocity* is not significantly effected by the number of obstacles. The *time* to goal parameter shows more large values in the 10 person scenario than in the 5 person scenario and strong differences in the value distribution between the map types. This can also be observed with the *angle-over-length* parameter.

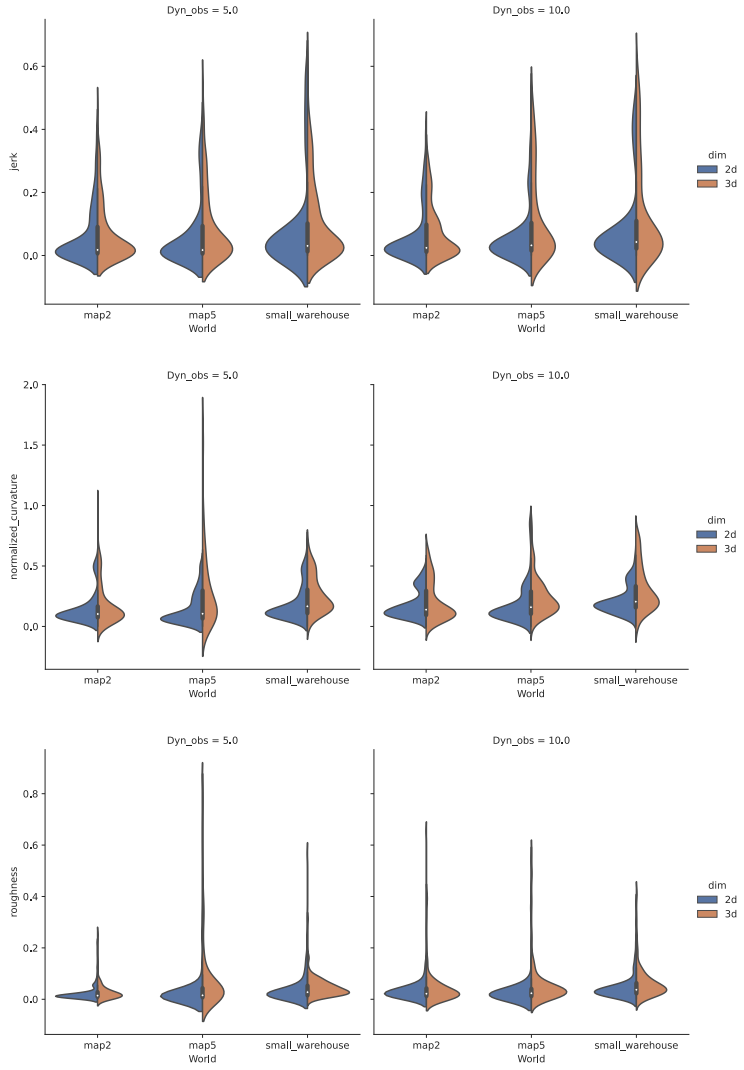


Figure 6.33: Distribution of performance parameters by map and number of obstacles (part 3)

Comparing *jerk*, *normalized-curvature* and *roughness* in Figure 6.33 significant differences between the performance parameters in the 5-person and 10-person scenarios are not observed. However, the *normalized curvature* distribution shows some more pronounced shapes in the 10-person scenario.

Interpretation

In the previous figures, an influence of certain worlds on the mean and variance can be observed. This is also the case for the number of obstacles in the simulation. However, this is only the case for certain performance parameters, namely *success*, *collisions* and *path-length*. For other parameters, the influence of the different worlds seems to be outweighed by other influencing factors. The effects of the world used and the number of obstacles tends to be more

pronounced in the 2D simulation. The similarity can be explained in part by the fact that while the worlds are very different, the navigational difficulties within the world are likely to be similar.

6.1.3 Conclusion

In summary, significant differences between the current implementation of *Flatland* and *Gazebo* can be observed in the collected data. These differences are largely due to differences in the navigation stack, as planners such as *dwa* and *teb* perform significantly different in the two simulators (it cannot necessarily be stated that a planner performs significantly worse in a particular simulator, but that different performance parameter are optimized depending on the simulator). In other words, it can be concluded that the design of the navigation stack and the planner used are much more important than the simulator chosen. Another notable finding is how little the number of obstacles and the chosen world seems to affect the performance parameters (in comparable scenarios). This suggests that the environment has a more complex than a linear relationship with robot performance (it can be presumed to have very little effect on robot performance until it reaches a certain level of difficulty, at which point it becomes very significant). However, there are some minor differences between the simulators. Most notably, the *Flatland* simulator appears to produce results with less variance than the *Gazebo* simulator, where the results appear to be much more evenly distributed.

With the above results, the second part of *RQ 1.0* (whether the differences between the simulators justify the use of the more computationally intensive 3D simulation) can be addressed. Since the impact of the simulator used on the performance results is almost negligible, the significantly better usability and simulation speed of the 2D simulation outweighs the complex and slow 3D simulation. However, as noted above, the navigation stack appears to have the largest impact on robot performance. Since manufacturers and researchers often already provide a navigation stack that can be used in the *Gazebo* simulation without customization and some cost map features are currently only available in 3D simulation, for the established conventional planners such as *teb* and *dwa*, 3D simulation seems to be the better choice, while for the *rosnav* planner that does not rely on a navigation stack, the 2D simulator is more than sufficient. In summary, the *Flatland* simulator would be the better choice if it addressed the above limitations.

6.2 The Emergency Brake System

6.2.1 Experiment Design

Since the arena suite supports a very large number of robot types (planners, etc.), only a small subset of their features was considered for this work, each representing functionalities of specific classes. The selection has been introduced in detail in section 6.1.1. For each unique combination of environment parameters (robots, planners, obstacles, worlds), a simulation scenario was prepared and run with a total of 30 resets. (30 resets were performed to be able to assume a standard distribution for the specific performance parameters). An overview of the simulation runs performed can be found in Table 6.5.

Savety System	**Planner**	**World**	**Robots and Obstacles**		
			Burger	**Jackal**	**Youbot**
Emergency Brake	dwa	map2	30	30	30
	teb		30	30	30
	rosnav		30	30	30
	dwa	map5	30	30	30
	teb		30	30	30
	rosnav		30	30	30
	dwa	small-warehouse	30	30	30
	teb		30	30	30
	rosnav		30	30	30
Adjusted Radius	dwa	map2	30	30	30
	teb		30	30	30
	rosnav		30	30	30
	dwa	map5	30	30	30
	teb		30	30	30
	rosnav		30	30	30
	dwa	small-warehouse	30	30	30
	teb		30	30	30
	rosnav		30	30	30
Standard Navigation	dwa	map2	30	30	30
	teb		30	30	30
	rosnav		30	30	30
	dwa	map5	30	30	30
	teb		30	30	30
	rosnav		30	30	30
	dwa	small-warehouse	30	30	30
	teb		30	30	30
	rosnav		30	30	30

Table 6.5: Overview of the simulation runs performed to evaluate the impact of the emergency-brake system

Table 6.5 shows the three different modes in which simulation runs were

performed, as explained in section 5.2, the navigation stack was adjusted by increasing the robot radius in the cost map. The three different modes provide an understanding of how large the impact of the emergency-brake system and the impact of the modified navigation stack (adjusted radius) are compared to an unmodified simulation (standard navigation). These setup differences are documented in the following figures using the naming convention listed in Table 6.6.

Sys	**Explanation**
embr	The emergency brake node is activated and the navigation stack is extended by the inflated robot radius.
normal	Neither the emergency brake node is activated nor the navigation stack is changed
without	The emergency brake node is not activated, but the navigation stack is extended by the inflated robot radius.

Table 6.6: Introduction to the naming convention used in the following figures.

It should be noted that it was not possible to increase the virtual robot radius with the rosnav planner since the planner does not rely on the cost map like *teb* and *dwa*. Since the developers of the *rosnav* planner ensure that the emergency-brake system works even without increasing the radius, it is not treated separately.

Since the *youbot-rosnav* agent performs significantly worse compared to the agents of the other robots, which is probably due to insufficient training, the agent is not considered in the following.

The data set created with arena evaluation contains several different metrics. For better clarity, the smoothing parameters have not been included, but they can be found in the original data-set published on GitHub[1].

6.2.2 Results

RQ 2.0 Emergency Brake Impact

This section focuses on answering the following research question:

"*Does the use of an emergency-brake system have a positive effect on navigation safety?*"

Observation data

In the following Figures 6.34 to 6.36 a comparison of the performance parameter will is being made.

[1] https://github.com/eliastreis/Master-Thesis/tree/main/data_collection/data/emergency_break

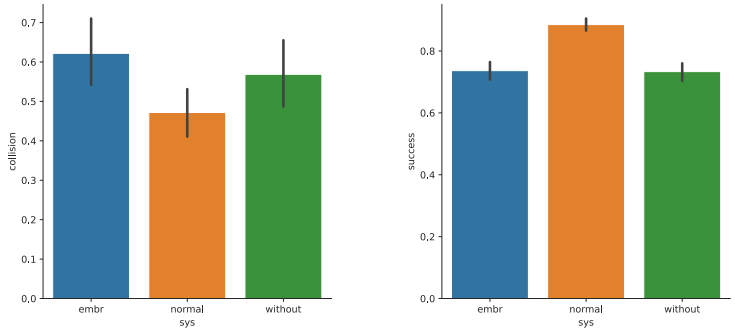


Figure 6.34: Probability of collisions and success in the case of normal configurations the safety system enabled, the navigation stack modified

When looking at Figure 6.34 it can be observed, that navigation without the emergency-brake system has a significantly positive impact on collision and success probability. A modified navigation stack, without an activated emergency-brake, slightly outperforms robots with an activated emergency brake.

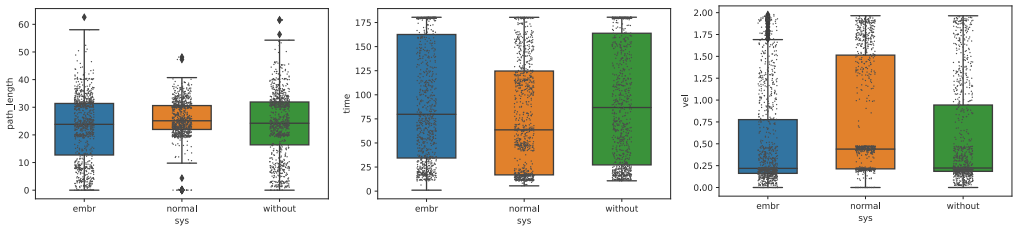


Figure 6.35: Mean and distribution of the path-length, time and velocity parameter

A look at Figure 6.35 shows a clear difference between the normal and the modified navigation stack. Even though the *path-length* is similar on average, the normal navigation stack gives more reliable results with lower variance. The time to reach the goal is also shorter and has lower variance. The speed parameter shows a higher average and variance value, as fast robots like the *jackal* utilize their speed.

Since the *rosnav* agent does not rely on the navigation stack like *teb* and *dwa* in Figure 6.36, the results of navigation with an emergency brake system are split by planner to examine if there are significant differences between planners.

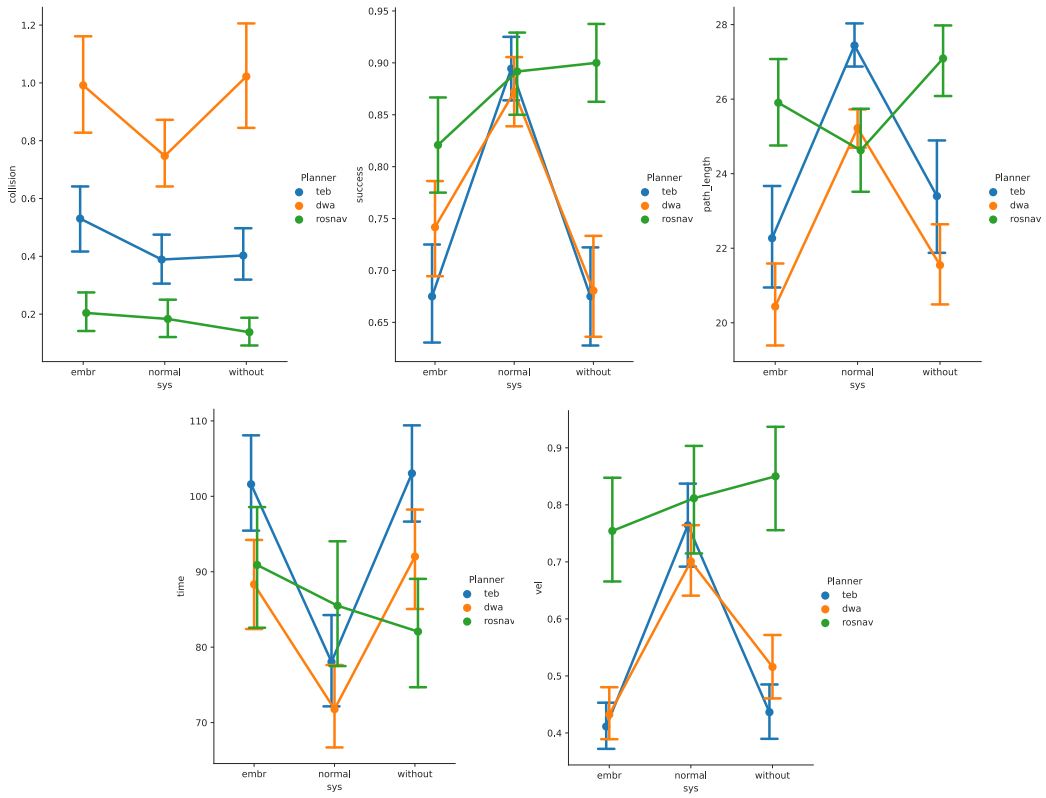


Figure 6.36: Mean and variance of performance parameters by planners

Looking at the performance of the different planners in Figure 6.36, a clear ranking of planner performance can be seen in most metrics, with *rosnav* clearly at the top, followed by *teb* and *dwa*, which have the worst performance in almost all cases. It can also be observed that using an emergency-brake system and changing the route have an impact on the *rosnav* agent performance. This leads, for example, to a longer *path-length*. It is also observed that the *dwa* planner is much more affected by the change of the navigation stack compared to other planning approaches. In addition, teb and *rosnav* show significantly worse performance in terms of collisions when the emergency-brake system is activated.

Interpretation

Based on the results of the previous section, it can be stated with certainty that the implementation of an emergency brake system in its current design on mobile robots based on local planners does not have a positive impact on navigation safety.

These findings are likely due to several factors:

- The implementation of safety zone 2 in particular (which triggers the emergency brake) significantly limits the robot's possible navigation space

and in many cases leaves the robot with only the option of triggering the emergency brake, even though it would have more solutions if it were implemented without the emergency brake.
- Local planners have a build-in collision avoidance system, such as changing or even reversing the trajectory to avoid collisions. This much more flexible approach has a larger solution space and can react more appropriately to obstacles.
- In the simulation, the *actors* (obstacles) follow a predefined trajectory and will not try to avoid collisions with the robot. A sudden stop of the robot will therefore not reduce the probability of a collision, since the obstacle will continue its trajectory regardless of the robot's behavior. An evasive maneuver, therefore, seems to be more promising in this situation
- The parameter of the cost map and local planner show an interdependency. An increase in the virtual robot radius must therefore be accompanied by a change in several other parameters. Great care must be taken in finding the best possible combination of parameters for the planner. The changes in the navigation stack may have compromised the careful parameter balance created by the manufacturer, resulting in reduced planner performance.

RQ 2.1 Safety Tiers

This section focuses on answering the following research question:

"*Within which safety tier perform our simulated robots by robot-type, world, planner, and obstacles?*"

The following figures show the collision probability by planner and robot. These data points are generated by aggregating the performance across all 3000, 3D simulations (without activating the emergency brake system or changing the navigation stack), aggregating (intentionally) influencing parameters such as the number of obstacles or the world environment.

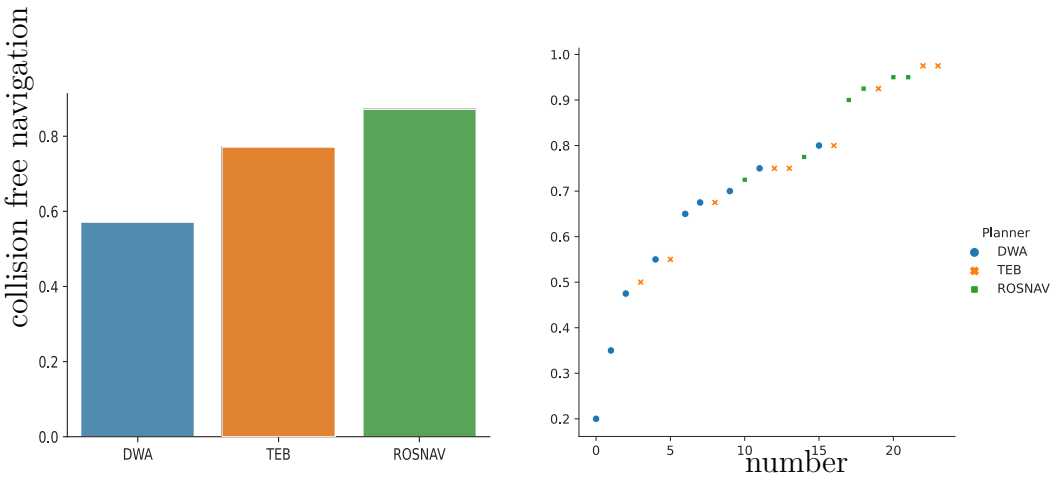


Figure 6.37: Average probability of a collision-free navigation by planner type

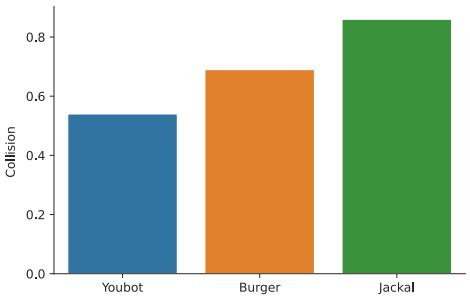


Figure 6.38: Average probability of a collision-free navigation by robot type

Observations

The *rosnav* planner navigates with the lowest collision probability, with a probability of collision-free navigation of over 80 percent. Similar behavior can be observed with the *jackal* robot, which has the lowest collision probability. As can be seen in Figure 6.37 in certain scenarios even the well-performing *rosnav* tends to have a high collision probability.

6.2.3 Conclusion

In summary, the implementation of the emergency brake system has a negative impact on robot performance parameters and does not lead to a reduced collision probability. This is likely since the emergency brake system overrides the planner's inherent safety functionality which is equipped with more features compared to the emergency brake system. It must however be noted, that these results, describe only the situation within the *Gazebo* simulation, transferring the concepts to real-world applications are likely to yield different results, as

real-live actors will attempt to avoid collisions, were as simulated agents do not show such behavior.

Comparing the collision probability of the local planner across all simulation runs, the *rosnav* agent shows the lowest collision probability, with the highest collision probability being with the *dwa* planner. Comparing the collision probability of mobile robots, the fast car-like *jackal* robot shows by far the lowest collision probability and the holonomic-*youbot* robot shows the highest collision probability, proving that robot design has a significant impact on robot performance (in the same range as the local planner).

7 Conclusion

This work describes the implementation of the arena-benchmarking repositories by combining several independent repositories into a repository suite with tested and well-documented interactions between the different components. This platform allows custom worlds and scenarios to be created using *arena-tools* and transferred to the 2D *Flatland*-simulation repository *arena-rosnav*, or the 3D *Gazebo* simulation repository *arena-rosnav-3D*. The robot's performance in the simulated world and scenario can be evaluated using the *arena-evaluation* repository. Which evaluates the simulation runs via various metrics such as *path-length* or *smoothness*.

Another contribution of this work is the implementation of an emergency stop safety function. Which regulates the robot's command velocity when the robot is endangered to collide with an obstacle, This is done by either reducing its velocity if the obstacle is found within a specific distance range or stopping the robot when the obstacle reaches a predefined safety parameter. In addition, this work proposes changes to the navigation stack that allow planners not created with a built-in navigation stack to use the emergency brake system.

The created infrastructure is utilized in a second step to answer several research questions, namely comparing simulation results in the 2D and 3D simulators to evaluate simulation differences and testing the impact of an emergency-brake system on mobile robot navigation safety.

The results of the comparison of the *Flatland* and *Gazebo* simulations can be summarized as indicating that the simulator has a minor influence on the performance parameters of the robot. This influence can be considered negligible since, for example, the selected local planner has a disproportionately larger impact on robot performance. Although the *Flatland* simulator outperforms the *Gazebo* simulator in terms of simplicity, ease of use and simulation speed, the *Gazebo* simulator can currently still be considered the simulator of choice since, due to its popularity, planning approaches and robots are often built for (or with) the *Gazebo* simulator and require significant knowledge and effort to be used with comparable results in the *Flatland* simulator.

The results of the comparison between the navigation safety of robots with and without an emergency-brake system clearly show that the system has no

positive influence on the navigation safety of the mobile robots tested. This can be attributed to several reasons. Most notably, it overrides the own more powerful obstacle avoidance approaches of the local planner.

8 Future Work

Future work can be broken down into two components, firstly improvements to the benchmarking platform and secondly utilizing the platform to answer some relevant research questions.

8.1 Benchmarking Platform

The benchmarking platform has four major components, the following section will list for every component open task and possible enhancements.

8.1.1 Arena-Tools

Arena-tools can be used for three main functions, firstly the creation of custom maps and worlds, secondly the creation of custom scenarios and thirdly the creation of *Flatland* obstacles.

Creating maps and worlds with *arena-tools* has several limitations.

1. It is not possible to preview the randomly created maps. A sample world is displayed, but not the map that will be saved in the *simulator-setup* folder.
2. *Arena-tools* can not be used to create individual worlds. Drawing walls etc is not possible.
3. When creating worlds, only meta-parameters such as the corridor width can be set. However, the resulting worlds can differ significantly in difficulty. For an inexperienced user, it can therefore be quite difficult to recognize how difficult a particular world is for the robot and whether the created paths are feasible.

Creating scenarios with *arena-tools* has a major shortcoming. The scenarios are created in pedsim format. Since using pedsim in scenario mode is not advisable in either 2D or 3D simulation, conversion scripts have been created to transfer the scenarios from pedsim format to the format required by the simulation repository's obstacle manager. This step is documented in the repository but is not intuitive. Future work could therefore consist of including a reliable obstacle manager that can be used to manage obstacles in 2D and 3D simulation, using input files in the same format.

8.1.2 Arena-Rosnav

This repository includes many different functionalities. Some major shortcomings and resulting future work options are listed below:

- The repository supports two obstacle managers, both of which have their respective strengths and weaknesses. *Pedsim* manages obstacles to create human-like trajectories. However, it only works well in wide-open spaces. Also, obstacles tend to freeze after a certain runtime for reasons not yet understood. The native obstacle manager works more reliably compared to *Pedsim*, but so far only supports certain circular obstacles that do not show leg movements. In addition, the velocity of the obstacles is not constant but can be affected by the length of the obstacle path. Therefore, the development of a reliable obstacle manager that can be used in both 2D and 3D repositories would be helpful to obtain more comparable and reliable results.
- So far, the repository only supports mobile bases with differential drive. The differential drive plugin could be extended to also support holonomic robots and car-like behavior. (Car-like behavior can be approximated with the differential drive plugin, but the accuracy of this approximation in *Flatland* has not yet been tested).
- The simulation factor of the repository has so far been constant and independent of the simulation. This causes problems when using the simulation in computationally intensive scenarios. For example, simulating the fast *rto* or *jackal* robot can lead to problems as it causes timeouts because the average computer is not able to keep up with the continuous computational load. By adding the ability to dynamically alter the simulation speed depending on the computational load, these limitations could be addressed.
- When integrating new/additional navigation algorithms, the navigation stack cannot be implemented directly by the manufacturer without some additional adjustments. For example, the voxel grid plugin can only be used in the 3D simulation. However, these differences are not documented, which is a hurdle for the integration of new navigation algorithms or robots.

8.1.3 Arena-Rosnav-3D

Possible improvements to the 3D repository were part of a recent project report [12] submitted to the chair. The following section extends this list and focuses more on the current limitations of the repository.

- By supporting the use of the mesh format, *Gazebo* interfaces with 3D modeling software such as *Blender*. This interface is sparsely documented, although recent developments in 3D modeling have the potential to greatly

enrich the use of the popular *Gazebo* simulator. For example, creating worlds, models, and actors with Blender is much more efficient than with the native *Gazebo* process. It also brings additional benefits, such as compatibility between *Gazebo* versions, better documentation, etc.
- Although steps have been taken to enable training of drl agents in the *arena-rosnav-3D* repository, this functionality is not yet fully usable. Training agents in a 3D environment could lead to even better robot performance in real-world scenarios due to the more realistic environmental simulation.
- *Pedsim* does not yet support the functionality to register interactive obstacles (currently obstacles have to be defined at startup to be registered in *Pedsim*). For this reason, static obstacles in the random world generator (*outdoor* mode) could not be registered in *Pedsim* so far, causing *Pedsim* agents to pass through static obstacles.
- Several of the algorithms are only supported with certain robots. Some like *mpc* also tend to work not very reliable. These problems could be addressed, in some cases with little effort. A list, featuring which planner works in which scenarios have been added to the repository[1].
- The data generator branch[2] developed in this project has some limitations in its current implementation. When starting the *Gazebo* simulation, the simulator loads the current configuration of the robot type and the number of dynamic obstacles. Changing parameters in the setup file would therefore only lead to changes after a restart of the simulation. This restart behavior has not been implemented yet. Therefore, research questions like the influence of the laser range on navigation success can so far not be answered by the data, generated by the data-generator mode.

8.1.4 Arena-Evaluation

The newly created *arena-evaluation* repository will very likely be of great interest to the robot navigation research community, as it involves the practical implementation of measures that have only been proposed theoretically in previous research. However, it has some shortcomings:

- Recorded data tends to contain some unreasonable outliers. This could be due to timeouts in the simulation. Even though most of the common issues have been resolved, recorded data often still contains unreasonable values. These issues could likely be resolved by adding a wait-for message/service in the simulation before actually calling the message/service. However, until now, the workflow has been to check the data for inappropriate outliers before calculating the evaluation measures.

[1] https://github.com/ignc-research/arena-rosnav-3D/blob/main/docs/Usage.mdnavigation
[2] https://github.com/ignc-research/arena-rosnav-3D/tree/data-generator

- So far the repository contains only very basic documentation. Adding more examples and tutorials might be helpful for new users.
- Implementation of complexity measures, such as evaluating scenarios and worlds for difficulty, is included in the repository but is not yet integrated into the standard workflow.
- Many of the features have only been tested by one or two users as they were still under development. For reliable public use, additional testing would be helpful to verify that the components work reliably.
- There appears to be a strong correlation between many performance parameters such as *jerk* and *roughness*. If this is the case, it may be advisable to reduce the total number of predicted performance parameters by removing duplicates as this would increase the understandability of the results.
- The *arena-evaluation* script uses only a small part of the recorded data so far. For example, to calculate the average speed per run, the average and variance data are recorded. For the later evaluation, only the average value is taken into account and the variance is not considered (although a variance analysis would be particularly relevant for an accurate description of the robot's performance).

8.1.5 Miscellaneous

The following list contains possible extensions of the benchmarking platform that cannot be assigned to specific elements of the benchmarking platform.

- Popular 3D robot simulation platforms like *Gazebo* and *V-REP* are almost 20 years old. However, with the rise of video games, there have been massive improvements in 3D simulation in terms of functionality and usability. There have been some steps to adapt platforms like *Unity* to the needs of robot navigation. However, these steps are only in their infancy. The use of these platforms could solve several problems with current 3D simulation platforms, such as better and simplified usability, more coherent and comprehensive obstacle management, better documentation of the platform, etc.
- When resetting the robot after it has reached its navigation goal. The robot starts navigating again from its start position to its goal position. However, the obstacles are not reset to their original starting position but continue on their current trajectory. Therefore, especially in scenario mode, the environments encountered after each reset are different and not directly comparable. (Although this could be understood as a feature, it must be assumed that the number of resets required to obtain reliable results is much higher.)
- This work provides an overview of the differences between the *Flatland* and *Gazebo* simulators, but it is unclear which simulator better simulates

real-world robot behavior. Therefore, it would be helpful to quantify the differences between the 3D *Gazebo* simulation and real-life robot behavior. The navigation stack from the *arena-rosnav-3D* repository could be used to control real robots with only minor modifications.

8.2 Future Research Questions

The following section provides a non-comprehensive list of research questions that could be answered using the benchmarking platform.

- Is there a significant difference in performance between robot types and if so, does it justify the significantly increased cost between e.g. a car-like and holonomic robots.

(This comparison could be made based on the data-set collected for this work, for comparing holonomic-rosnav agents; however, a holonomic-drive plugin must be created for the *Flatland* simulation.)

- What is the impact of environmental factors such as the number of actors or the number of wires on the Gazebo simulation speed?
- How is the 2D navigation stack different from the 3D or real navigation stack? What are the practical reasons for these differences, can these differences be resolved?
- What is the impact of robot speed on navigation safety?
- Which ratio between the robot and the obstacle speed leads to the lowest collision probability?

Bibliography

[1] *2D Robot Simulator*. 2022. URL: https://www.chunshangli.com/2017/08/31/2D-robot-simulator.html (visited on 03/19/2022) (cit. on p. 8).

[2] *blender*. 2020. URL: https://www.blender.org/ (visited on 03/01/2022) (cit. on p. 5).

[3] Hugo Caselles-Dupré et al. "Flatland: a lightweight first-person 2-d environment for reinforcement learning." In: *arXiv preprint arXiv:1809.00510* (2018) (cit. on pp. 3, 7).

[4] Jack Collins et al. "A review of physics simulators for robotic applications." In: *IEEE Access* (2022) (cit. on p. 3).

[5] Dieter Fox. "Kld-sampling: Adaptive particle filters and mobile robot localization." In: *Advances in Neural Information Processing Systems (NIPS)* (2002) (cit. on p. 12).

[6] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. "The dynamic window approach to collision avoidance." In: *IEEE Robotics & Automation Magazine* 4.1 (1997), pp. 23–33 (cit. on p. 15).

[7] *Gazebo Tutorials*. 2014. URL: https://gazebosim.org/tutorials (visited on 03/19/2022) (cit. on p. 5).

[8] Ronja Guldenring et al. "Learning local planners for human-aware navigation in indoor environments." In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 6053–6060 (cit. on pp. 3, 7).

[9] Eric Heiden et al. "Bench-MR: A motion planning benchmark for wheeled mobile robots." In: *IEEE Robotics and Automation Letters* 6.3 (2022), pp. 4536–4543 (cit. on p. 4).

[10] Sergi Hernàndez and Fernando Herrero. "Autonomous navigation framework for a car-like robot." In: (2015) (cit. on p. 14).

[11] Hyeong Ryeol Kam et al. "Rviz: a toolkit for real domain data visualization." In: *Telecommunication Systems* 60.2 (2015), pp. 337–345 (cit. on p. 8).

[12] J. Kmiecik, N. Kmiecik, and E. Treis. "Project Distributed Control Systems Benchmarking Obstacle Avoidance Algorithms on Different Robotic Systems in 3D." In: (2022), pp. 1–37 (cit. on p. 75).

[13] Nathan Koenig and Andrew Howard. “Design and use paradigms for gazebo, an open-source multi-robot simulator.” In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 3. IEEE. 2004, pp. 2149–2154 (cit. on p. 3).

[14] Anis Koubâa et al. *Robot Operating System (ROS)*. Vol. 1. Springer, 2017 (cit. on p. 12).

[15] Dana Kulić and Elizabeth Croft. “Pre-collision safety strategies for human-robot interaction.” In: *Autonomous Robots* 22.2 (2007), pp. 149–164 (cit. on p. 16).

[16] Alexandre Lampe and Raja Chatila. “Performance measure for the evaluation of mobile robot autonomy.” In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE. 2006, pp. 4057–4062 (cit. on p. 4).

[17] *Mobiler Manipulator für Forschung und Ausbildung*. 2022. URL: `https://www.all-electronics.de/elektronik-fertigung/mobiler-manipulator-fuer-forschung-und-ausbildung.html` (visited on 03/01/2022) (cit. on p. 31).

[18] Mark Moll, Ioan A Sucan, and Lydia E Kavraki. “Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization.” In: *IEEE Robotics & Automation Magazine* 22.3 (2015), pp. 96–102 (cit. on p. 4).

[19] Akos Nagy, Gábor Csorvási, and Domokos Kiss. “Path planning and control of differential and car-like robots in narrow environments.” In: *2015 IEEE 13th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*. IEEE. 2015, pp. 103–108 (cit. on p. 14).

[20] *Navigation*. 2022. URL: `http://wiki.ros.org/navigation` (visited on 03/19/2022) (cit. on p. 8).

[21] Lucas Nogueira. “Comparative analysis between gazebo and v-rep robotic simulators.” In: *Seminario Interno de Cognicao Artificial-SICA* 2014.5 (2014), p. 2 (cit. on p. 3).

[22] *Offizielle ROS Entwicklungsplattformen*. 2021. URL: `https://www.robotis.at/turtlebot3/` (visited on 11/11/2021) (cit. on p. 31).

[23] *Offizielle ROS Entwicklungsplattformen*. 2022. URL: `https://www.generationrobots.com/de/402144-unbemanntes-landfahrzeug-jackal-ugv.html/` (visited on 03/01/2022) (cit. on p. 31).

[24] Franklin Okoli et al. “Cable-driven parallel robot simulation using gazebo and ros.” In: *ROMANSY 22–Robot Design, Dynamics and Control*. Springer, 2019, pp. 288–295 (cit. on p. 7).

[25] Daniel Perille et al. “Benchmarking metric ground navigation.” In: *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE. 2020, pp. 116–121 (cit. on p. 3).

[26] Lenka Pitonakova et al. “Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators.” In: *Annual Conference Towards Autonomous Robotic Systems*. Springer. 2018, pp. 357–368 (cit. on p. 3).

[27] Sebastian Pütz, Jorge Santos Simón, and Joachim Hertzberg. “Move base flex a highly flexible navigation framework for mobile robots.” In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 3416–3421 (cit. on p. 12).

[28] I Rañó and J Minguez. “Steps towards the automatic evaluation of robot obstacle avoidance algorithms.” In: *Proc. of workshop of benchmarking in robotics, in the IEEE/RSJ int. conf. on intelligent robots and systems (IROS)*. Vol. 88. 2006, pp. 90–91 (cit. on p. 4).

[29] Christoph Rösmann. “Difference between DWA and TEB local planners.” In: () (cit. on p. 15).

[30] *SDFormat*. 2020. URL: `http://sdformat.org/` (visited on 03/01/2022) (cit. on p. 5).

[31] Aaron Staranowicz and Gian Luca Mariottini. “A survey and comparison of commercial and open-source robotic simulator software.” In: *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*. 2011, pp. 1–8 (cit. on p. 3).

[32] *Stopping moving robot during navigation*. 2021. URL: `https://answers.ros.org/question/193998/stoppingmoving-a-robot-during-navigation/` (visited on 01/01/2022) (cit. on p. 17).

[33] Nathan R Sturtevant. “Benchmarks for grid-based pathfinding.” In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.2 (2012), pp. 144–148 (cit. on p. 4).

[34] *Teb Local Planner Tuning Guide*. 2022. URL: `https://mowito-navstack.readthedocs.io/en/latest/step_5c.html` (visited on 08/19/2022) (cit. on p. 15).

[35] *TEB Tutorials*. 2022. URL: `http://wiki.ros.org/teb_local_planner/Tutorials/Setup%20and%20test%20Optimization` (visited on 08/19/2022) (cit. on p. 15).

[36] Jian Wen et al. “MRPB 1.0: A Unified Benchmark for the Evaluation of Mobile Robot Local Planning Approaches.” In: *2022 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 8238–8244 (cit. on pp. 3, 18).

[37] Kaiyu Zheng. “Ros navigation tuning guide.” In: *Robot Operating System (ROS)*. Springer, 2021, pp. 197–226 (cit. on p. 15).