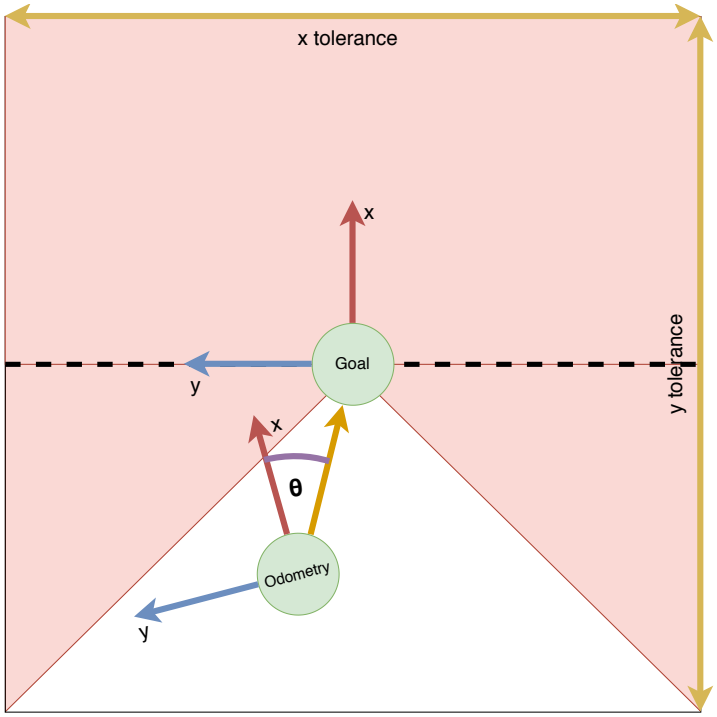
Lucerne University of Applied Sciences and Arts



School of Information Technology

Master Thesis

Strategy for an Autonomous Behavior that Guarantees a Qualitative Fine Adjustment at the Target Pose of a Collaborating Mobile Robot

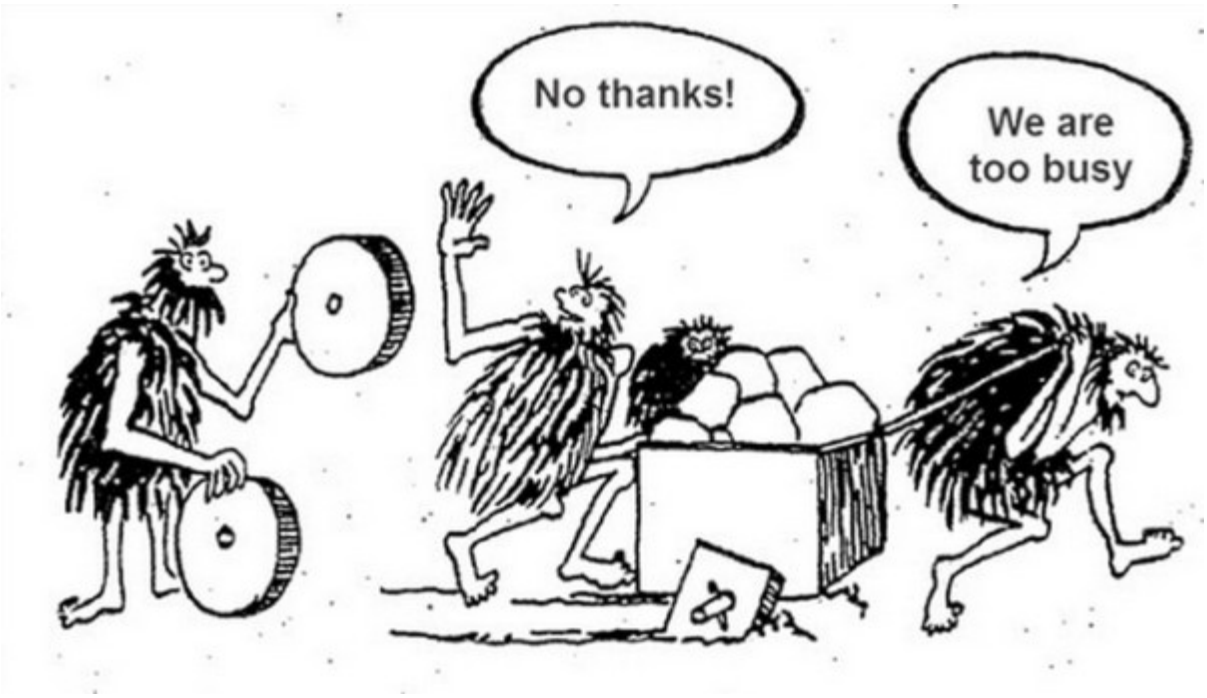
Kai Waelti

1. Reviewer Patricia Duchamp
Mobile Robot Navigation, SLAM and Path Planning
F&P Robotics AG

2. Reviewer Dr. Tim Schwartz
Cognitive Assistants (COS)
German Research Center for Artificial Intelligence (DFKI)

Supervisor Prof. Dr. habil. Jana Koehler

February 16, 2020



Kai Waelti

Strategy for an Autonomous Behavior that Guarantees a Qualitative Fine Adjustment at the Target Pose of a Collaborating Mobile Robot

Master Thesis, February 16, 2020
Reviewers: Patricia Duchamp and Dr. Tim Schwartz
Supervisor: Prof. Dr. habil. Jana Koehler

Lucerne University of Applied Sciences and Arts
School of Information Technology
Suurstoffi 41b
CH-6343 and Rotkreuz

Abstract

Flexible automation technologies often require an accuracy in the millimeter and milliradian range at different target positions. This thesis introduces a strategy that leverages a pose graph-based localization approach to reduce positioning errors at target poses of collaborating mobile robots. Using the precise localization from the pose graph we propose an odometry-based method to improve the accuracy at target locations, thus avoiding manual teach-in of reference scans for ICP that can degenerate in dynamic environments over time. As a basis for this strategy, an extensible software architecture for socially acceptable navigation of mobile robots is designed and implemented using ROS components. This architecture proposal includes a DevOps toolchain to automate the process of software testing, building and delivery to different robotic platforms or the cloud. Finally, we evaluate the proposed method in a industrial setting achieving a position error below $25\,\mathrm{mm}$ in 92.7% and a rotation error below $1.5°$ in 93.9% of the tests. Although the advances shown in this thesis couldn't be statistically tested due to skewed data distributions and some possible outliers, the improvement in both trueness and precision underlines the gains in terms of accuracy positioning.

Abstract (Deutsch)

Flexible Automatisierungstechnologien erfordern oft eine Genauigkeit im Millimeter- und Milliradiumbereich bei unterschiedlichen Zielpositionen. In dieser Arbeit wird eine Strategie vorgestellt, die einen auf dem Pose Graph-basierenden Lokalisierungsansatz nutzt, um Positionierungsfehler bei Zielpositionen von kollaborierenden mobilen Robotern zu reduzieren. Unter Verwendung der präzisen Lokalisierung aus dem Posendiagramm schlagen wir eine auf der Odometrie basierende Methode vor, um die Genauigkeit an den Zielorten zu verbessern und so das manuelle Einlernen von Referenzscans für ICP zu vermeiden, das in dynamischen Umgebungen mit der Zeit ausarten kann. Als Grundlage für diese Strategie wird eine erweiterbare Software-Architektur für eine sozialverträgliche Navigation mobiler Roboter entworfen und mit ROS-Komponenten implementiert. Dieser Architekturvorschlag beinhaltet eine DevOps-Toolkette zur Automatisierung des Prozesses des Softwaretests, der Erstellung und der Auslieferung an verschiedene Roboterplattformen oder die Cloud. Schliesslich evaluieren wir die vorgeschlagene Methode in einem industriellen Umfeld, wobei ein Positionsfehler unter $25\,\mathrm{mm}$ in $92,7\%$ und ein Rotationsfehler unter $1.5°$ in $93,9\%$ der Tests erreicht wird. Obwohl die in dieser Arbeit gezeigten Fortschritte aufgrund verzerrter Datenverteilungen und einiger möglicher AusreiSSer nicht statistisch getestet werden konnten, unterstreicht die Verbesserung sowohl der Richtigkeit als auch der Präzision die Gewinne in Bezug auf die Genauigkeit der Positionierung.

Acknowledgement

First and foremost I want to thank my advisor Prof. Dr. habil. Jana Koehler, for providing me the opportunity to do this project and giving me all support and guidance during the whole project. The same goes for Dr. Tim Schwarz for providing me the topic and good advice at times when it was especially needed. Furthermore, I am grateful for the helpful information and support provided by my whole research team at DFKI and MRK4.0. Especially, I would like to thank Fabio Andres Espinosa for introducing me to the platforms and the long hours of navigation tuning. I am thankful to Patricia Duchamp from F+P Robotics AG for examining this thesis and for their insightful comments on my work. I would also like to thank from all my heart Koel Dutta Chowdhury for her immense support. My most sincere thank you to all of you. A very special thanks to the Enterprise Lab team from HSLU which provided me without interruption a virtual environment for experimentation, as well as the most important parts of the CI/CD toolchain infrastructure.

Contents

1 Introduction **1**

1.1 Thesis Outline and Achieved Contributions 2

1.2 Terminology 3

2 Related Work **5**

2.1 Extensible Software Development Framework for Robotics Research and Deployment 5

2.2 Pose Accuracy of Mobile Robot Localization 9

2.3 Architecture and Algorithms for Socially Acceptable Navigation 17

2.4 Summary of Findings 18

3 Proposal for an Extensible Solution Architecture for Reproducible and Deployable Robotics Research **21**

3.1 Architectural Forces 22

3.1.1 System Vision 22

3.1.2 Context delimitation 23

3.1.3 Functional Requirements 24

3.1.4 Adressed Quality Goals 26

3.1.5 Constraints of the Environment 31

3.2 Solution Architecture 32

3.2.1 Robotic Agent and Environment 33

3.2.2 Intelligent Agent Design 39

3.2.3 Micro Services as Architectural Style 41

3.2.4 Applied Tactics to Achieve the Quality Goals 43

3.2.5 Key Architectural Decisions 49

3.3 Architectural Views 56

3.3.1 Building Block View 56

3.3.2 Runtime View 69

3.3.3 Deployment View 71

3.3.4 Technologies 73

3.3.5 Infrastructure 75

4 Precise Positioning of Mobile Robots at Semantic Poses based on Pose Graph Localization **77**

4.1 Fencepost error correction 77

4.2 Mapping and Localization . . . 79
4.2.1 SLAM and Map Resolution . . . 79
4.3 Robust Navigation Strategy . . . 80
4.3.1 Goal Tolerances . . . 80
4.4 High Accuracy Pose Convergence . . . 82

5 Experimental Evaluation **89**
5.1 Evaluation Methodology . . . 89
5.2 Benchmark Pose Accuracy of Odometry-Based Algorithm . . . 93

6 Conclusion **101**
6.1 Summary of Thesis Findings . . . 101
6.2 Future Work . . . 103

Bibliography **105**

List of Algorithms **119**

A Appendix **121**
A.1 Related Work Analysis for Pose Accuracy . . . 121
A.2 Guiding Quality Design Decisions: Design Checklists . . . 123
A.2.1 Modifiability . . . 123
A.2.2 Usability . . . 125
A.2.3 Testability . . . 126
A.3 Cohesion Analysis . . . 128
A.3.1 Known Pose API . . . 128
A.3.2 Web Dashboard Application . . . 128
A.3.3 Accuracy ROS Node . . . 128
A.3.4 Pose Evaluator . . . 128
A.4 Detailed Architectural Views . . . 129
A.5 More Web Dashboard Screenshots . . . 133
A.5.1 Accuracy Tests UI . . . 135
A.6 Tutorials . . . 137
A.6.1 LifeLong Mapping Using slam_toolbox . . . 137
A.7 Infrastructure Resources . . . 145
A.8 Additional Evaluation Plots . . . 149
A.9 Pipelines . . . 151

Declaration **155**

1 Introduction

> *I do not fear computers. I fear the lack of them.*
>
> — **Isaac Asimov**
> (American writer and professor of biochemistry)

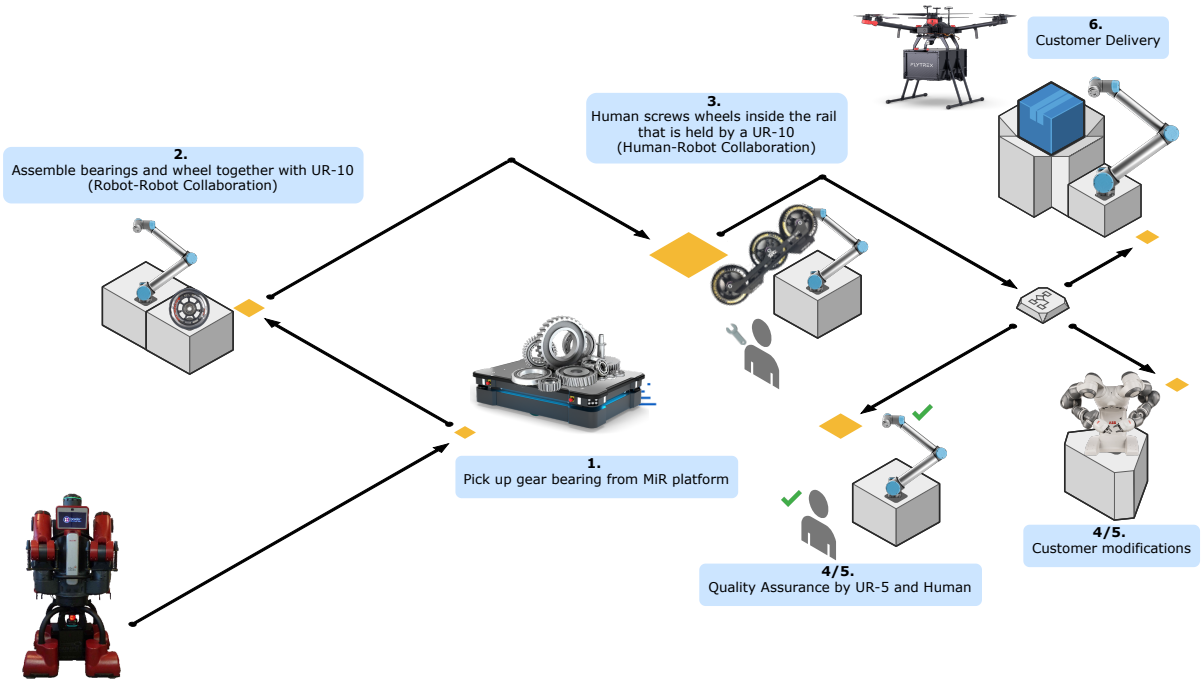


Fig. 1.1.: Co-bot scenario: Precisely visit different workplaces and interact with humans, objects and other mobile or gripping robots

The increasing individualization of products and the desire of customers for faster product cycles is a current trend in the economy [1]. Production systems' design and operation has to become more flexible in order to adapt to current market demands set out by international roadmaps [2], which includes on-demand production, shorter lifecycles, mass-customization schemes, high quality standards, rising speed of delivery and lower fixed costs. A key technology to allow flexibility in industrial applications are collaborative robots (co-bots) that can navigate autonomously and achieve a precise positioning in order to perform mobile manipulation tasks or docking [3].

A co-bot scenario is illustrated in Figure 1.1 as an example of an imagined flexible production process. The process requires a co-bot to transport producing parts over several stations in collaboration with humans

and robots. While a station with a human is rather negligible, a mobile manipulation robot has to pick up or place the parts in contrast, requires a low positioning error. Therefore, the co-bot has to be able to approach the individual stations with different accuracies. Such autonomous navigation requires a complete system that provides autonomous and socially acceptable behaviors. In order to allow the required flexibility, markers of all kinds, are not very welcome in the industry.

This navigation system have to be as generic as possible, to support different mobile robotic platforms that move using different types of mobility types. Irrespective of the platform, the components of the system can be distributed differently. While one co-bot has a powerful control computer embedded, others might require to be controlled using a remote computer.

The term *accuracy* as defined by the *International Vocabulary of Metrology VIM* [4] and ISO 5725-1 [5] as a measure for closeness of a measurement to the true value. Applying this term of accuracy to sets of measurements, involves two components: *random error* and *systematic error*. The sum of both errors yields the *total error*. The ISO standard defines the *closeness of the mean* of a measurement set to the true value as *trueness*. Further, it defines the *closeness of agreement* between the measurement results as *precision*. Therefore, to improve accuracy it is required to improve both *trueness and precision*. The term accurate as an indicator of the quality of an unbiased and precise measurement result. This term should only be used as a general comparison that a method or technology is "more accurate than" the other. A multitude of other terms are used in this thesis and are outlined in Section 1.2

Finally, this thesis attempts at solving the problem that can be stated in a short sentence: "Navigate to different target poses accurately in (x, y) and yaw $\angle\theta$ while giving way to human coworkers and avoid collisions with robot coworkers and other obstacles."

1.1 Thesis Outline and Achieved Contributions

This thesis begins by designing a complete proposal for an extensible solution architecture supporting flexible automation technologies and provides a generic solution for autonomous and socially acceptable navigation. The architecture combines autonomous exploration with life-long mapping and a DevOps toolchain for developing, testing and deploying software increments to robot platforms or the cloud. This system is based on a pose graph-approach for simultaneous localization and mapping which provides precise odometry-like localization. The thesis then further goes on to leverage that precise localization for proposing a strategy that converges the odometry pose with a previously saved goal pose. This is deployed without the need of a manually teached-in reference scan that can degrade over time. The proposed strategy is experimentally evaluated by using an external motion tracking system to capture precise measurements at two tar-

get locations. Finally, the last chapter discusses the findings of the thesis and its limitations, and gives guidance for future research.

Chapter 1
introduces the thesis topic and summarizes the main contributions

Chapter 2
discusses the related work and technical background required

Chapter 3
documents the proposed solution architecture for socially acceptable navigation in flexible environments. The solution is built using a service-oriented architecture with a micro services division where every component is isolated in its own virtualized container together with the data needed for that process. The architecture combines various components with state-of-the-art algorithms that yield a robust and socially acceptable navigation. The second contribution of this thesis is a DevOps toolchain for developing, testing and deploying software increments using a continuous integration and delivery approach. The third contribution is a Gazebo 8.6 port for the simulation plugin of the Mobility Base platform.

Chapter 4
discusses an error with the laser scanner that led to a contributed bugfix. The chapter then goes on to present the improvement opportunities. The fifth contribution is a strategy to converge a precise odometry pose with a goal pose.

Chapter 5
evaluates the strategy presented in Chapter 4 using an external measurement system. The thesis compares the results with the standard navigation stack that was configured in Chapter 3.

Chapter 6
Lastly, in our final chapter, we not only briefly summarise our main findings,but also point to some of the limitations of our work and towards future directions.

1.2 Terminology

Here we introduce some key terminology that is used throughout the thesis

Pose of an object include its location and orientation in 6D

Yaw is a movement (rotation) around the yaw axis of a rigid body

Virtualization is the process of creating, running and managing virtual environments or computing resources; abstracts physical resources

Virtual Machines is a hardware virtualization technology that emulates a full computer

(Linux) Container containers are an operating system virtualization technology used to package applications and their dependencies and run them in isolated environments

Container Runtime is the component that actually runs and manages containers on a host computer

Docker first technology to successfully popularize Linux Containers; is the widely used solution to this date

Development tools are computer programs for creating, debugging, maintain or otherwise support other programs and applications used by developers

Development Strategies are the methods and processes employed for developing software

DevOps is a combination of development and operations, which is a set of practices that automates the processes between software development and IT operation teams

2 Related Work

> *If you're not failing every now and again, it's a sign you're not doing anything very innovative.*
>
> — **Woody Allen**
> Director

In this chapter all previous work found that relates to solving the task is discussed. First this includes investigating the state of the art in robot software development for research. Second, the last section discusses the methodology to evaluate pose accuracy of mobile robot navigation and localization and how to improve upon it. Finally, a sighting of architectures and algorithms for socially acceptable navigation was done to identify solutions best fit for my purpose.

2.1 Extensible Software Development Framework for Robotics Research and Deployment

An integrated robot system for mobile manipulation requires diverse skills. Skills that bridge the gap between AI and robotics, such as robot perception, knowledge representation and reasoning for robotic agents. Therefore, developing a complex robotic system from the ground up requires a high level of specialization in a large number of diverse scientific areas. Also, great effort has to be applied in order to join the separate software modules for the whole robotic system to work fluently. Using robotic frameworks thus is required to overcome these obstacles. There are a number of well known robotic frameworks available [6, 7, 8, 9, 10, 11] that vary in operation system and programming language support, as well as their focus. The focus of a framework can be e.g. real-time, control or simulation and is not exclusive.

In [12], extensive comparison between robotic frameworks is performed. The paper first defines a nomenclature to distinct between and compare different robotic frameworks, middlewares and architectures. A collection of software tools, libraries and conventions that aims at simplifying the task of developing software for a complex robotic device is defined as a *Robotic framework*. Using such a framework often restricts the general architectural principles of the developed software. Such general architectural principles could be of two kinds - centralized or real-time system. A *Robotic middleware* is de-

scribed as the glue that holds together the different modules of a robotic system. A middleware provides the communications infrastructure between software nodes running in a robotic system as its most basic task. As these interfaces consist of various OS specific drivers, a middleware also usually provides the essential software, such as hardware interfaces between the high level and low level components of the system. Differentiating from the definition of robotic frameworks and middlewares is a *Robotic architecture*, which is a more abstract concept. A robotic architecture describes how modules in a robotic system should be interconnected and interact.

Having defined the nomenclature, [12] found that the Robot Operating System ROS, has the largest momentum and appeal to the community. This fact is also supported by the large amount of submitted algorithms by the scientific community in the ROS wiki that covers a wide range of applications. Contrary to its name, the Robot Operating System is a framework as defined in the previous nomenclature. Specifically targeted for writing robot software, ROS is comprised of tools, libraries and conventions that aim for complexity reduction, concerning the procedure of writing complex and robust robotic behaviours. As it provides standard system operating services, such as hardware abstraction, low-level device control, implementation of commonly used functionality and message-passing between processes and package management.

Because of the huge amount of diversity in robotic applications, the research in [13] finds that establishing a unified structure for intelligent robot systems is difficult. Absorbing previous experience and lessons learned from past typical architectures is important. Moving away from modules towards agent-based control architectures can bring independence, autonomy, intelligence and flexibility. Such architectures consist of different types of agents that are responsible for different aspects. The different types of agents can be found in the AI classic [14]. A navigation task can be abstracted into different layers with different types of agents that are just reactive or deliberate. Examples for a deliberate agent are task or path planning and localization. As for a reactive agent, examples are navigation and obstacle avoidance. Building on this, [13] propose six design principles for constructing intelligent robot systems. These design principles are implemented in the proposed architecture and consist of

1. Modular Division based on the primitive Sense, Plan, Act (S, P, A)
2. Hierarchical Principle of Architecture
3. Principle of High Cohesion and Low Coupling
4. Design Principle of Redundancy
5. Coordination Principle Between Deliberative and Reactive Layers
6. Extend Software Design Methods to Robotics

The first design principle *modularity* is key to designing an intelligent robot system, which basis division depends on the task and the application domain. Classification of the modules depends on choosing different granularity such as function, behavior and component. All modules are explained as expansion or extension of the primitive (S,P,A). Sensing can be seen as the ability to percept internal and external state changes, and to understand the meaning of these changes.

Planning is the ability to make decisions autonomously based on conditions, states and constraints. Acting is the basic action and behavior of the robot. As such, a module can i.e. follow a *reactive paradigm* $(S \rightarrow A)$ or *deliberative paradigm* $(S \rightarrow P \rightarrow A)$.

The hierarchical principle, divides behavior into layers similar to the animal spatial behavior. Division of the agent-oriented distributed hybrid architecture is based on deliberative and reactive paradigms. Splitting reflex-like, fusion, learning and cognition behaviors in different layers to achieve a more complex cognitive behavior. The principle of high cohesion and low coupling is embodied during the design process. Relationships between modules determine data and control flow. A reasonable division of subsystems that solve sub-problems are important. At the same time, this design strategy considers how to synthesize the sub-problems for solving the original problem.

Redundancy in the design increases robustness and adaptability to the environment. Redundancy can come from either part or function. Functional redundancy is easier and cheaper to achieve, as it mostly involves fusing perception from different sensors. An example for functional redundancy for mobile robot navigation is overlapping vision from a camera with range data from a laser scanner. Both sensors have their advantages and disadvantages in certain situations and can provide more robustness if they are combined. Coordination between the different reactive and deliberative layers make a robot have strong features and application flexibility. Complementing deliberative abilities such as modeling, planning and other intelligent decisions with reactive abilities. Such reactive abilities allow real-time responding for overcoming the uncertainty of dynamic changes during the execution of intelligent decisions. Lastly, extending software design methods to robotics, such as Service-Oriented Architecture (SOA) brings important advantages. Such methods can improve modularity, robustness, flexibility, durability, scalability and reuse of robotic software.

These proposed principles are used in the solution architecture described in Chapter 3. The desirable requirements for the control software architectures of autonomous mobile robots described in [15] also supports these principles. They describe a gap between available methodology/technology and the new application and market demands that require fully autonomous robots. These demands motivate researchers and practitioners to develop novel methods and techniques to overcoming the large uncertainties in the real world environment and facing the sensory imprecisions and inaccuracies. In turn, this motivates the importance to achieve

Robot hardware abstraction for portability to different platforms

Extendability and scalability for adding new hardware and software components to the system

Reusability of software components, structure, framework and patterns

Repeatability of behaviors and achieved results

Low overhead in memory and CPU requirements, frequency and end-to-end latency at run-time

A critical factor in software development for ROS is that building, running and shipping

applications and services is a daunting endeavor for non-experts with a formidable learning curve described in [16]. The ROS ecosystem builds from fast-growing, open source, continuously evolving community of newly released linux distributions[1], updated dependencies and deprecated packages. Combined with a curse of dimensionality that plagues developers when developing robotic applications, makes packaging an application so that it runs on any system regardless of operating system, a hard challenge. The curse of dimensionality comes from the the various robotic platforms, peripherals, operating systems and ROS distributions that a developer has to consider, as well as used third-party software libraries. The packaging is further complicated by shared packages or libraries on which several other packages have dependencies upon but where they depend on different and incompatible versions, often called "dependency hell".

Furthermore, the field of robotics is not solely composed of trained software engineers, but is rather one of the most interdisciplinary fields in existence. While classically it started in engineering, it has now evolved to psychologists for human-robot collaboration [17] and neuroscientists which enrich the collaboration by integrating brain-robot interfaces [18]. In order to allow non-experts in robotic software development to develop their component independently, easier development tools and environments are required.

In [16], they utilize advances in Linux containers to showcase development tools and strategies to construct repeatable and reproducible environments, as well as running and shipping portable ROS applications. This opens up many new possibilities in the interdisciplinary field of robotics such as getting started without cumbersome environment setup and setting up workflows for continuous integration and test verification. Due to the lack of such a workflow, [19] found that this represents a significant obstacle for testing and reusing purposes. A hard to test and reuse package, mixed together with the tendency for experimental code to be accompanied by little explanatory material, leads to issues with repeatable and reproducible environmental setups. Research reproducibility and performance evaluation as described by [20] has a high importance, especially for dependable robots. Properties such as e.g. safety, reliability and availability that can be objectively measured are part of the dependability. Additionally, the dependability also contains a subjective perspective that reflects the level of trust end-users have in the system performance.

A standard architecture called "Plug and Prototype P&P" to enable fast software prototyping and generating modular and easily extensible structures is proposed by [21], which aims at providing a standardized pattern for developing applications for ideally any robot. This framework however suggests to encapsulate all resources needed in one single container, which leads to spawning all nodes in a single bash process. This approach is in contrary to the paradigm of a microservice architecture, where each small service runs in its own process and communicates with lightweight mechanisms. Running every ROS node in a seperate container allows longer life-cycles for certain nodes and to start them dynamically. Furthermore, this

[1] A linux distribution is an operating system made from a software collection that is based upon the Linux kernel

eases the process of running multiple or different versions of language runtimes and collections of software with conflicting dependencies.

In contrast to Virtual Machine VM technologies, Linux Container virtualization has the ability to load and communicate directly with the GPU driver using the `Nvidia Container Toolkit` project `nvidia-docker`[2]. This allows to run all sorts of hardware accelerated/intensive applications inside a container directly on the GPU with only minimal overhead as [22] found. Their research found no noticeable drawbacks of Docker containers in CPU and GPU tests. They performed just as good as programs running in the host system. However, this technology only supports Linux platforms as of now, which limits the deployability of such containers. Containers are mostly for simulation or model training purposes they can be run on a seperate machine running any Linux distribution supporting Docker.

2.2 Pose Accuracy of Mobile Robot Localization

Determining the robot's position and rotation (its "pose") by using its sensor and motion observations is known as robot localization. The localization of where the robot is occurs within a frame of reference, which is its internal map of the environment. Because motion and sensors are not always perfectly accurate and the environment is dynamic, localization needs to deal with some uncertainties.

The definition of *pose* used in this thesis is of a 6D Cartesian pose that is relative to a reference coordinate frame. Such a Cartesian pose is composed of a position and orientation component. The position component is simply a point consisting of 3 coordinates x, y and z. The position point describes an absolute location in 3D space using these coordinates. The second component of the Cartesian pose is the orientation and comes in *quaternion* form.

Quaternions represent the shortest path to get from one rotation to another. [23] gives a good introduction to quaternion algebra and rotation operators. Simply put, a quaternion represents an *orientation* or *rotation delta*. Quaternions are made out of 4D vectors of the form $[x, y, z, w]$. The x, y and z components are factors of complex numbers $\mathbb{C}$, and w is the scalar part $\mathbb{R}$. Rotating smoothly and directly over the shortest path using quaternions is as simple as matrix multiplication. Typically, this is done by taking the current orientation as a quaternion and multiplying it by another rotation quaternion. While the quaternions are not commutative, they are associative, and form a group known as the quaternion group. This is the case, because rotating consecutively $90°$ about x-axis and y-axis returns a different rotation than doing it in the reverse order. These different sequences can be observed in Figures 2.1 and 2.2. Therefore, for quaternions p and q the following has to be consid-

[2] NVIDIA Toolkit can be found under https://github.com/NVIDIA/nvidia-docker

ered i.e., $pq \neq qp$. A subset $\{v_1, v_2, ..., v_k\}$ of a vector space V is called orthonormal if the inner product $\langle v_i, v_j \rangle = 0$ such that $i \neq j$. These vectors v_i and v_j are said to be mutually perpendicular when $\langle v_i, v_i \rangle = 1$. A basis is called an orthonormal basis, if the orthonormal set is linearly independent and is a vector basis for the space it spans. The standard orthonormal basis for $\mathbb{R}^3$ is given by three unit vectors $i = (1, 0, 0)$, $j = (0, 1, 0)$ and $k = (0, 0, 1)$.

A quaternion is defined as the sum of a scalar w and a vector $q = (x, y, z)$:

$$q = w + q = w + xi + yj + zk$$

Utilizing the inner product and cross product of two vectors in $\mathbb{R}^3$ the quaternion product pq can be written as follows

$$pq = w_p w_q - \boldsymbol{p} \cdot \boldsymbol{q} + w_p \boldsymbol{q} + w_q \boldsymbol{p} + \boldsymbol{p} \times \boldsymbol{q}$$

In the equation above, $\boldsymbol{p} = (x_p, y_p, z_p)$ and $\boldsymbol{q} = (x_q, y_q, z_q)$ are the two vector parts of both $\boldsymbol{p}$ and $\boldsymbol{q}$, respectively.

A complex conjugate of a complex number in mathematics is a number with an equal real part $\Re(z)$ and an imaginary part $\Im(z)$ equal in magnitude but opposite in sign. If for example $z = a + bi$, where a and b are real, then the complex conjugate of $z = a + bi$ is $z^* = a - bi$. The complex conjugate of a quaternion $q = w + xi + yj + zk$ denoted q^* is defined as

$$q^* = w - \boldsymbol{q} = w - xi - yj - zk$$

From this definition also follows, that $q^*q = qq^*$ and given two quaternions p and q, $(pq)^* = q^*p^*$ can be easily verified.

The *norm* of a quaternion q is denoted by $|q|$ and is the scalar $|q| = \sqrt{q^*q}$. If the norm of a quaternion is equal to 1, it is called a *unit quaternion*. For two quaternions p and q the norm of their product is the product of the individual norms as follows

$$|pq|^2 = (pq)(pq)^* = |p|^2|q|^2$$

With the definitions of a complex conjugate and the norm of a quaternion, the *inverse* of a quaternion q can be defined as

$$q^{-1} = \frac{q^*}{|q|^2}$$

While it can be verified that $q^{-1}q = qq^{-1} = 1$, in the simple case of a unit quaternion, the inverse is just its conjugate q^*.

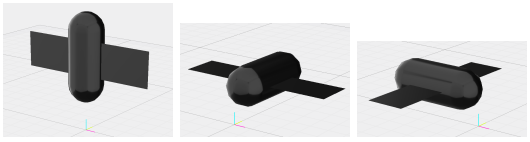


Fig. 2.1.: Rotation around x-axis then around y-axis

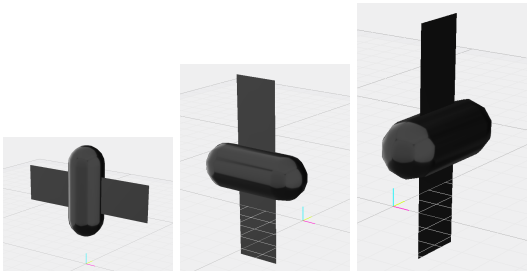


Fig. 2.2.: Rotation around y-axis then around x-axis

Euler Angles on the contrary, are intuitive and easier to work with. They can be thought of as a sequence of steps in 3D called *pitch* X, *roll* Y and *yaw* Z. However, these angles have a *gimbal lock* flaw, that occurs when two of the three axis line up. This *locks* the system into rotation in a degenerate 2D space where one degree of freedom is lost. Converting from and to Euler angles can be difficult, as there are multiple solutions for a given orientation. Therefore, if required to work with Euler angles, the *transformation* library from ROS should be used. Also, these conversions should happen only at the last possible moment, doing as much with quaternions as possible.

Localization and Pose Accuracy is a key requirement for mobile robots. On the factory floor in industrial applications a high degree of accuracy is required, in order to perform accurate docking or mobile manipulation tasks such as pick and place. For such applications, it is required to achieve precise positioning at target locations. Popular solutions for accurate positioning as found by [3] relies on modifications of the environment using embedded wires or magnetic tapes. Other solutions depends on the modifications of the environment, which make use of V or VL shaped [24] or QR code markers [25].

Whereas many approaches to mobile robot localization have been proposed in the past, mostly probabilistic approaches that localize robots with respect to a given map have been the most successful application-wise. These approaches often rely on techniques such as ICP-based pose graphs [26], extended Kalman filters EKF [27] or the faster and unstable unscented Kalman filters UKF [28], histogram filters [29] or particle filters. Particle filters are often referred to as Monte-Carlo localization MCL [30] and variants thereof.

Some marker-based technologies using Radio-Frequency Identification RFID or wireless receivers estimating radio signal strength have been evaluated in [31, 32, 33] for the accuracy they provide and how well they can deal with changes in the environment. Using the Active Beacon technology [31], achieved a positional accuracy below $5\,\mathrm{mm}$, which can deal with *heavy changes and dynamics* (rotating tables, moving furniture, robots, people, etc.) in the environment. In contrast to this, the solution using landmark bearings [32] was only able to achieve below $3.8\,\mathrm{cm}$, while only dealing with *isolated changes* to the environment. However, none of these works considered the *orientation accuracy* of the final pose. While some of those systems work precisely, they rely on modifications of the environment, such as installing beacons and landmark bearings at specific locations. These modifications limit the flexibility of a robotic system.

As for the egocentric sensors such as 2D cameras, there are three vision-based approaches that are available for mobile robot localization. The oldest experimental results from [34] achieved - with a maximal error of $6.8\,\mathrm{cm}$ - the best positional accuracy of that category of sensors. This work capitalized on the strengths of image and landmark-based methods by encoding images as a set of visual features. These features are then used to detect landmarks and perform localization using a landmark tracking and interpo-

lation method. This approach is markerless, as it learns the landmarks using visual characteristics of interest related to object recognition. Other approaches uses Scale Invariant Feature Transform SIFT to extract features with a single perspective camera in [35] and a stereo camera in [36] and uses Monte-Carlo localization. Using a monocular camera yielded below $39\,\mathrm{cm}$ *positional* and $4.5°$ *rotational accuracy*, while using a stereo camera yielded a $19\,\mathrm{cm}$ better positional but $1.5°$ worse rotational accuracy. Dealing with dynamics in the scene is rather difficult using single cameras, but moderate changes can be overcome by fusing various low cost cameras as described in [37]. Unfortunately, this approach of fusing low cost cameras only achieved a sub-metre positional and reported no rotational accuracy.

Most robotic platforms are equipped with laser rangefinders, which provides precise distances to obstacles and requires no data pre-processing. In [3], they combined Monte-Carlo localization with KLD sampling and fine repositioning using scan alignment with the *Iterative Closest Point ICP* algorithm. In an experimental evaluation of their system, they were able to achieve position errors between $5\,\mathrm{mm}$ and $15\,\mathrm{mm}$. The reason for that interval is the amount of change in the environment, where the upper bound was reached when dealing with eight people as well as additional objects placed in the direct proximity of the robot. Also, a small rotational error of $0.15°$ was achieved.

Using the ICP algorithm for finding the transformation that best aligns points of a current point cloud reading with respect to a reference. ICP is a rather simple and modular algorithm that was originally introduced in [38]. Numerous variants have been developed along the years, with the wide range of selection of proper configuration and its parameters that normally requires empirical tests and experience. Factors like the environment in which the robot evolves, the sensor, the amount of processing data, the data organization schema, the algorithm configuration and the measurement errors in the data needs to be considered, to make a precise scan registration.

Using time of flight ToF or structured-light 3D camera technologies, similarly to laser rangefinders, returns point clouds from scans that can be aligned to compute a translation and rotation between them. This can be thought of as a 3D point cloud that would yield more information and a more accurate description of the environment. In order for the computation to be efficient, the data would have to be filtered wisely. The performance of two registration algorithms ICP and *normal distribution transform NDT* on 3D point clouds have been evaluated for autonomous vehicles by [39]. Their comparison showed that NDT possesses a better ability to handle realistic adversity conditions such as static and dynamic environmental changes. With NDT they achieved a positional error below $40\,\mathrm{mm}$ realistic field tests in dynamic environments. As the vehicle was moving while computing the registrations and the (sub-)urban environments are continually changing which leads to significant differences. These temporal changes that occur over varying time-scales include dynamic and ephemeral objects (such as parked cars), seasonal changes (vegetation, snow, dust) and human impacts such as construction. Therefore, it is expected, that 3D

augmented scan alignments in more controllable environments perform better.

Monte Carlo Localization deals with this uncertainty by generating many random guesses of where the robot is going to be next. These guesses are known as particles and contain a full description of a possible future state. Such a state for a 2D mobile robot consists of a tuple (x, y, θ) for position x, y and orientation θ. The robot's estimate of its current state is a probability density function distributed over the state space and is called the belief. In the MCL algorithm, this belief at time t is represented by a set of M particles, where each state can be considered a hypothesis.

$$X_t = \left\{ x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]} \right\}$$

Regions in the state space with many particles is predictive of the robot's location and vice versa. The MCL algorithm assumes the "Markov property" where the current state depends only on the previous state (i.e. X_t depends only on X_{t-1}). This assumption only works if the environment does not change with time and is static. The algorithm at time t takes an input – the previous belief X_{t-1}, an actuation command u_t, some data from sensors z_t and outputs its new belief X_t.

KLD-Sampling adapts the number of particles which is based on how sure the robot is of its position. The original Monte Carlo localization algorithm is fairly simple. Many variants to this algorithm have been proposed that has tried to address its shortcomings or adapt it to perform effectively in certain situations. One of them is the MCL with KLD-sampling which samples the particles based on an error estimate using the KLD. Using the same state space as with normal MCL, a histogram is overlaid where each bin is initially empty. A new particle is drawn from the previous weighted patricle is set at each iteration with a probability proportional to its weight. Resampling in classic MCL, the robot generates a set of new particles, with most of them generated around the previous particles with more weight. In KLD-sampling algorithm, particles are drawn from the previous, weighted particle set and gets applied motion and sensor updates before placing the particle into its respective bin. Furthermore, the algorithm tracks the number of non-empty bins k by recalculating M_x each time a particle is inserted into a previously empty bin. This benefits computational requirements, as it increases linearly in k and is repeated until the sample size M is equal to M_x. In practice, KLD-sampling consistently outperforms and converges faster than classic MCL.

Simultaneous Localization and Mapping SLAM

In order for the robot to localize itself, it first has to create a map of the environment. The problem of building a map of an unknown environment while continuously estimating the robot pose is called simultaneous localization and mapping SLAM and as described by [40] is one key issue that prevents the creation of truly autonomous mobile robots. Although different algorithms have been proposed, the universal one has not been found yet. Laser range-finders are widespread sen-

sors and indoor navigation of mobile robots can be represented as a 2D problem. This representation in lower dimension comes at the cost of reduced autonomy in uneven or cluttered environments. The focus of this thesis lies on the 2D laser scan based SLAM. While this does not exclude 3D cameras of different types, it does require their point clouds to be down-projected onto a 2D plane. The benefit of doing this is being able to register obstacles not only on a horizontal line but obstacles on all heights, which should increase the robustness and autonomy of the robot's navigation. This benefit comes at a cost of much higher computational requirements as reported by [41].

Loop closure is the problem of detecting when the robot has returned to a past location after discovering new terrain for a while, which is crucial to enhance the performance and robustness of SLAM. The most popular SLAM algorithms to date are Google's Cartographer [42], GMapping [43] and Hector SLAM [44], which have been evaluated and compared by [45]. The research found that Google's Cartographer constructed maps in almost all scenarios with the smallest error relative to a precise ground truth while being robust w.r.t. the different type of mobile robot movements. The maps constructed with Gmapping, whose quality are not far from Cartographer's, ranked second in the test. Compared to Hector SLAM, both Cartographer and Gmapping make additional use of odometry for localization rectification and map correction, besides using light detection and ranging LiDAR sensor data. Cartographer and Gmapping also provide an explicit option for loop closure. Because Hector SLAM does not provide an explicit option for loop closure and uses only LiDAR data, its results are less accurate than other algorithms.

Cartographer can be seen as two separate subsystems for local and global SLAM. The local SLAM subsystem is sometimes called the frontend and is responsible for building a succession of submaps. Each of those submaps are meant to be locally consistent but are allowed to drift locally over time. The global SLAM subsystem is also called the backend and tries to find loop closure constraints. This is achieved in the backend by matching scans from sensors against submaps. Incorporating other sensor data allows it to get a higher level view and identify the most consistent global solution. At a lower level of details, this means generating good submaps with local SLAM and tying them together as consistenly as possible with global SLAM. Unfortunately, Cartographer seems not very consistent in repetitive environments and does not work that well in practice as can be found in some issues on GitHub [3]. Cartographer requires tuning to make it pick up global loop closures of submaps in large and repetitive environments, which defines most industrial environments.

A graph-based approach `KartoSLAM` was proposed by SRI International's Karto Robotics, which [46] found to produce accurate maps with lower CPU load. The complete standalone library for robust 2D mapping was open sourced as `OpenKarto`[4] with very little documentation. The library uses the correlative scan matching algorithm by [47] in the

[3] Issues #127, #141 and #1617 can be found on GitHub https://github.com/googlecartographer/cartographer/issues
[4] The ROS Karto Wiki can be found under http://www.ros.org/wiki/karto

frontend and a Sparse Pose Adjustment SPA from [48] in the backend. The high quality scan matcher gives the algorithm good accuracy and combined with the SPA which uses the poses to minimize the error caused by the constraints, and in doing so, reduces the errors accumulated before loop closure. This combination has been previously shown by [49] to give excellent mapping results with minimal errors in large real-world datasets. Using a graph-based approach, [50] shows the possibility to build maps using multiple robots, by keeping track of the positions and subscribing to the laser feeds of all robots.

ICP-Based Pose-Graph SLAM provides an odometry-like localization solution upon LiDAR sensors as proposed by [26]. The proposed solution registers sequentially point clouds along a robot trajectory. Such a point cloud represents a set of data points in space. Generated by 2D or 3D scanners, they measure points on the external surfaces of objects around them.

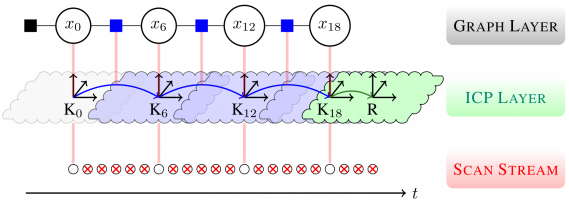


Fig. 2.3.: Two layers in the ICP-based pose-graph SLAM system

A problem in only using ICP and with odometry systems in general, is the accumulation of errors made at every registration that leads to a drift of the overall position estimate. Without any prior map of the environment, resorting to a SLAM approach is the only way to reduce this drift. Probabilistic graphical model theory proposed in [51] was used to solve the SLAM problem through optimization. The computation time needed to converge to a solution can be considerably reduced by exploring the sparsity of the SLAM problem. Graphical models used to model the optimization problem provide a powerful layer of abstraction that helps to better understand the problem and design powerful solutions.

The diagram in Figure 2.3 gives an overview of an ICP-based pose-graph SLAM system. It describes a hypothetical state after the system processed some scans. Two main layers on the top plus the scan stream on the bottom represents simply the history of acquired scans. The ICP layer is composed of *keyframes* K_i, and their associated scans that are called *keyframe scans*. All associated scans for a keyframe scan in the ICP layer are found within the cloud shapes. Each keyframe K_i is associated with a node x_i in the graph layer above. Some of these keyframe scans are selected to compose *local maps* represented in the Figure by blue clouds. These local maps are used as reference inputs for the ICP process. Other keyframe scans represented in the Figure by a gray cloud are only stored and may at a later point be used to compose a local map. Such a composition is only possible if the robot revisits this part of the environment again. The robot pose R is always expressed with respect to the closest keyframe in the current local map. The system does not require a complete map of the environment, but can be computed on demand as the concatenation of all keyframe scans.

The robot initial position with respect to the world origin is added to the graph depicted as black square in the Figure. The first keyframe K_0 is associated to the first ac-

quired scan and with it, the associated node x_0 is created in the pose-graph. Every time a new scan depicted as green cloud is available at the current robot position R, the ICP finds the transformation that aligns it best with the current local map depicted as green arrow, correcting the robot position. If the overlap between the current scan and the current local map is lower than a defined threshold, a new keyframe is created. Otherwise the scan is discarded, as depicted by the crossed circles in the scan stream.

As new keyframes are created, they are initialized with the corrected robot frame. This adds a new frame variable as a node to the pose-graph, as well as a factor depicted as blue square that contains the transformation between the new and former keyframes. This transformation is shown in the ICP layer by the blue arrows. Finally, the local map is rebuilt by incorporating the newest and removing the furthest keyframe scan.

Potential loop closures between two keyframes, when detected by the system trigger a local map is built around the oldest between these loop closing keyframes. A ICP call then tries to align the scan of the other loop closing keyframes with this local map. If that ICP call is successful, a new factor is added to the pose-graph between the variables associated with these loop closing keyframes. This closes a loop at the graph representation layer, which as a result, triggers a optimization that uses the pose-graph data. Subsequently, the loop ends by repositioning the keyframes. Local maps are also reconstructed using these new keyframe values. At the same time, the robot frame R being expressed in the closest keyframe, is also updated by the loop closing process.

The `slam_toolbox`[5] is built on top of the `OpenKarto` SDK with support for large and dynamic spaces. This project contains different tools and capabilities for 2D planar SLAM. The pose-graph approach maintains a rolling buffer of recent scans in previously visited poses, which can be used to load a saved pose-graph and continue mapping while removing extraneous information from newly added scans. Unlike other SLAM libraries, it does not rely upon other localization such as adaptive (or Kullback-Leibler divergence KLD-sampling) Monte Carlo localization AMCL from [52] which uses a particle filter to track the pose of a robot against a known map. The `slam_toolbox` provides a motion model by getting odometry using LiDAR which uses optimization-based localization built on the pose-graph and OpenKarto's high-performance scan matcher. Getting precise odometry information aids on the generation of more precise localization, since understanding the robot dynamics we can estimate its pose.

[5] Announcement of the package https://discourse.ros.org/t/announcing-stable-release-of-slam-toolbox/9872

2.3 Architecture and Algorithms for Socially Acceptable Navigation

Navigation of mobile robots is a key factor in order to be fully autonomous and achieving flexible automation technologies. A lot of research has been done on mobile robot navigation with various robotic platforms in order to make the systems more autonomous and robust. Some of these research built resources for ROS that allows a robot to plan and follow a path while deviating from obstacles [53]. The ROS navigation stack that can be found in the package `move_base` that provides those resources for differential drive and holonomic robots. The navigation task requires many other resources, such as robot localization in a static or dynamic map, as well as a motion model of the robot and some sensor sources.

As powerful as the `ROS` navigation stack is, there are a ton of parameters to fine tune in order to maximize its performance. The guide from [54] describes the main parts and offers a description of some of the available algorithms for global and local planning. The guide also describes the meaning and effects of some of the parameters the individual components offer. This makes the navigation guide a good starting point and can be used as a reference to perform navigation performance optimization for the mobile platform at hand.

In order for robots to seamlessly integrate into the human physical and social environment, they must display appropriate *proxemic behavior* (the use of space). This means that they must adhere to social norms in determining their physical and psychological distance from people. There are many proficient competing models of human proxemic behavior. All models conclude that the proxemic behaviour of an individual is determined by the proxemic behaviour of others and the proximity of the individual to them. In [55], they explore whether these models of human proxemic behavior can also be used to explain how people physically and psychologically distance themselves from robots. As path planning in mixed human-robot environment could distract, alarm or distress humans as [56] points out, robots should behave in a socially acceptable manner. This paper also lays a mathematical background for formulating the social path planning problem which eases tracking and compares different frameworks.

There are 6 rules as high-level requirements that are accepted in literature for successful human-robot interaction, which are stated in [57]

Collision-free Maintains robot and human safety

Interference-free Robot should not - unless it is the objective - enter the personal space of any human

Waiting If robot enters personal space of a human, it has to stop within a fixed amount of time

Human priority Humans always have highest priority

Robot intrusion If a robot enters the workspace of another robot, it should

leave, while the other robot(s) should stop their activities

Robot priority Robots with higher priority should get yielded passage

However, those rules should not be strictly applied following [56] as they could lower the *robot behavior optimality* (could lead to unwanted effects). While trying to give the robot a friendly behavior, for efficiency reasons, the robot should also act optimally in terms of path length, smoothness, safety, energy consumption and execution time. Therefore, these conventions should rather be seen as general preferences or guidelines following [58, 59], which discourage a subset of paths without disallowing them outright.

Most path-planners used in navigation systems nowadays use a discretized costmap, where values in the *costmap cells* (cells in a 2D grid, often called occupancy grid[6]) correspond to the "badness" of the robot being in that position. Using that costmap, the path-planner then generates a path from the start position to the end, which has the minimum accumulated cost. While a rather close approach of any obstacle is fine most of the times, it is not socially acceptable with people. The work in [58] explored a Gaussian adjustment to the costmaps of the ROS navigation stack in order to model the personal space of people with a amplitude ranges and without limits on variance. They found that there are large *dead zones* (provide region of zero output) in the parameters and discontinuities when configuring single parameters, which have to be tackled when configuring a system. Also noteworthy is the associated difficulties of creating the desired behavior around people which are often due to unforeseen interactions between different elements in the costmap. More complex cost functions may alleviate some of those problems. While the approach using a Gaussian adjustment does not lead to a perfect social navigation it is certainly a better solution than having no costmap modifications at all.

2.4 Summary of Findings

In this chapter we show that while using the state-of-the-art robotic framework ROS together with the containerization of Docker it is possible to build an extendable software development framework for mobile robots, which support flexible automation technologies. This framework should follow the micro-service architecture in order to benefit of varying life-cycles and dynamic starting of components, as well as to manage dependencies and make deployments maintainable with continuous integration. In combination, this makes robotic research more collaborative, eases repeatability and reproducibility and encourages reuse of components or complete architectures. Additionally, this ecosystem enables experts of related fields without a deep computer science understanding to develop prototypes rapidly.

[6] Each grid cell in the costmap has a value in the range [0, 255] representing the cost of navigating through that grid cell.

Navigation of mobile robots is a key competence for full autonomy and flexible automation technologies. In order to use the powerful ROS navigation stack, it is required to provide a map of the environment in which the robot is localized, along with motion model that suitably describes the robot motions and sensor sources. Every component needs to be tuned further for the mobile platform at hand. Mapping large and dynamic spaces is a challenging task and requires continuous mapping using accurate odometry at an adequate scanning resolution. Path planning in mixed human-robot environments could distract, alarm or distress humans. Therefore, robots should try to follow 6 guidelines for HRI, while being optimal in terms of efficiency metrics. Using a Gaussian adjustment to costmaps leads to a better social navigation with and around humans.

Because of the flexibility limitations of marker-based solutions, this thesis focuses on *ego-centric solutions* to localize a mobile robot system using only *built-in sensors*. Promising results have been achieved using laser rangefinders, which are available on most robotic platforms, that returns precise distances to obstacles without requiring any additional data pre-processing. Using out-of-the-box *MCL localization with KLD sampling* and fine repositioning using *scan alignment with ICP* a position error below $15\,\mathrm{mm}$ and rotational error below $0.15°$ is seen to be achievable. However, this approach requires a reference scan captured at a given point in time. This reference scan can degrade over time as the environment changes and requires re-registration to achieve previous precision.

A recent trend using an ICP-based pose-graph approach for SLAM allows to combine laser odometry with velocity or wheel odometry results in an accurate motion model. Refined by a pose-graph layer to reduce the drift of the overall position estimate offers a powerful solution. This localization is independent from the occupancy grid generated, which is used by the ROS nagivation stack. Consequently, the localization has the potential to be more accurate than AMCL in the defined map resolution. The `slam_toolbox` integrates localization with lifelong continuous mapping for 2D environments.

This thesis combines the benefits of continuously updating ICP scans transformation with fine adjustment of the robot pose at target locations. The evaluation of the custom fine adjustment odometry based method for fine adjustment at target locations is compared to the ROS standard navigation stack and the ground truth measured using a $\pm 0.3\,\mathrm{mm}$ and $0.05°$ accurate external system. In order to be successful and efficient in a human-robot collaboration scenario, a socially acceptable navigation is employed on the platform. This navigation detects human legs and uses that information to follow a set of general preferences that discourages certain paths that would give the robot a less friendly behavior. Every process is developed in a seperate and isolated container along with the environment and data needed for that process. Container templates for various ROS environments in C++11 and Python 2.7 are made available for robotic software developers. For non-robotic developers, a web-based *Integrated Development Environment IDE* containing all required dependencies and integrated with ROS, OpenCV, TensorFlow, a.o. is provided that works for

Python 2.7. This web-based platform can also be used for rapid prototyping, that is, to quickly test a new idea without cumbersome environment setup. For all kinds of users, a Web Application can be used to get various status and debug information, as well as for sending commands to and from the robot. The web application can also be used to define new semantic positions that the base should drive to accurately ($\pm 1\,\mathrm{cm}$) or inaccurately ($\pm 5\,\mathrm{cm}$).

3

Proposal for an Extensible Solution Architecture for Reproducible and Deployable Robotics Research

> *There are many ways to do design badly, and just a few ways to do it well.*
>
> **— Bass, Clements, Kazman**
> (Software Architecture in Practice)

Several definitions of Software Architecture have been proposed. In [60, 61] which conforms to ISO/IEC/IEEE 42010:2011, the definition is based on the notion of a system. This definition organizes elements in a system by its structure and relationships with a given purpose. The purpose is to achieve goals and separate the system from its environment through a clear boundary. As a result, Software Architecture can reduce complexity by thoughtful modeling and good documentation.

The key aspects of modeling can be listed as follows: decomposition and structuring of components, abstraction and reusability of parts. To achieve this, an architecture has to define components of a system, describe its essential features and characterize the relations between the components. Different views that describe static and dynamic aspects are developed and discussed in this chapter.

This chapter proposes the key elements that define the solution architecture and the used architectural means. First, this involves determining the systems' requirements and quality goals as well as available assets. Second, these inputs are then used to create a solution architecture choosing one architectural style and applying tactics in order to influence the control the response of a quality attribute. In order to test coding increments faster and without being bound to hardware, an integrated simulation environment can be used. This simulation environment should reflect the real world environment as closely as possible, such that the developed coding increments can be deployed "as is" to the real robot. Lastly, the output of this process should be a viable Software Architecture in terms of the identified quality goals. Along the way, many decisions and compromises needs to be made, which should be documented together with multiple views showing different aspects of the system. Each of those views only represent the concerns of a particular set of stakeholders. This focus

on the views is due to the fact that we can only understand and reason about a complex software system's architecture from multiple points of view. The key is to grow a set of architectural decisions that yield an optimal balance of the functional and non-functional forces having weightage on every view.

3.1 Architectural Forces

In this section we show how the system is built in its context and the vision is formulated. Achieving that optimal balance of functional and non-functional forces mentioned earlier requires a certain engineering process to identify actors and systems in its context. The identified actors have to be abstracted into general personas which have a goal of using this system. This goals can be formulated using user stories forming the functional requirements. In order to judge the operation of the system, non-functional requirements can be used to specify criterias. Concluding the forces are the constraints that come from the systems' environment and cannot be circumvented.

3.1.1 System Vision

The aim of this *navigation system* is to provide a flexible and extensible navigation framework for mobile robot bases. Fast deployment to new robotic platforms should be possible with minimal effort. Furthermore, the adaptation of the platform configuration to new simulated or real world environments should be possible without expert knowledge in Robotics or Software Development with C++ or Python.

Interaction with the navigation system should be possible without the user having to install anything but a recent browser. The system should work on any combination of platforms and browsers that are available. The system should be easy to use, such that an user should not be required to have any prior education or skills for using the navigation in a short amount of time. For advanced users that wants to extend the functionality of the framework, a rapid prototyping interface should be provided integrated with all tools needed. These coding increments should be easily testable on the real platform or using a simulator. For faster testing without hardware dependencies, it should be possible to simulate large areas with navigation, exploration and pose accuracy on a powerful GPU equipped machine. A certain degree of freedom in deployment is to be achieved that allows running nodes on the robot platform, if possible or running them on a remote machine. The framework should be easy to setup according to the user's needs for production or development purposes.

To achieve autonomy in flexible environments, a mechanism to update internal representations of the environment is required. Final pose accuracy should be improved to allow robot-robot collaboration and execute

pick and place tasks. A mechanism to store and retrieve semantic poses is required. This mechanism should allow two different pose accuracy requirements. One is the standard non-modified pose that just drives to a goal approximately, while the other pose helps to drive more accurately to a goal pose.

These goals that the navigation system tries to achieve and the values formulated should be reflected in a short and catchy name. Thus, the name *NavAjust* (i.e. shorthand "*nav*" for Navigation and the old english spelling of adjust, i.e. "*ajust*") was chosen, to reflect the flexibility and adaptability of the solution.

3.1.2 Context delimitation

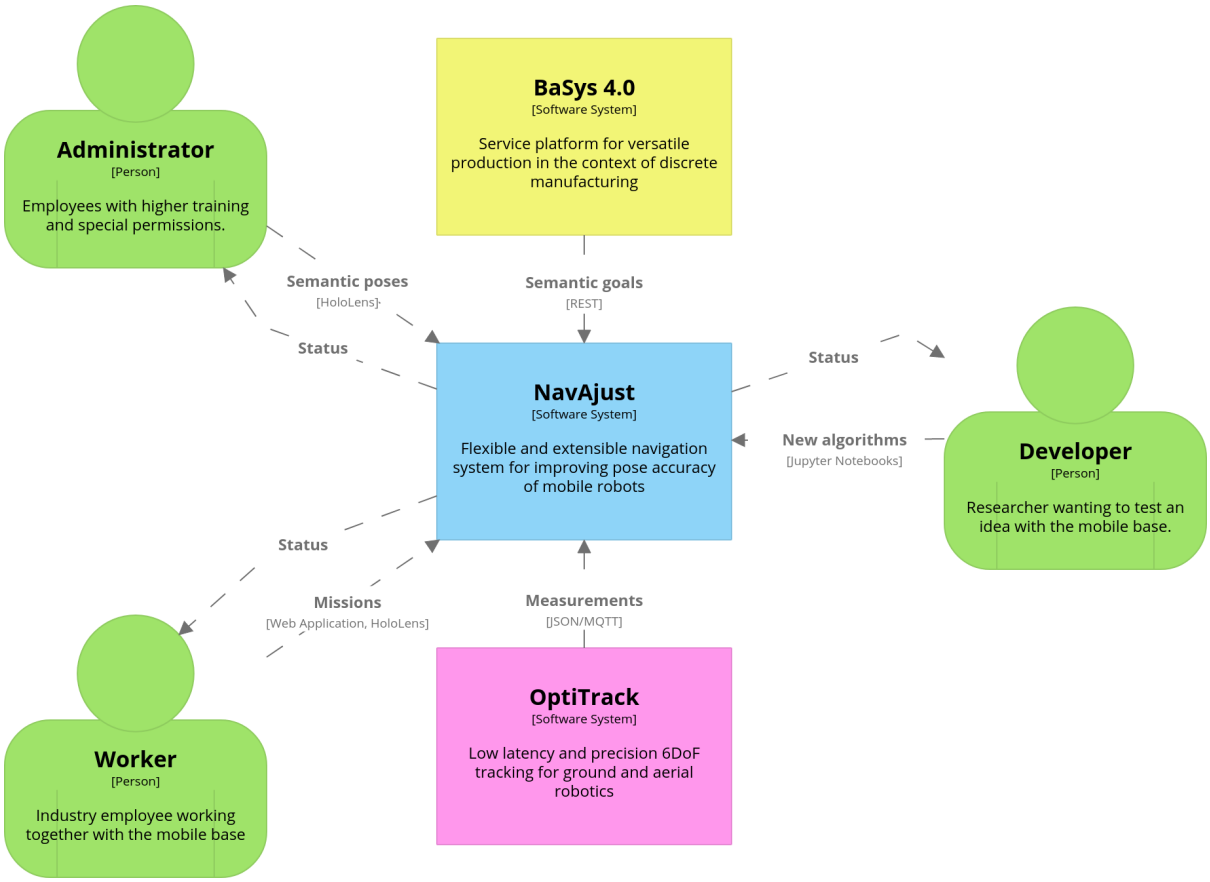


Fig. 3.1.: Context Diagram shows how the software system in scope fits into the world around it

As a means to delimit the scope of the system from its context, the context view in Figure 3.1 helps. It shows how the *NavAjust* Navigation System is embedded into its environment and how it interfaces with neighboring systems as well as with all of its users. The arrows between the users and the neighboring systems depict all events that can enter or leave the system. The system is presented as a black box to focus attention on the environment and show the system interactions with all elements around. Because omissions in the system context are the main source of risks in software projects, a good amount

of time is spent to refine the context view as well as re-iterating through various versions. A broader view on the software system is shown in the system landscape diagram in the Appendix A.3. This system landscape shows the bigger picture of how the system fits into the service platform without focus on the particular software system developed here.

3.1.3 Functional Requirements

Persona Specifications

Needs of different users have to be taken into account in order to determine how the product fits the requirements. These functional requirements are defined by the users of different fields and ages with unequal contexts to help the functional design. In this thesis, we have achieved this goal by creating *Personas* that are fictional characters. These fictional characters represent the majority of potential users of *NavAjust*. The specified personas should answer "who" the system is designed for. The use of persona is handy, because it allows to have a typical image of many real groups of people.

Three relevant Personas have been identified in the context. For every persona, its specifications are listed in Tables 3.1 to 3.3. Each listing contains a goal that is the desired result, the persona wants to achieve. The aim sets the determined course in order to achieve said result. Furthermore, for each persona, the required education and skills are listed below in a general fashion.

`Worker`	
Goals	*Fast and easy mission definition*
Aim	*Specify goal poses to a mobile base. These poses are entered on a simple to use web interface or over a Microsoft HoloLens*[1].
Education	*None required*
Skills	*No common skill set*

Tab. 3.1.: Worker Persona Specification

Firstly, there is the `Worker` who directly collaborates with the platform. This persona does not needs no specific education or skills. The Interface for the worker should be as simple and intuitive as possible. Also, the platform should not interfere with the normal work activities in any way. The platform should ideally be perceived as a work colleague. Secondly, there has to be an `Administrator`, who is responsible for the

[1]The HoloLens is a pair of mixed reality smartglasses developed and manufactured by Microsoft. The first version was released on March 30, 2016. The second version of the HoloLens is announced and already available to Enterprise customers and developers.

platform and its maintenance. This persona has access to more advanced features like defining new semantic poses and customizations. In order to do the critical customizations, some IT education as well as some sytem programming skills are required.

`Administrator`	
Goals	*Customize solution to own environment*
Aim	*Responsible for the maintenance. Has access to advanced features like defining new semantic poses and customizations.*
Education	*Junior IT or Senior IT*
Skills	*System programming*

Tab. 3.2.: Administrator Persona Specification

`Developer`	
Goals	*Rapid prototyping*
Aim	*Works in a robotics related field and proposes improved or innovative features*
Education	*Some robotics related field*
Skills	*Software Development w/o Robotic Operating System ROS, Robotics, Machine Learning ML or Human-Robot Interaction HRI Expert*

Tab. 3.3.: Developer Persona Specification

Finally, there are various kinds of `Developer`'s who proposes improving or adding innovative features. For this persona, all robotics related education profiles that are thinkable that can make a contribution. This could be a robotics expert that can improve the accuracies of the robot at target locations, i.e. a Machine Learning ML Engineer that improves people detection or a Brain-computer interface BCI expert that develops new ways to interact with the platform.

User Stories

A user story identifies the user and their need or goal in a short statement. It specifies who the user is, what they need and why they need it. The user stories formulated in this thesis follow the simple and most common *Connextra template*.

"As a <persona> I can <capability>, so that <receive benefit>."

The following 4 user stories have been identified for the 3 personas defined previously.

- As a ***worker***, I want to specify precise poses for the mobile base to visit, so that it can assist me in tedious tasks.

- As an ***administrator***, I want to define new semantic positions, so that the mobile base can reach it autonomously and precisely.

- As an ***administrator***, I want to customize the system to the new environment, so that the mobile base can reach targets safely and precisely.

- As a ***developer***, I want to quickly test an idea, so that I can prove a concept.

3.1.4 Adressed Quality Goals

The basic statement of the systems capabilities, services and behavior comes from qualities of the system's architecture that goes beyond functionality. Systems are frequently redesigned not because they are functionally deficient, but they are difficult to maintain. The mapping of a systems functionality onto software structures is what determines the architectures support for qualities. A *quality attribute QA* is a measureable or testable property of a system. The QA can therefore be used to indicate how well the system satisfies the needs of its stakeholders. System's functions and quality attributes go hand in hand, as they reinforce each other.

In this thesis, we wanted to build an easy-to-use navigation system for the *Worker* and *Administrator* persona, while allowing developers and researchers from different fields to improve or extend the system. This is important because research in robotics and Artificial Intelligence is progressing rapidly. Therefore, seperate modules for individual behaviors should be easily replaceable or added to the system. In order to test if a replaced or added module improved the quality of the solution, a fast and easy testing is important. Because of varying requirements for accuracy in different environments, it should be possible to test different algorithms and configurations. Software that does not work reliably or incorrectly can lead to significant problems as motivated by [62]. Problems such as harming people or equipment, loosing money and time or damage to company reputation can be the consequences.

Three quality attributes have been selected as goals for the system built in this thesis, i.e. *Modifiability, Usability and Testability*. All quality attributes that are specified with requirements, are unambiguous and testable. A common form to specify the quality attribute requirements from [60] is used for that purpose. This common form specifies the requirements as six-part scenarios using *stimulus, stimulus source, environment, artifact, response and response measure*.

These required quality attributes are achieved using a technique called *architectural tactics*. Such tactics are design decisions that influence the achievement of a quality attributes response. Thus, Tactics directly affect the response of a system to some stimulus. Applied tactics are discussed in the following Section 3.2.4. In the end, a systems design consists of a collection of de-

cisions. While some of these decisions help to control the quality attribute responses, others ensure achievement of system functionality.

Modifiability

As pointed out in the introduction and related work, setting up the development can be a daunting endeavor. This setup is even worse for non-experts in ROS development. Also, change happens. Already the Greek philosopher *Heraclitus of Ephesus* knew that change is central to the universe. If the only constant in the universe is change, then change in software is not only constant but ubiquitous. Changes could be for various reasons, as mentioned in [60], such as adding or changing features, fixing defects, tighten security, or improve performance. Changes can also be made to enhance the user experience or to embrace new technologies and algorithms. Adding support for other robotic platforms that requires porting to a different processor family could also be another reason for change.

All of this change revolves around the quality of modifiability. Planning for modifyability requires considering 4 factors. As change can happen to any aspect of a system, it is important to define *what can change and what is the likelihood of the change*. This is important, as not all potential changes can be planned in the system. The other two factors are *when and by whom is the change made and at what cost*. All of the possible costs involved are measured in the response measure, which is both time and cost consuming.

Modifiability scenarios contain a stimulus and the source of that stimuli. The stimulus portion specifies the change to be made, while the source defines the persona making that change. The artifact represents what is to be changed and the environment defines when that change can be made. As a response, the change should be made, tested and deployed. All of the responses should happen within a response measure that specifies some time constraint.

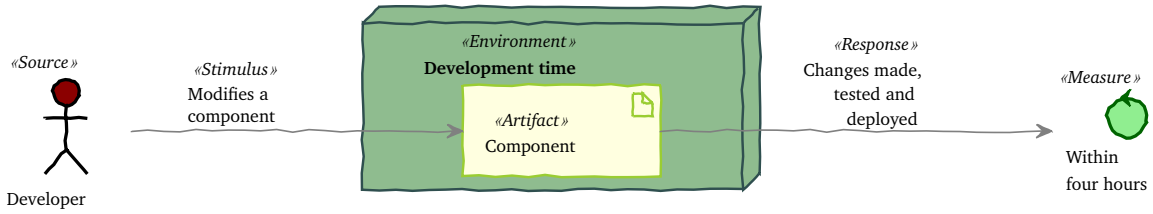


Fig. 3.2.: Modifiability scenario for improving an existing component

To enable modifications as stated in the first paragraph, this thesis specifies the two modifyability scenarios as illustrated in Figures 3.2 and 3.3. The first scenario is defined to control the effort and time needed to get a new developer to make changes to a specific component of the system. This component could be anything from a service backend,

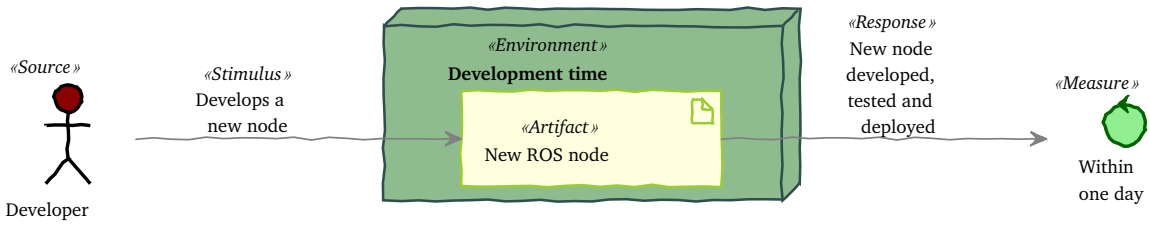


Fig. 3.3.: Modifiability scenario for developing a coding increment as ROS node

database or user interface. This measure also includes the time required for a new developer to get up and running with the required system components needed as well as understand the structure, behavior and communication schemes used in the system in order to contribute. The contribution of a component addition comes in the form of the second modifiability scenario. After understanding and getting the system running in under one hour, it is beneficial that a developer can develop a new component within one day. The scenario specifies a ROS node, as this represents the most time-consuming component in the system to set up.

For the first scenario, a developer wants to modify a specific component of the navigation system. The modification can be made to any component already present in the system. An example would be an improvement to the pose accuracy component that is responsible for the qualitative fine adjustment at the target pose. This change can only be made at development time, as rigorous testing has to be performed before it is deployed. This testing can take place in the simulator but should be confirmed on the real hardware before placing it into production. It should be possible to make all changes, test and deploy them to the platform within four hours. The second scenario is similar to the first i.e., instead of changing a already present component, a new one is added to the system. Keeping everything else the same, it should be possible to develop a new node, test and deploy it to the platform within one day. Both of these scenarios are important in Robotics and Artificial Intelligence, in order to keep up with research as it progresses rapidly.

Usability

Usability is defined as the ease of use and learnability of a system. The degree of usability is quantified by objectives such as effectiveness, efficiency and user satisfaction in a quantified context of use. As such, Usability encompasses a vast area of topics: *Learning system features, using a system efficiently, minimizing the impact of errors, adapting the system to user needs and increasing confidence and satisfaction*. The developed navigation system adresses those areas in the two scenarios sketched in Figures 3.4 and 3.5. First, *learning system features* should be supported by the system to make the task of learning easier for an unfamiliar user. Second, *using the system efficiently* includes thinking

about tactics to make the user more effective. Third, *adapting the system to user needs* includes thoughts about how the system could facilitate the user's task. Fourth, *increasing confidence and satisfaction* that is required to give the user feedback for when the correct action is being taken, to increase his confidence. Lastly, as the error is human, and they can occur when people interact with user interfaces. Errors can be slips or mistakes. Slips happen when a user is on autopilot, and takes the wrong actions. Mistakes on the other hand happen, when a user has developed a wrong mental model of a user interface. Both of these errors should *minimize the impact of user induced errors* by providing correcting and cancelling options.

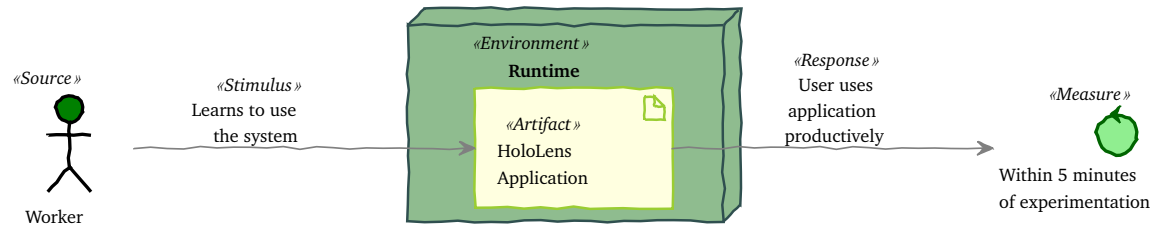


Fig. 3.4.: Usability scenario for using the system productively

To measure the learnability of the navigation system for the worker persona, the scenarios in Figure 3.4 is specified. In this concrete scenario, the worker wants to learn to use the HoloLens application productively. This application is not developed as part of this thesis. Using an intuitive application deployed on the HoloLens, the worker should be able to use the platform productively. This includes using the application to add semantic positions and send goals to the platform. The time for learning should remain under 5 minutes. This short amount of time is important, because it is thought as a support feature should threfore not waste the time of the worker. The environment in which this scenario takes place is at runtime.

The second usability scenario illustrated in Figure 3.5 measures the task time for an administrator that deploys the navigation system in a new environment. Deploying the system in a new environment requires autonomous exploration, creating maps of the environment and defining initial semantic positions. An administrator should be able to configure the system within an hour, to achieve robust navigation results. The administrator only interacts with the system using the web application at run time.

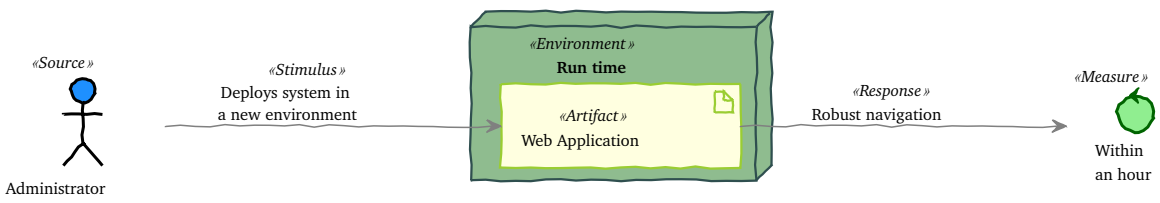


Fig. 3.5.: Usability scenario for deploying the system in a new environment

Testability

Because software is intangible, a test cannot be carried out on a tangible basis. There exist varying quality requirements for navigation solutions. A target pose can be reached with varying degrees of accuracy. For the problems mentioned in the beginning of this section, it is important to test the software system in order to be able to assess the quality and minimize risks. Risks in such cases come in the form of software failure and errors. Unfortunately, testing is generally a random sampling of the software. The subsequent evaluation checks whether the test object meets the required properties. The process of testing encompasses, besides the execution and evaluation, also planning, analysis, design and realization of tests. Even if most of the tests do not reveal an error, it cannot be excluded that there are additional errors. This is especially true for Systems with a certain degree of complexity and amount of program lines.

Testing a system can be done at 4 different levels. On the lowest levels, unit and component tests should check a single component's internal aspects. Isolation is the key in such cases to exclude component external influences. The next level called integration testing, which assumes that components are tested and components internal error states have already been corrected. This is the level at which the scenario in Figure 3.6 is defined. A coding increment is viewed as the contribution to a single service that is part of a bigger software where all individual parts have to function correctly with each other. The goal is to find error states in interfaces and in the interaction between integrated components.

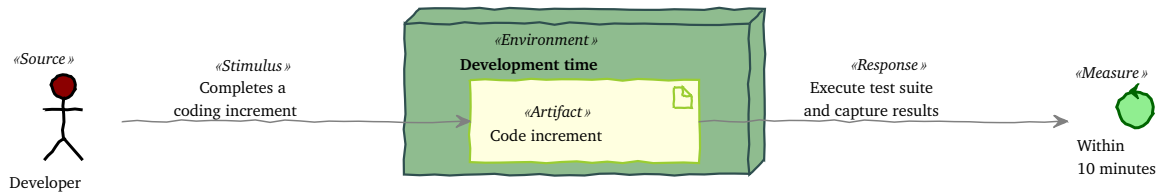


Fig. 3.6.: Testability scenario for executing the test suite

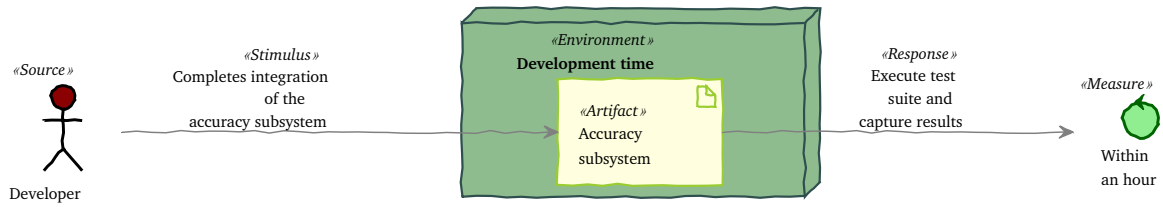


Fig. 3.7.: Testability scenario for executing system tests

As a second testability scenario, a system test for the accuracy subsystem is defined in Figure 3.7. As the system test, this is one level above the integration test from the first testability scenario. As such, it verifies that the subsystem meets all specified requirements. These tests are executed from the perspective of end-users such as the administrator.

Executed by the developer that completed a coding increment and passed the unit and integration tests. This scenario is specific to the accuracy subsystem in a way, that it ensures an automated way of testing the performance of the introduced changes. By completing the integration of the accuracy subsystem a developer can run the automated testing pipeline. This pipeline will run on its own after calibrating initially a goal pose manually. The platform will then randomly navigate to explored areas in the map and try to achieve the pose defined using the algorithm defined in the accuracy subsystem. The captured results of each run are computed, compared and plotted for an easy evaluation of the achieved performance.

3.1.5 Constraints of the Environment

Constraints are design decision that have already been made by the environment in which the system operates and have in general zero degrees of freedom. In the case of the navigation system, the constraints in Table 3.4 have been identified. First, a service-orientation is required by surrounding software systems and personas. Second, the navigation system is to be build using hardware that is already available to save time and resources. Third, a markerless setup is required to support deployments in flexible environments. This requires sensors that perceive the egocentric position and velocity. Egocentric perspective transformations involve a self-to-object representational system. This system represents the location of objects in space relative to its own body axes. Fourth, all platforms that provide some sort of range data and accepts velocity commands are to be supported. New platforms are required to work with the navigation system with minimal configuration and deployment effort. Lastly, the state-of-the-Art Robotic Framework ROS has to be used together with the respective device manufacturer drivers and ROS wrappers. A ROS wrapper is a design pattern, which exposes a ROS interface for an underlying driver. If the device manufacturer does not provide a ROS wrapper for their drivers, a third party wrapper can be integrated. If everything else fails, a ROS wrapper has to be written from scratch for the device at hand. Fortunately, there is a lot of documentation available[2].

Constraints	
Service-Orientation	• System should offer various services to other systems and users • Easy to use by other developers
Time & Resources	• Use hardware that is available

[2]Documentation on how to write an Object-Oriented C++ ROS Wrapper can be found under https://roboticsbackend.com/create-a-ros-driver-package-introduction-what-is-a-ros-wrapper-1-4/

Constraints	
Markerless	• Use sensors that perceive the egocentric position and velocity
Multiple Platforms	• Support compatible mobile robot platforms • Working solution with minor configuration and deployment changes
ROS	• State-of-the-Art Robotic Framework • Robot manufacturer provides driver and a ROS wrapper

Tab. 3.4.: Constraints of the Environment

3.2 Solution Architecture

In this chapter, the solution architecture for the navigation framework is presented. This is done by first specifying the setting for the design of the navigating agent defining the performance measure, environment, actuators and sensors (PEAS) and risks associated with the agent task. The design of a navigating agent requires an architecture, which is appropriate for the task. The task requires a mix of deliberate and reactive components. This requirement leads to the introduction of the layered Mobile Base Task Control Architecture model that is developed in this thesis. The general idea of this model is to combine the more low-level reactive components with higher level and more intelligent deliberate components. To support this design, control architecture and gain combination flexibility of components with dynamic reconfiguration and isolated components an *Micro Service* style has been chosen. The following sections introduce this style with its benefits and drawbacks and discusses the tactics used to help achieve the quality goals. All central decisions made in designing the architecture are documented in the subsequent sections. Finally, the technologies used are presented in the last section.

3.2.1 Robotic Agent and Environment

Defining the agent, its task and environment properties following [14] allows a classification of the difficulty of the task in the environment. The environment specification is used to determine the types of agent programs that are needed to deal with the environment. This thesis focuses on mobile robots moving about in their environment using wheels. The agent's task can be defined as follows:

> "*Navigate to different target poses accurately in* (x, y) *and yaw* $\angle\theta$ *while giving way to human coworkers and avoid collisions with robot coworkers and other obstacles.*"

A function that implements the mapping from percepts to actions is the agent program, whose design is the job of Artificial Intelligence AI. A complete agent is made up of an agent program and an architecture, which provides the run-time. The architecture is responsible to make the percepts from the sensors available to the program, run it and feed the actions of the program back to the effectors.

The proposed solution in this thesis is developed using the *Mobility Base* from Dataspeed as depicted in Figure 3.8, but ideally should work with any holonomic or differential drive robot. The robot platform should, however, provide a ROS wrapper for its device drivers. Because of the real-time constraints of navigation, it would be beneficial if the robot platform provides access to the onboard computer. This access is important, in order to deploy the agent programs locally and avoid network latencies.

Fig. 3.8.: Mobility Base with and without Baxter Robot mounted on top

The navigating agent under discussion, the *Mobility Base for Baxter*, serves as a mobile base for the *Baxter Robot*. This top mounted robot is available separately as an industrial or research robot. Together with the *Mobility Base MB* it has the potential to be an autonomous mobile robot. For the objective of this thesis, the *Baxter Robot* was removed from the base. The MB has an on-board Intel *Next Unit of Computing NUC* that runs on a fully featured *Ubuntu 16.04* Linux Distribution. As effectors, the MB has 4 mecanuum wheels that allow movements in all directions at all times. Independently of the lateral movements, it can also rotate around its own axis. The viable sensors on the MB platform are the 360° 2D LiDAR, 8 bumpers and the floor facing camera for reading QR codes. A *inertial measurement unit IMU* reports a body's specific force and angular rate using a 3-axis accelerometer, gyroscope and magnetometer. While the MB includes an IMU, the magnetometer data is not published, and accelerometer is set to "*Not a Number NaN*".

The data is not published, because a problem with the IMU data has been identified. Therefore, for the time being neither specific force nor angular rate can be used to enhance odometry.

Task Properties The task of the agent has the properties of being a *single achievement goal* with *weak committment*. All goals have equal utility with constraints on the goal achievement. These constraints include that no harm or damage is done to the collaborators and movements should be restricted to goal poses in allowed and reachable areas. Because the agent is only weakly commited to its goals, it needs to be able to decide when to give up trying to reach a goal pose.

As for the nature of the environment, a clear trend in industry towards a flexible and dynamic shop floor can be observed. This workplace will include *Human-Robot Collaboration HRC* as well as *Robot-Robot Collaboration RRC*. This nature leads to a difficult task environment to percept.

Task Environment As only the surroundings of the base at a height of $30\,\mathrm{cm}$ are seen in 2D horizontally, the task environment is only *partially observable*. This leads to a requirement of having an internal state of the environment.

Working cooperatively with other mobile and non-mobile robot agents and avoiding collisions, maximizes the performance measure of all agents in this *multi-agent* setting. Although, it could become partially competitive when multiple robots try to reach the same target pose accurately. Social abilities of the agent should be investigated as a possibility to mitigate problems arising from competing agents.

Uncertainty lies in the environment, as well as in between the agent's actions. These uncertainties come from sensory errors as well as from wheel slippage. Because the probabilities are not known, the environment is classified as *non-deterministic*. A notion of uncertainty is therefore required for representation and reasoning capabilities.

Decisions which are taken at a certain point have an influence in future decisions. As most decisions only result in time or energy lost, they are not fatal and can be neglected. Therefore, the environment can be seen as episodic, where the outcome of an action only depends on the current state and action. The agent receives a percept in each episode and performs a single action. However, while the agent calculates the next action to perform, the *dynamic* environment around can change. Therefore, the time it takes to choose an action to perform needs to be fast enough and requires reactive safety mechanisms.

Speed and location of the agent and its collaborators sweep over time through a range of *continuous* values. Possible movement actions of the base are infinite. This requires abstraction in order to get a simplified model of the environment. For 2D navigation, this means creating a cellular decomposition of the environment. Such a decomposition discretizes the environment as a grid of cells with specific sizes between 5 and $50\,\mathrm{cm}$. Each cell holds a value between $[0, 100]$ that represents the occupancy proba-

bility, else -1 for the unknown cases. Decision making is then realized via various software layers, each of which is more (or less) explicitly reasoning about the environment at different levels of abstraction.

Actions Performing actions has resource cost attributed denoting the *costs of actions*. The utility of an action is the utility of the state which results from the action. If an action has maximum utility it is called *correct*. An action is therefore *optimal* if it is correct and there is no other correct action with lower cost. The actions in the task environment are *fallible* as they are not guaranteed to produce their intended effects when executed. Also, the *utilities and costs could sometimes be contrasting*. Imagine that there are two paths leading from a current location to a defined goal: a shorter path, which has a opened or closed door along the way, and a longer path without any obstacles. In this case, the longer path would have greater utility, but higher cost. Whereas the shorter path could have lower cost, if the door is open. In the latter case, the utility is lower than going the longer route. Also, to follow a friendly *humanly navigation* it considers the 6 rules for successful HRI introduced in section 2.3. Therefore, the agent needs to be able to choose between alternative courses of action. The agent communicates with other robots indirectly by its actions in the environment.

Performance Measure For measuring the performance of the agent, the following aspects have to be evaluated. First, the navigation should avoid all collisions, be interference-free and give human priority. Second, goal poses should be reached with a positional error below $1\,\mathrm{cm}$ and rotational error below $1°$. Third, the calculation and execution of paths should be seamless, while the movement speeds in HRC applications are irrelevant. Finally, the navigation should be robust to scene dynamics such as moving objects and people. All these aspects were considered while working on this thesis but only the second one is evaluated in more detail in the following chapters.

Sensors The success of real robots also depends on the design of sensors and effectors that are part of the agent architecture. The interfaces between the robot and its environment have to be appropriate for the task. Various active and passive sensors exist that sense the environment, the robot's location, or its internal configuration. The MB comes equipped with a range sensor that uses laser beams and a special 1-pixel camera, which is directed around $360°$ using a complex arrangement of rotating elements. This sensor is frequently also called scanning LiDAR, which is short for *Light Detection and Ranging*. Scanning LiDARs are (generally) accurate at longer ranges and perform well in various light settings.

Augmenting the dimensionality using *time of flight ToF* or *structured-light 3D* camera technologies provides a more accurate description of the environment at closer ranges. For 2D navigation purposes, the resulting 3D point clouds are then down-projected into 2D. This accounts for obstacles, which are above or below the 2D LiDARs horizontal scan line. Integrating this information makes the agent potentially more autonomous. Additionally, more robust people detection is

possible by fusing the LiDAR and Camera data. Also, better Human-Robot Interaction HRI techniques such as pose detection and ultimately gesture recognition imaginable using cameras.

Range sensing based on physical contact is made possible by tactile sensors, such as bump panels, whiskers and skin that is touch-sensitive. The MB has two Bumpers equipped per wheel on each side with a total of 8. Such sensors can only be deployed for sensing objects close to the robot.

Information about the robot's own motion can be obtained by proprioceptive sensors such as shaft decoders and inertial sensors. Shaft decoders count the revolution of motors in small increments which can be combined over all wheels to measure the distance traveled. This measurement of distance traveled is often called *odometry*. Because of wheel slippage and drift, these measurements are accurate only over short distances. Unfortunately, the MB does not provide any such measurements. Therefore, the travelled distance is approximated by calculating movements based on executed velocities. Fusing these measurements with estimated odometry from consecutive planar distance readings (based e.g. on laser scans or down-projected 3D point clouds) can yield better motion estimates. This method can also be used if the base odometry of the platform is inaccurate.

PEAS	
Agent	*Humanly navigating base with precise goal pose localization*
Performance Measure	*Safe, accurate, timely and robust navigation*
Environment	*Flexible floor spaces with other mobile robots and human workers, tables, benches and fixed machines*
Actuators	*Wheel drives with at least 3 effective DoF and 2 controllable DoF*
Sensors	$360°$ *2D LiDAR*

Tab. 3.5.: PEAS: Task environment specification for the Mobility Base

Effectors Mobile robots move their base in the environment by means of effectors. Omni drive robots can move at any heading and turn at the same time. This motion is characterized as 3 degrees of freedom DoF, where each independent direction in which the robot can move counts as one degree. For omni drive there are 2 DoF for its (x, y) planar location as well as 1 DoF for its angular orientation, known as *yaw*. The reference of the planar location of the robot is defined in a right-hand coordinate system, where positive x values lie in front and positive y values lie to the left of the robot. For differential drive robots, as they have a limited mobility in y direction, they have only 2 controllable DoF. However, because it is possible to maneuver the body to any (x, y) point, in any orientation, the kinematic configuration of a differential drive robot is 3-dimensional on a planar surface. Therefore, differential drive robots also have 3 effective degrees of freedom. A robot is called *holonomic* if effective DoF equal the controllable DoF. This makes differential drive robots *nonholonomic*. The degrees of freedom define the

kinematic state known as *pose* of the robot. In order to make the proposed navigation solution in this thesis as general as possible, the movements are restricted to the x-axis and rotations. This restriction is also beneficial for the MB, as the motion estimates using only the velocities, tend to increase the localization error in y-axis direction.

Risks There are multiple risks that can be identified in the task and environment. In order of priority, the highest risk is in relation to the horizontally scanning laser that only perceives obstacles at a height of $30\,\mathrm{cm}$. This leads to the robot failing to perceive a smaller object or a Table that is at a certain height. Also at risk could be a human foot that is longer on the ground than at the lasers height. Possible second risks are reachable, but occluded areas of the MB when approaching a junction or corner. These areas are regions of potential collisions with dynamic obstacles like walking humans or other mobile robots.

Appropriate Architecture: Requirements	
Task level	• *Ability to decide when to give up on trying to pursue a goal* → **Weak committment**
Percept level	• *Internal state to keep track of current state of the world* → **Partially observable** • *Minimize time it takes the agent to choose which action to perform* → **Dynamic environment** • *Representation or reasoning capabilities require notion of uncertainty* → **Non-deterministic/Stochastic** • *Constraints: social abilities of robotic agent* → **Multiple agents**
Action level	• *Environment monitoring to see if action succeeded* → **Actions are fallible** • *Needs ability to choose between alternative courses of action* → **Costs and utilities are contrasting**

Tab. 3.6.: Requirements for an appropriate architecture for the navigating agent

Summarizing the found requirements in Table 3.6 from analyzing the agent task properties and environment. Using these requirements and the following agent definitions, allows designing an appropriate architecture. This appropriate architecture is discussed in the next section.

Agent Types There are 4 types of sub-agents in the solution that are gathered in a hierarchical structure. These sub-agents implement specific capabilities and taken together, create a complete system that can accomplish difficult tasks. First, there are two *Simple reflex agents* that are responsible for safety routines and commands that are given over the WebApp or HoloLens. The safety

agent checks with high frequency (120 Hz) if an obstacle is in the proximity and adapts its speed linearly to that distance. Another simple reflex agent is responsible for executing and checking tasks on the robot platform that a user wishes to perform. Both agents are simple, which means they have limited intelligence and only work if the correct decision can be made on the basis of only the current percept and condition-action rules.

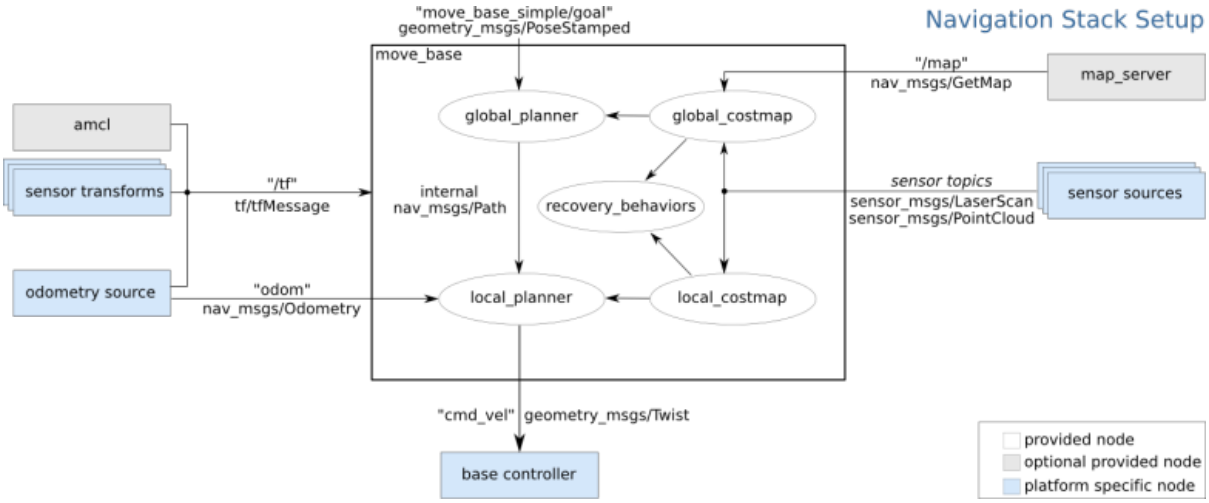


Fig. 3.9.: Overview goal-based agent `move_base` and both utility-based agents `global` and `local planners`. Image courtesy of Open Source Robotics Foundation OSRF downloaded from https://wiki.ros.org/move_base in January 2020.

Second, a *Goal-based agent* tries to successfully direct the robot to a given goal pose or shows that the pose is unreachable by throwing an error. For a correct decision, the agent needs to maintain a belief of where the robot is at all times and the position and orientation information that describes the desired target pose to be reached. The agent then tries to combine the current state description with its model of the world in order to choose actions that achieve the desired goal pose with a tolerance of 5 cm and 10°. This internal model controls the progress of the robot on the path and in case it is stuck, performs recovery behaviors. The actions follow the path suggested by the next two agents responsible for global and local path planning. That setup using the aforementioned goal-based agent to reach a goal pose and the following two path planners are depicted together with their context in Figure 3.9.

Third, a global path planner responsible for determining an efficient travel from the current pose to the goal pose is a *Utility-based agent*. This agent will choose plans with high expected utility from a number of considered alternatives. It has to deal with uncertainty from actuators and sensors, which complicate this planning. As it can come to inaccurate path following, this agent has to replan constantly and consider, that the shortest distance is not always the fastest path (e.g. narrow pathway). Similarly, a local path planner tries to - as closely as possible - follow the path determined by the global planner. While doing so, it uses a rolling window of percepts to locally optimize the robots' trajectory with respect to trajectory execution time, separation from obstacles and compliance with kinodynamic constraints. Because of the dynamic aspects, this planner can deviate from the suggested global path in order

to deal with obstacles or sensor and motor uncertainty.

Finally, two *Model-based reflex agents* are used for calculating the motion model and taking care of the final pose accuracy when reaching a goal. The motion model agent behaves similar to the simple reflex agent while also keeping track of the current state of the world using an internal model. This internal model maps the previous robot state and odometry measurements or estimations to the new robot state. As for the final pose *accuracy agent* (the agent responsible for the fine adjustment) this internal model is a representation of how its sensor readings should look like at a precise target location. Calculating how to transform itself to reach that representation and executing actions to getting to it as close as possible.

3.2.2 Intelligent Agent Design

The humanly navigating MB with precise goal pose localization task requires an appropriate architecture. There are three levels at which the architecture has to succeed in order to reach the targets: *Task*, *Percept* and *Action*.

Mapping sensor measurements into internal representations of the environment, is the task of robotic perception. This perception is difficult because of noisy sensors, and the environment is only partially observable, unpredictable, and dynamic. In order to reason in such environments, the agent must keep track of the current state to the extent that the sensors permit. Maintaining a *belief state*, which represents which states of the world are currently possible, and a *transition model*, the agent can predict how the world might evolve in the next time step. This belief state is updated using the current percepts, a sensor model and probabilities that quantify the degree of belief in *elements* of that belief state.

Filtering (or state estimation) is the task of computing the belief state given all evidence to date. Good internal representations to do that have according to [14] the following 3 properties

1. Contain enough information for the robot to make good decisions
2. Structured so that they can be updated efficiently
3. Internal variables correspond to natural state variables in the physical world

Deliberate Architecture Because the agent needs the ability to choose between alternative courses of action, there is a preference to be defined in terms of costs or utility. Also, a deliberative component is required, that chooses its preference amongst alternatives. Furthermore, planning in such environments is a hard problem, as the initial state of the world is often incomplete or mistaken. The world around the agent is continually changing, also while the agent is busy planning. *Uncertainty* as key characteristic in robotics, arises from partial observability of the environment and from the stochastic effects of the robot's actions. These stochastic effects

are not modeled in the agent. The use of filters also introduces errors that arise from the use of approximation algorithms. This leads to a belief state that is not exact even if the stochastic nature of the environment is modeled perfectly. In case the situation changes in between planning and the change is significant, it requires a replanning. Replanning is also needed in case of a old and incorrect agent's model of the world. The agent therefore needs online replanning of paths during plan execution.

An agent's percepts give rise to goals, which are a representation of a state to be achieved. While an agent deliberates about how to achieve a goal, it also has to manipulate a model of the world and simulate possible courses of action. This then results in a representation of *actions to be performed*. This process requires an accurate world model. Guaranteeing real-time with accurate world models is generally difficult. As reactive control is sensor-driven it is appropriate for making real-time low-level decisions.

Hybrid Architecture Therefore, a hybrid approach that combines reactive control and model-based deliberative planning is more appropriate. Both techniques have strengths and weaknesses. A hybrid approach generally carries the advantages of both. To achieve both advantages, hybrid architectures use reactive techniques at the lower levels of control and deliberative techniques at the higher levels.

Pipeline Architecture Additionally to the layered hybrid architecture for robots, the *pipeline architecture* executes multiple processes in parallel. While the perception components processes the most recent sensor data, the planning and control components base their choices on slightly older data. This is similar to the human brain in that it perceives, plans and acts all at the same time. The processes in the pipeline architecture run asynchronously and all computations are data-driven. This results in a robust and fast system.

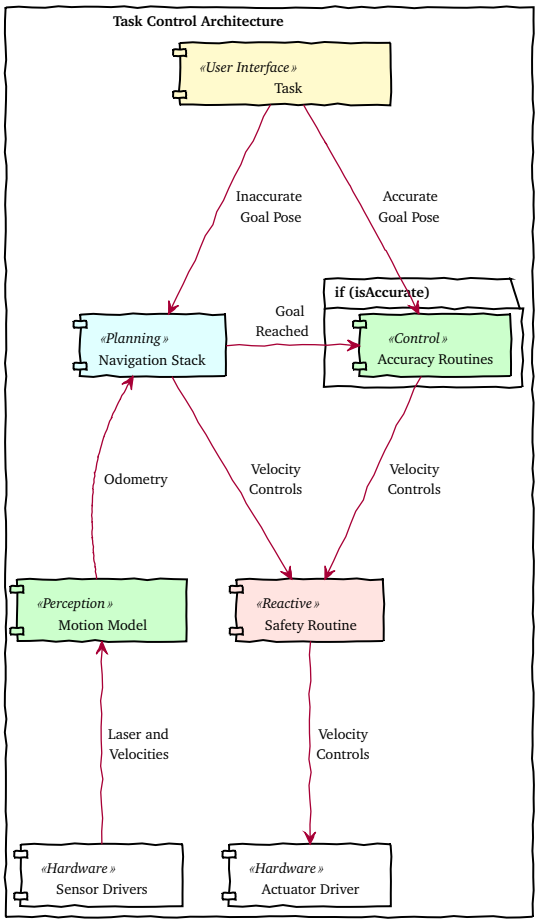


Fig. 3.10.: Mobile Base Task Control Architecture

For the Control and Navigation of the MB a *Pipeline Architecture* is appropriate and consists of 5 layers. Figure 3.10 shows the *Mobile Base Task Control Architecture*. The components used in this architecture have been explained when discussing the agent types in the previous section. The 5 layers con-

tain on the bottom the hardware layer, which provides data from the sensor drivers and accepts direct velocity commands over the actuator driver. The *perception* layer is responsible for updating the robot's internal model of the environment based on the most recent data from sensors. This model aggregated with sensor data are handed to the *planning* layer, which adjusts the robot's plans and turns them into actual controls for the robot to execute. The planning and control layers base their choices on slightly older data. If the *user interface UI* task requested an accurate pose, the accuracy *control* routines kick in once the navigation stack has reached the destination. This accuracy routines only perform fine adjustments at the target. These fine adjustments try to reach a goal pose precisely using the aggregated data and an internal model creating controls. Similar to the planning layer, these controls from accuracy routines are to be executed by the actuators. All controls to execute are checked by a *reactive* routine for low-level and robust behavior for safety. It turns those commands into safe velocity commands that get executed directly by the hardware or servo controllers.

A task that a user enters over an interface of his choice, gets executed by the navigation stack alone or in combination with the accuracy routines. If the task is an inaccurate navigation goal pose, it gets taken care of by the navigation stack alone. If the user wants to save a new accurate semantic goal pose, the accuracy component will update its internal models. When the user then asks to drive to a accurate goal pose, first the navigation stack will reach the goal pose inaccurately and will trigger the accuracy component with data to achieve that goal.

3.2.3 Micro Services as Architectural Style

For the architectural style, *Micro Services* has been chosen, where every service is an isolated component. That is, every ROS node, API, Web Application, Proxy is an isolated container. All containers include per definition all files, environment variables, dependencies and libraries necessary to run the service. Thanks to this property and network virtualization, it is possible to have a near identical setup for deployments in real world as in simulation. Using *software-defined networking SDN* container networks enables defining entire network of containers that can be migrated between platforms or clouds, without modifications. Such networks can even span multiple platforms. These containers can be individually combined and dynamically reconfigured while the services are running. The dynamic reconfiguration of running services can be done without damaging other nodes and allows to add, remove or extend functionality easily.

Following the pragmatic approach of [63] the micro services architectural style, structures an application as a collection of services that are

- highly maintainable and testable
- loosely coupled
- independantly deployable
- organized around business capabilities

- owned by a small team

Benefits This style has four main benefits that enables the continuous delivery and deployment of large, complex applications. First, it leads to improved *maintainability*. Because the services are small, they are easier to understand and change. Additionally, these services can easily be replaced with equivalent components. Second, these small services are faster to test individually, which leads to an overall better *testability*. Third, an independent deployment of services enables better *deployability*. Lastly, the organization of development effort around multiple, autonomous teams is easier, as each team can develop, test, deploy, and scale their services independently. There are additional benefits due to the small size of the services and the individual components being easier for developers to understand. For instance, a developer is more productive, because his development environment is faster and the application itself loads faster, and speeds up deployments in itself. Due to the isolation of each service, a better fault isolation and elimination of long-term commitment to technology stacks are achieved. A fault isolation improvement is achieved e.g. if a service is affected by a memory leak. In such a case only that isolated service is affected, leaving all other services intact. Developing a new service or improved version of a service leaves a developer the choice of picking the technology stack most suited for the task.

Drawbacks Unfortunately, the micro services architecture also has several drawbacks. There is additional complexity that developers have to deal with when creating a distributed system. The additional complexity comes from different aspects of inter-service complications. Developers need to implement inter-service communication mechanisms and deal with partial failure. Dealing with requests and testing interactions that span multiple services is more difficult, and requires careful coordination between the teams. There is also no explicit support by developer tools or by any integrated development environment IDE for developing distributed applications. The deployment complexity is also a drawback, where operational complexity gets added to deploying and managing such a distributed system. There is also increased memory consumption compared to monolithic applications. For example, when a micro service architecture replaces N monolithic `Java` application instances with $N \cdot M$ services instances, it requires M times as many `Java Virtual Machine JVM` runtimes, provided that these services run directly on the host system. If each service runs on its own virtual machine VM, the overhead is even higher.

Challenges Determining if using a micro service architecture is the right approach, is one challenge in the beginning. It is especially a challenge in the beginning, because the problems that this pattern solves are not present at that point. Using a elaborate and distributed architecture generally slows down development in the early stages of development and using funcional decomposition might make it more difficult to iterate rapidly. Later on, this functional decomposition makes it easier to scale the services horizontally. Scaling a services horizontally con-

sists of running multiple copies of the service behind a load balancer. The investment of setting up a continuous delivery and deployment pipeline has to be considered as well. This will pay off later on by accelerating software development.

The second challenge using the micro services pattern is the art of decomposing the solution into “micro” services. Two strategies described in the book [63] that helped decomposing the navigation system into micro services is the *decomposition by verbs or nouns*. The former uses particular actions to define services, such as a `Mapping Service` that records in detail the spatial distribution of the environment. Latter defines a service that is responsible for all operations on entities of a given type. Such a decomposition by noun is a `Known Pose API` that is responsible for managing accurate and inaccurate poses. Ideally, all services should have only a small set of responsibilities. This is also widely known as the *Single Responsibility Principle SRP*. This principle states, that a class should only have one reason to change. The same principle makes sense to apply to service design as well. For ROS nodes, this decomposition can be easily defined that each node should be a seperate service.

Maintaining data consistency accross services can be a difficult task. To ensure loose coupling, each service has to have its own database. In this case, a service has to publish an event when its data changes, which interested services consume and update their data. To reliably update data and publish events in such cases, there are patterns such as *Event Sourcing* or *Transaction Log Tailing*. Another issue that arises from this architectural style are queries, which need to retrieve data that is owned by multiple services. Such challenge requires either pattern like *API Composition* or *Command Query Responsibility Segregation CQRS*. As there are no such challenges in the current set of services, the reliable update and querying accross multiple services are not discussed further here.

3.2.4 Applied Tactics to Achieve the Quality Goals

Section 3.1.4 characterized a number of quality attributes the system should achieve with the help of scenarios. This characterization allows to bring forth and measure quality requirements but does not provide understanding of how to achieve them. Achieving the three system qualities *modifiability*, *usability* and *testability* requires deciding on and applying a combination of tactics. The chosen tactics are used to create a design using design patterns, architectural patterns, or architectural strategies. Such patterns and strategies implement a collection of tactics. The chosen tactics will then guide the architectural decisions. This section can be seen as connecting the quality attribute requirements with the architectural decisions described in the next section.

Tactics are design decisions that influence the control of a quality attribute response to a stimuli. A tactic focuses on a single quality attribute response. The tactics used in this

thesis come from [60], which provides a categorization and many general tactics. In order to be applied, these have to be refined in order to make each tactic concrete. Additionally, because applying tactics to a architecture depends on the context.

Modifiability Tactics

Making changes to software often involves cost and risk. Changes can occur to any aspect of the system with varying likelihoods. The art lies in deciding which changes are likely and should thus be supported. Modifiability tactics help in controlling the complexity of making changes, as well as the time and cost to make changes. Thus, as a change stimuli enters a system, tactics control that these anticipated changes are made withing the measures defined. Figure 3.11 gives an overview of the applied tactics in this section.

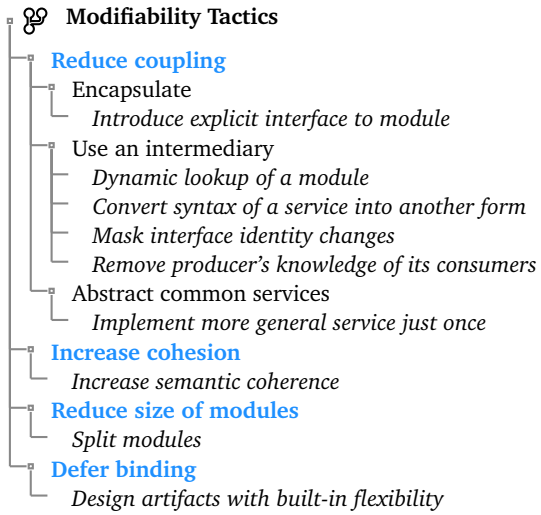


Fig. 3.11.: Tree overview of the applied modifiability tactics

Reduce Coupling Change that only affects one module is easier and less costly than if it affects multiple modules. However, if responsibilities of modules overlap then a single change could potentially affect them both. A overlap measurement of probability that such a modification propagates to another module is called *coupling*. A high percentage in this measurement is an enemy of modifiability. Multiple tactics to reduce coupling have been implemented in the architecture. First, *encapsulation* was used to provide all modules with an explicit application programming interface API that did not have one already. Most modules running ROS nodes already define such a API. ROS nodes and other modules that were contributed as a result of this thesis received an interface abstract with respect to the details of the module that are likely to change. This abstraction is a means to hide those details.

Second, *using an intermediary* contains a lot of concrete implemented sub-tactics that are meant to break dependencies. Using a service-oriented architecture adds the requirement for a directory service intermediary, which discovers services by dynamic lookup. This dynamic lookup of services enables the location of a service to be changed at runtime without affecting clients. Going in and out of the ROS Core System is a ROS Bridge, that uses an event-driven approach to *convert the syntax of the services* running inside the ROS environment into forms that are assumed on both sides. Using such a bridge prevents changes in the ROS environment to propagate to the outside. A dependency of a module on an *identity of an interface* of another module can be *masked* by a broker. This broker can make the connection between modules, such that if the identity of a module changes, the other module

can remain unchanged. Wherever possible a producer's knowledge of its consumers was removed using a publish-subscribe intermediary. Getting measurements from the OptiTrack system In the *ROS Core System* of the navigation system that runs all ROS nodes in an isolated environment, strongly typed ROS messages over ROS topics are used. Using such an intermediary is the standard when using ROS.

Lastly, common services have have been outsourced to a general form. The `Known Pose API`, which provides a service for creating, retrieving and deleting 6D poses in free space. This service is used by nodes inside and outside of the ROS environment as well as in the various UIs. Other planned common services beside the Known Pose API can be seen in Figure 3.16 in the Appendix and have not yet been implemented. This includes abstracting `Mission API`, `Teleoperation API`, `Mapping API` and `Exploration API`. This reduces the cost of modifications, as changes are occurring just in one place.

Increase Cohesion The measure of *cohesion* reflects the strength in relatedness of responsibilities a module. This measure consists of determining the probability that a change scenario that affects a responsibility will also affect other responsibilities. A low probability in affecting multiple responsibilities signifies high cohesion and vice-versa. In order to *increase semantic coherence* there are two goals to achieve in all modules. First, all functionality embedded in the module accessed through its interface, have much in common. Second, the functions inside a module carry out a small number of related activities, by avoiding unrelated sets of data or data that is not fine grained. Data granularity is the size in which data fields are subdivided, where fine granularity means multiple fields. This fine granularity leads to a bigger overhead for data input and storage and manifests itself in a higher number of objects and functions. The fine granularity however helps the flexibility of data processing in treating each data field in isolation if required.

There are 7 types of cohesion in [64] that can be ranked on a scale from least desirable (1) to most desirable (7) cohesion types:

- Coincidental cohesion
- Logical cohesion
- Temporal cohesion
- Communication cohesion
- Sequential cohesion
- Functional cohesion
- Data cohesion

Those types of cohesion come from various sources. *Coincidental* cohesion is the least desirable, as it means that there are multipl elements in the same module for no particular reason. *Logical* cohesion happens if elements perform logically related tasks accross modules. *Temporal* cohesion requires elements of different modules to be used at approximately the same time. *Communication* cohesion occurs if elements share the same *input or output I/O*. *Sequential* cohesion is when elements are required to be used in a particular order. *Functional* cohesion requires elements to cooperate to carry out a single function. *Data* cohesion is when elements cooperate to present an interface to a hidden data structure. These seven points are not ranked on a linear scale. Applied to a system design, the first three constitute low and generally

unacceptable levels of cohesion. This low level of cohesion suggests a poor and costly design. Having functional or data cohesion is considered the most desirable type of cohesion for a software module as it is the highest and superior. Having cohesion in between those is considered moderate and acceptable cohesion. There are cases where communicational or sequential cohesion is the highest level of cohesion that can be attained under the circumstances.

Estimating the degree of cohesion within modules can be achieved by writing a brief statement in one sentence of a module's purpose and performing 4 tests, suggested by the inventor of the software quality metrics, Larry Constantine. If that sentence

1. describes the purpose of the module as a **compound sentence**
    - module probably performs more than one function
    - probably has *sequential or communicational* binding
    - could also mean less cohesion through *temporal, logical, or coincidental* binding
2. contains **words relating to time**
    - module has probably a *sequential or temporal* binding
3. has a **predicate** that does not **contain a single, specific object following the verb**
    - module probably *logically bound*
    - e.g. "edit all data" has logical binding; "edit source data" may have *functional* binding
4. contains **words such as "initialize" or "clean up"**
    - module probably has *temporal* binding

All existing modules developed in this system and modules that will be added in the future, have to be described using such a brief statement sentence and be evaluated using these 4 tests. The details of estimating the cohesion of the current solution can be found in the Appendix A.3 and reveals some sequential and functional binding presence. While the sequential binding is moderate cohesion, it could be the highest attainable in the circumstances.

Reduce Size of Modules If a module includes a great deal of capability, modifying it will likely have high costs. Reducing the size of modules should reduce the average cost of future changes. Using the micro services style where every module has only one responsibility as stated by the single responsibility principle means that modules tend to be small in size.

Defer Binding Time Letting computers handle changes as much as possible will reduce costs of making changes. Therefore, designing artifacts with built-in flexibility is cheaper than hand-coding a specific change. There are 4 different times at which values can be bound: *compile, deployment, startup and run time*. The later in that life cycle a value is bound, the better. Supporting such late binding mechanisms is a tradeoff, as it tend to be more expensive to put in place. The art lies in finding the latest possible bind time, as long as the mechanism that allows it is cost effective.

At deployment time, tactics to bind values using *configuration files* was applied. These configuration files allow to set environment variables such as unified resource identifier URI, ports or custom paths. Tactics that have been applied for binding values at runtime include *dynamic lookup and startup time binding*. Dynamic lookup was realized for services using software defined networking and adds deployment flexibility, as services are found at runtime. Startup time binding is similar to configuration files as it allows to overwrite environment variables but at runtime.

Usability Tactics

Usability is concerned with how easy it is for the user to accomplish a desired task. The process of user interface UI design consists of generating and then testing a UI design. Deficiencies in the design are corrected and the process repeated. There is a connection between the achievement of usability and modifiability. Both qualities attributes complement each other, because one of the best ways to make a system more usable is to make it modifiable. Facilitating experimentation with the UI via rapid prototyping is therefore one of the most helpful things to make a system usable. Building several prototypes, let real users experience the interfaces and give their feedback pays enormous dividends. Designing software so that the UI can be quickly changed is the best way to achieve that. Figure 3.12 gives an overview of the applied usability tactics.

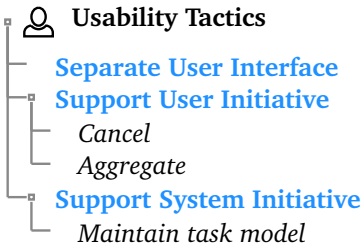


Fig. 3.12.: Tree overview of the applied usability tactics

Separate User Interface To support the rapid UI prototyping approach and facilitate experimentation the UI is separated using a single entry point for clients. This single entry point implements a API gateway that handles requests in two ways. Most requests are simply routed to the appropriate service, while other requests are handled by fanning out to multiple services. Figure 3.13 shows the setup.

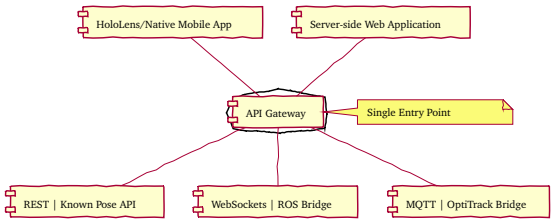


Fig. 3.13.: API gateway that is the single entry point for all clients

Support User Initiative The design of a response for user initiative comes by enumerating and allocating the responsibilities of the system to respond to the user command. First, as a user issues a cancel command, the system has to listen for it. This listening is the responsibility to not be blocked by the actions of what is to be cancelled. Such cancelling option is provided in the architecture for navigation tasks that can be terminated at any time on the path. Second, when a user performs repetitive operations, such as

removing known poses in a large number from the database, the architecture provides the ability to aggregate the objects into a group.

Support System Initiative Getting an idea of what the user is attempting requires a task model that is used to determine the context and provide assistance. The system can then provide useful feedback and can help navigating through the workflow. In the simple web application developed, this tactic is applied to guide the administrator through the setup and accuracy estimation workflow.

Testability Tactics

The goal of tactics for testability is to allow easier testing of software increments. Anything that helps reducing the high cost of testing in the architecture will yield a significant benefit for the reliability of a system. Figure 3.14 gives an overview of the applied testability tactics.

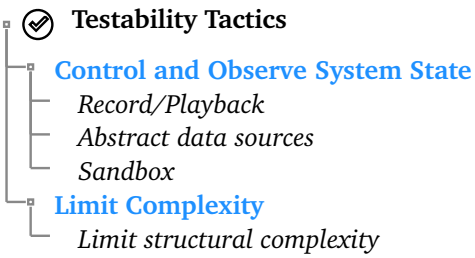


Fig. 3.14.: Tree overview of the applied testability tactics

Control and Observe System State Control and observation is central to testability. In its simplest form a control provides a software component with a set of inputs, let it do its work, and then observe its outputs. The tactics discussed in [60] go further and provide insight into software components to maintain state information and change values in that state information. This state information can be an operating state, the value of a variable or sensor data, intermediate process steps or anything else useful to re-create component behavior.

The operating state space of complex software is large and therefore it is more difficult to re-create an exact state that caused a fault. Making behavior repeatable is crucial to finding a fault that caused a failure. Tactic *Record/playback* defines the recording of the state when it crosses an interface in order to "play the system back", and it that way re-create the fault. A program's state can be controlled easily by its input data and makes it easier to test. *Abstracting interfaces to data sources* such as databases or possibly even to files of test data allows to substitute test data easily, without having to change functional code.

Isolating instances of the system from the real world using simulations enables experimentation that is unconstrained by the worry about having to undo the consequences of the experiment. *Sandboxing* helps testing by providing the ability to operate the system in such a way that it has no permanent consequences or that any consequences can be rolled back. Using a virtualized resource that simulates the imortant aspects of the real world system offers the benefit of building a version of the resource whose behavior is under the develpers control.

Developing software for a robot requires physical access to the hardware as well as

to the defined environment where the robot should operate in. Using record/playback or substituting data sources and simulation in virtualized environments, it is possible to test components quickly and without hardware dependencies. These virtualized environments also set clear test environments and add capabilities to abstract resources such as sensors, time, environment, network, and so on. Virtualized resources such as environments allow e.g. adding obstacles, other robots or people that behave as a scripted by a developer.

Limit Complexity As stated in previous paragraphs making behavior repeatable is important to find the fault that caused a failure. *Limiting structural complexity* includes avoiding or resolving cyclic dependencies between components, isolating and encapsulating dependencies on the external environment, and reducing dependencies between components in general. Having high cohesion, loose coupling and separation of concerns from the micro services style and modifiability tactics helps with testability. The first two limit the structural complexity, while the latter helps achieving controllability and observability.

3.2.5 Key Architectural Decisions

An architecture of a system can be viewed as the result of applying a collection of design decisions. [60] categorizes these decisions in different architectural design decision categories. The relevant categories

1. Allocation of Responsibilities
2. Coordination Model
3. Data Model
4. Binding Time Decisions
5. Choice of Technology

are discussed next. This categories provide a rational divisio of concern but can overlap. The design checklists for the 3 quality attributes that lead to these key decisions can be found in the Appendix A.2.1 to A.2.3.

Allocation of Responsibilities

Decisions that involve the allocation of responsibilities include identifying responsibilities that are important and determine how they are allocated to (non-)runtime elements.

System Subdivision	
Issue	How is the system to be subdivided in order to allow for the specified modifiability, usability and testability qualities?
Alternatives	1. monolith + Easy maintenance of the application during operation + Faster project spin-up time − Single programming language must be used − Cannot achieve required quality objectives 2. micro services + Each module developed in appropriate language + Each module can be switched out at runtime + Greater autonomy in the development of individual services + Able to achieve the required quality objectives − Initially greater cost to set up − Elaborate deployment
Outcome	***micro services***
Rationale	The positive consequences of using a micro services subdivision of the system overweigh the negative ones by far. Initially, there is more effort and time needed to set up the elaborate and distributed architecture. Later on, this investment pays off by allowing • each module to be developed in appropriate language • each module to be switched out or turned on/off at runtime • allow for autonomous development of module Micro services follows the principle of single responsibility, where each service has one function, which it must do well. There should be no overlap in responsibilities between modules (low coupling). The relatedness of modules responsibilities should be high (high cohesion). Following these principles leads to a high maintainability. Because each service is isolated, better testability is achieved. Furthermore, this style dictates to separate the UI in a seperate module, which aids in rapid UI prototyping and facilitates experimentation for achieving greater usability.

Tab. 3.7.: Application subdivision in many subprograms

Coordination Model

Interaction between elements through designed mechanisms is the way software works. Such mechanisms are collectively referred to as coordination model. Decisions about the coordination model come in 3 flavours. First, they include identifying elements of the system, that must coordinate or are prohibited thereof. Second, properties of the coordination are determined. Lastly, the communication mechanisms are chosen that realize those properties.

Coordination Between two Environments	
Issue	The ROS Core System in the NavAjust solution combines and configures existing ROS packages together with contributed packages from this thesis. Packages in that ROS environment reduce coupling using masking interface identity changes, removing producer's knowledge of its consumers and defer binding to as late as possible. Using ROS constrains the use of programming language to C++11 or Python3.6. How should simpler components such as a web application or API, that do not directly depend on ROS coordinate with the ROS environment?
Alternatives	1. use ROS for every module 2. use an intermediary bridge to convert syntaxes
Outcome	***Use an intermediary bridge to convert syntaxes***
Rationale	In order to remove the dependency on ROS for simpler components such as web applications, various APIs and rapid prototyping it is necessary to convert the syntax of a service into another form. Introducing this bridge removes some benefits of loose coupling but also removes constraints on programming languages and provides a universal interface, allowing any client to send and receive ROS messages. Using the bridge also prevents changes from inside the ROS environment to propagate out. The coordination model proposed can be seen in Figure 3.15.

Tab. 3.8.: Coordination between two environments: ROS and non-ROS programs

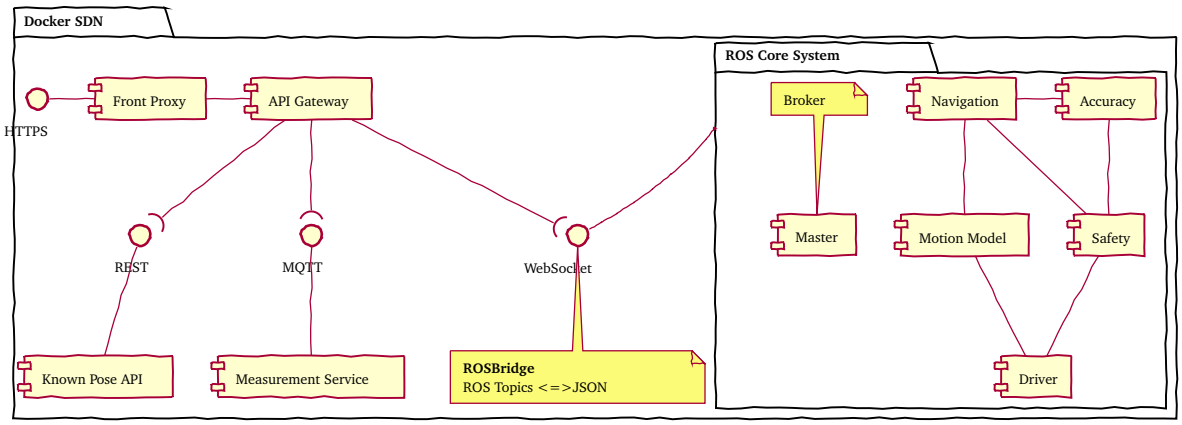


Fig. 3.15.: Coordination model: interactions between elements

Communication Mechanism for Known Pose API	
Issue	The Known Pose API should serve previously recorded known poses in a known environment. What communication mechanism should be used for accessing the Known Pose API?

Communication Mechanism for Known Pose API	
Alternatives	1. Representational state transfer REST API + Based on standard CRUD operations and stateless in nature + Client and Server can be implemented independently + Can be documented using a open standard + Generate mature tools automatically – High communication cost 2. GraphQL + Easy to produce and consume + Contract-driven by nature – Server and clients coupled at the client programming time – Not using HTTP 3. WebSockets + Low communication cost + Allows client and server to talk independently – Stateful protocol – No automatic generation of tools
Outcome	***REST API***
Rationale	GraphQL, a library initially created by Facebook allows to represent all remote data sources as a single domain model via a virtual JSON graph. While GraphQL has some advantages over REST by being easy to use and contract-driven it also neglects some problems of the distributed system, coupling server and clients at programming time. Because GraphQL is not using HTTP, it throws away what the protocol has already solved: scalability, performance, the mechanics of network communication, and many others. While this functionalities can be added to a GraphQL API it leads to bikeshedding[3]. The request/reply interaction of adding, deleting and retrieving known poses is a great fit for REST. Additionally, every programming language is interoperable with the transport layer of REST APIs. Verb-based action i.e. create, read, update or delete operation are executed successfully over HTTP protocol. Message payloads can be easily documented using OpenAPI specification. The documented OpenAPI proposal can be found on SwaggerHub[4].

Tab. 3.9.: API Design: Communication Mechanism for the Known Pose API

[3]Definition: Spending disproportionate time and energy spent over an insignificant or unimportant detail of a larger concern.
[4]Documentation at https://app.swaggerhub.com/apis/kw90/known-pose-api/1.1.0#/AccuratePoseArray

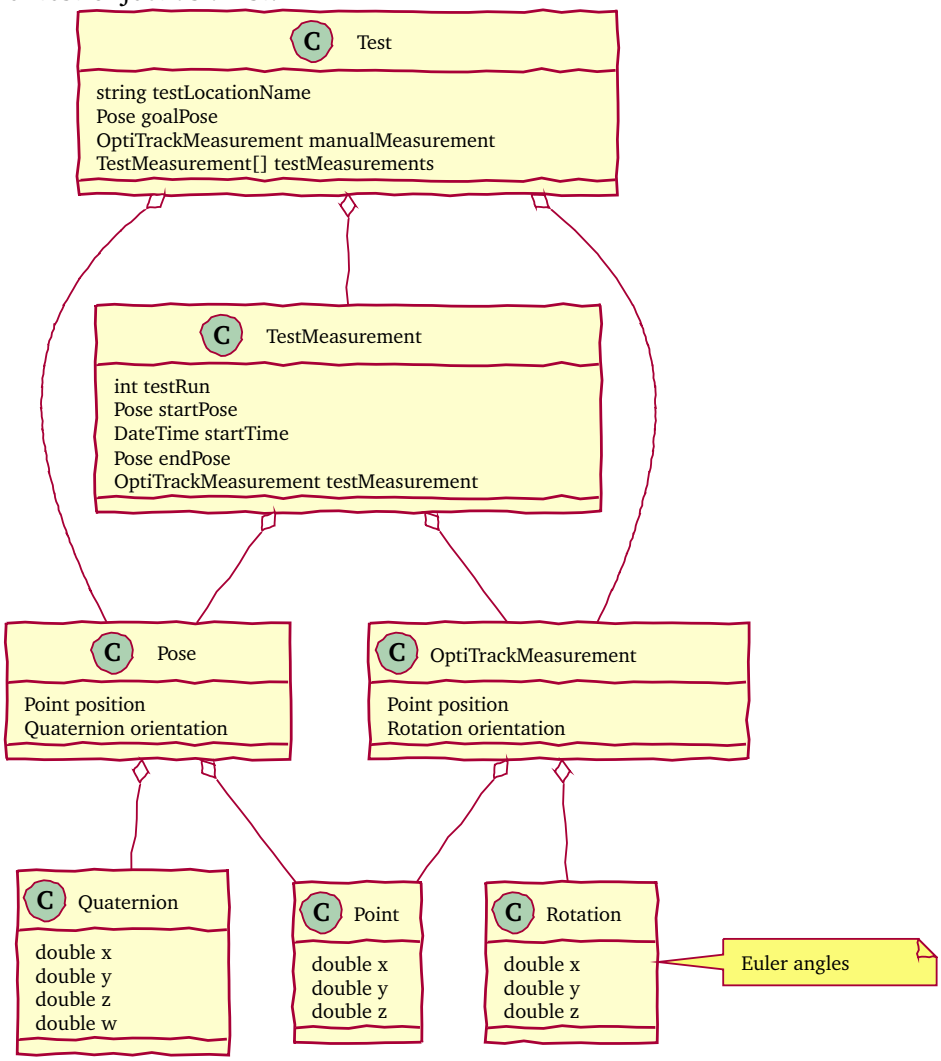
Data Model

Data that is of system-wide interest has to be represented in some internal fashion. These decisions that define the representations and how they are interpreted is the data model. This data model contains decisions on the choice of the major data abstractions, their operations, their properties, the compilation of metadata and how the data is organized.

Representation of Point Cloud Data	
Issue	Point clouds acquired using laser scanners can be represented using numerous formats. Which should be used to describe the reference point cloud saved in the known pose database?
Alternatives	1. PLY - Polygon File Format 2. STL - Stereolithography File Format 3. OBJ - Geometry Definition File Format 4. X3D - ISO Standard XML-based File Format 5. CSV - Comma Separated Values 6. VTK - Visualization Toolkit File Format 7. PCD - Point Cloud Library File Format
Outcome	***PCD***
Rationale	For the point cloud representation `PCD` was chosen in version 0.7 (PCD_V7), because no other alternative offers the flexibility and speed. Also, the PCD format allows a simple `ascii` form where each point is separated on a new line, space or tab. This alternative representation allows to have the best of both worlds: simplicity and speed, depending on what the underlying application needs. This adaptability makes this format perfect for this system.

Tab. 3.10.: Representation of point clouds for adaptability

Representation of Measurement Data	
Issue	Choosing the right format for a dataset in which the data is structured and made available to humans and machines. Ensuring the data can be simply managed and reused needs selecting the right format. The data shows the hierarchy in the following Figure with relationships between tests and measurements and their start, resp. end pose. Start and end times are recorded for every test run. Every test measurement is aggregated into a top level test object as a list.



The captured measurement data can be downloaded over the web application. What data format should be picked for easy access and reuse?

Alternatives

1. CSV - Comma Separated Values
2. JSON - JavaScript Object Notation
3. XML - eXtensible Markup Language
4. UTF-8 - Self Defined Text Lines

Representation of Measurement Data	
Outcome	***JSON***
Rationale	The JavaScript Object Notation is an open data format standard. JSON allows to store and transport data in a lightweight format. For hierarchical data structures, JSON is ideal. The notation is “self-describing” and easy to understand for humans. many modern programming languages include code to generate and parse JSON-format data. Many modern programming languages include packages to generate and parse JSON-format data.

Tab. 3.11.: Representation of measurement data

Binding Time Decisions

Binding Time of Values	
Issue	When in the life cycle should values that are most likely to change be bound?
Alternatives	1. compile time 2. deployment time 3. startup time 4. run time
Outcome	***run time***
Rationale	Binding values that late in the life cycle comes at a cost, but allows greater flexibility. Some ROS parameters can be changed at run time using the built-in dynamic reconfigure tool `rqt`. For other values that are most likely to change, such as • navigation behavior (change planners and configurations) • switch between different fine adjustment algorithms • sensor scan frequency, sampling resolution and frame ID a run time binding using environment variables defined in modules and overwriting them in Dockerfiles or Docker-Compose files is cost effective and allows fast changes of the expected changes.

Tab. 3.12.: Binding time of values that are likely to change

3.3 Architectural Views

The individual subsystems and their components are explained in this Section. First, the building block view gives an overview of the rough division of the system into different microservices. Then the important microservice component are described as white box with black box descriptions of their internal building blocks. Second, runtime views show the behavior of important building blocks as scenarios. Finally, deployment views are shown with different deployment strategies that are possible using this architecture.

3.3.1 Building Block View

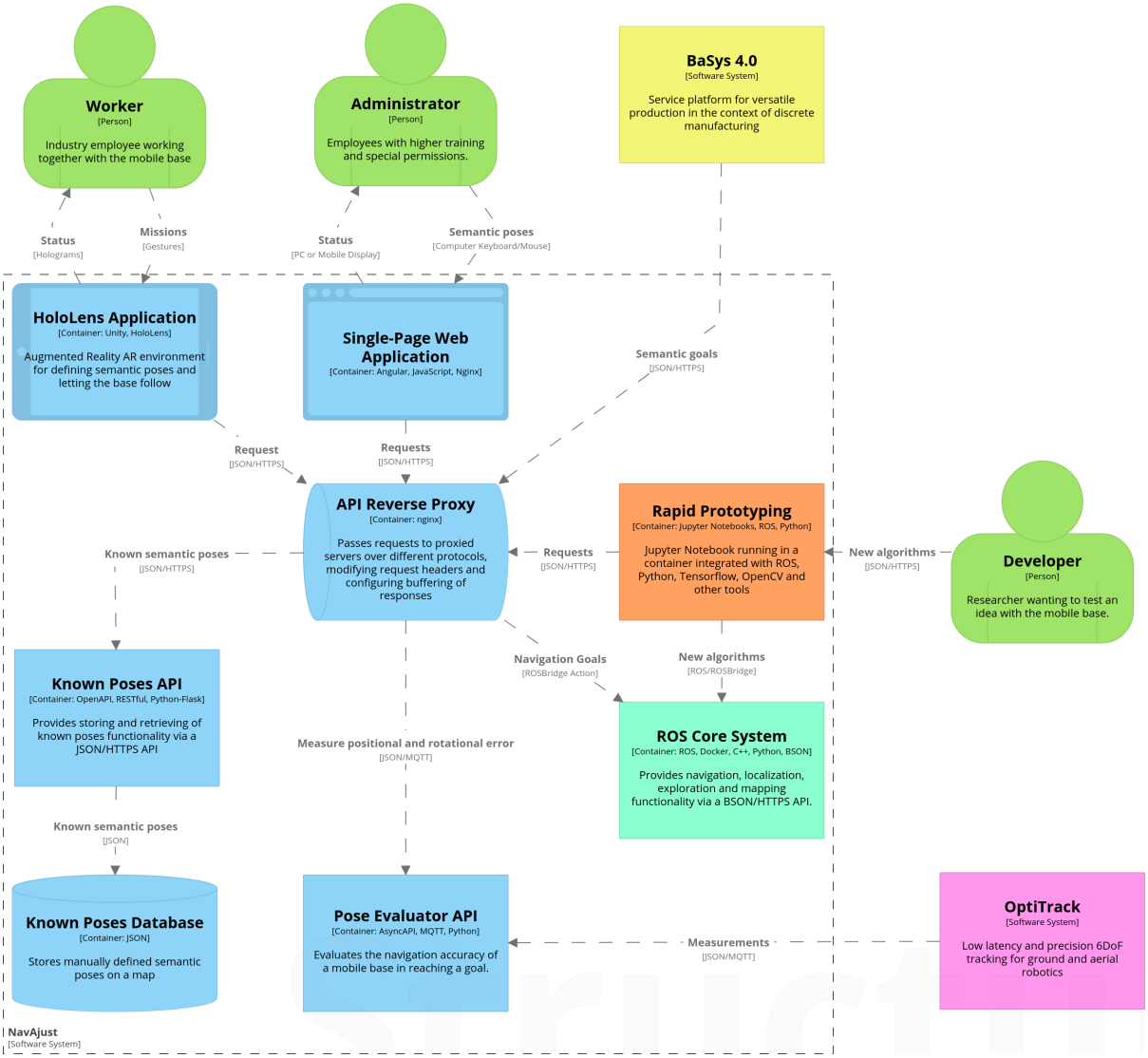


Fig. 3.16.: Container view zooms into navigation system in scope showing the high-level technical building blocks at level 1

In the building block view, the static decomposition of the system into building blocks is shown with their dependencies. The container view in Figure 3.16 zooms into the navigation system as can be seen in the context view in Figure 3.1 of Section 3.1.2. This container view is a slimmed down version of the complete navigation system planned in Figure A.4 in the Appendix and corresponds to the implemented software stack in this thesis. The view gives an overview of the rough division of the system into different microservices and the `ROS Core System` subsystem. The ROS Core System is again made up of multiple microservices, described as white box view.

Component	Purpose	Technologies
`ROS Core System`	Implements the *Mobile Base Task Control Architecture*. Provides navigation, exploration and mapping functionality via a JSON WebSocket API.	ROS Kinetic C++11 Python3.6 JSON/BSON Docker 19.03.5
`Known Poses API`	Provides storing and retrieving of known poses functionality via a JSON/HTTP API.	OpenAPI 3 Python3.6 Flask 1.1.1 JSON Docker 19.03.5
`Known Poses Database`	Stores manually defined semantic poses with a map reference.	MongoDB 4.2.2 JSON Docker 19.03.5
`API Reverse Proxy`	A proxy to prevent the cross-origin problem and control *Cross Origin Resource Sharing CORS*. This proxy is the only interface that the outside world can access. Enforcing a secure HTTPS connection using a valid SSL certificate.	Nginx 1.16.1 Docker 19.03.5
`Rapid Prototyping`	Jupyter Notebook integrated with ROS, Python and any other tool that a developer might use for testing an idea quickly in the browser without tedious installation and configuration of the ROS environment. The component can make request to any service running in the environment.	IPython 5.1.4 IOctave 0.31.1 ROS Kinetic OpenCV 4.2.0 TensorFlow 2.1.0 Docker 19.03.5
`HoloLens Application`	The HoloLens application provides an augmented reality AR environment for defining semantic poses and controlling the navigation using gestures. This component is not part of this thesis implementation.	Microsoft HoloLens Unity 2019.3.0

Component	Purpose	Technologies
Pose Evaluator API	The Pose Evaluator API provides methods to measure the positional and rotational error using a manual reference measurement and current measurements from the OptiTrack system. This component is not part of this thesis implementation.	Async API 2.0 MQTT Python3.6 JSON Docker 19.03.5
Single-Page Web Application	This web application is one user interface component that is used by the administrator. It is its own independent application and is executed in the browser. The application requests data via HTTP from services.	Nginx 1.14.1 Vuetify 1.4.3 ROSLibJS 1.0.1 Docker 19.03.5

Tab. 3.13.: Overview of microservice components and subsystems

Single-Page Web Application

This section gives an impression on the views created in this thesis using some screenshots in action. More screenshots can be found in the Appendix A.5.

GUI Dashboard Figure 3.17 shows the dashboard of the web application. This view contains debugging and monitoring tools such as ROS Topics, ROS Services, ROS Actions and the state of the robot's navigation. On the left hand side, the navigation state is rendered together with the TeleOp Mobility Base TOMB tool for controlling the base with the keyboard.

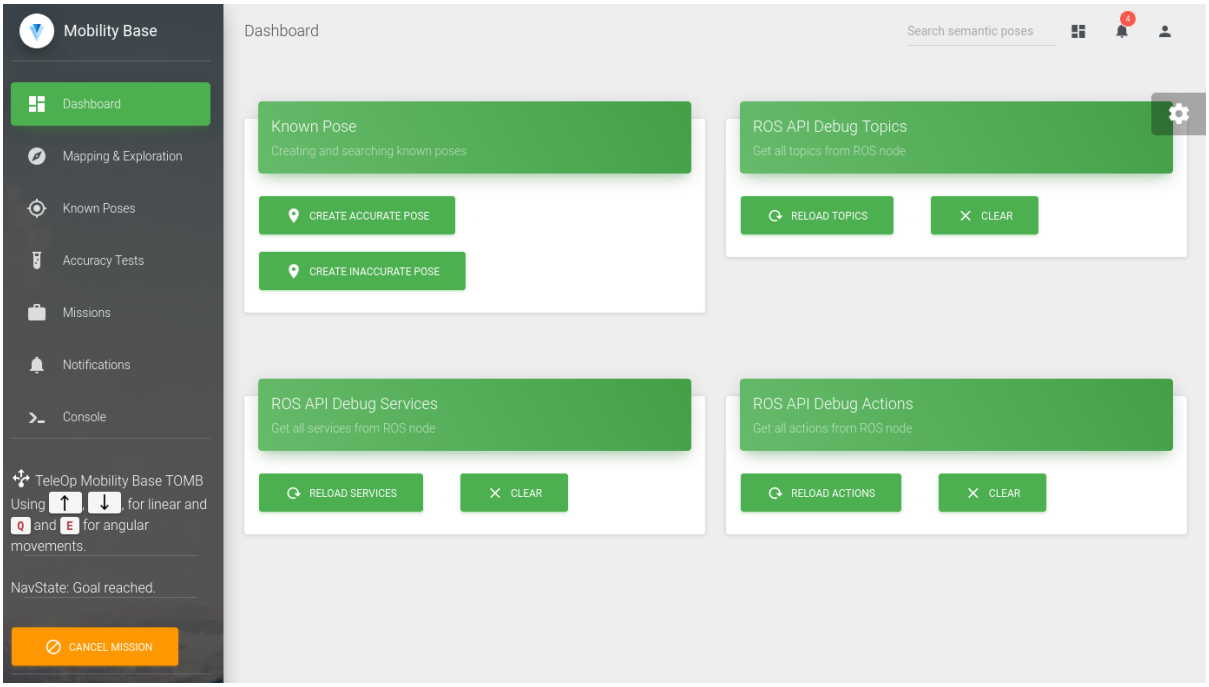


Fig. 3.17.: GUI Dashboard

GUI for Mapping, Exploration and Navigation In Figure 3.18, the view for all mapping and navigation related tasks is rendered. This includes a 3D environment with a 2D occupancy grid representation of the generated map. Furthermore, this view shows the current robot pose with a 30 Hz refresh rate using an arrow for its position and rotation, which leads to smooth movements of the robot. Finally, the controls in this view provides the ability to start an exploration of previously unseen areas of the map and the (de)serialization of created maps for continuous mapping.

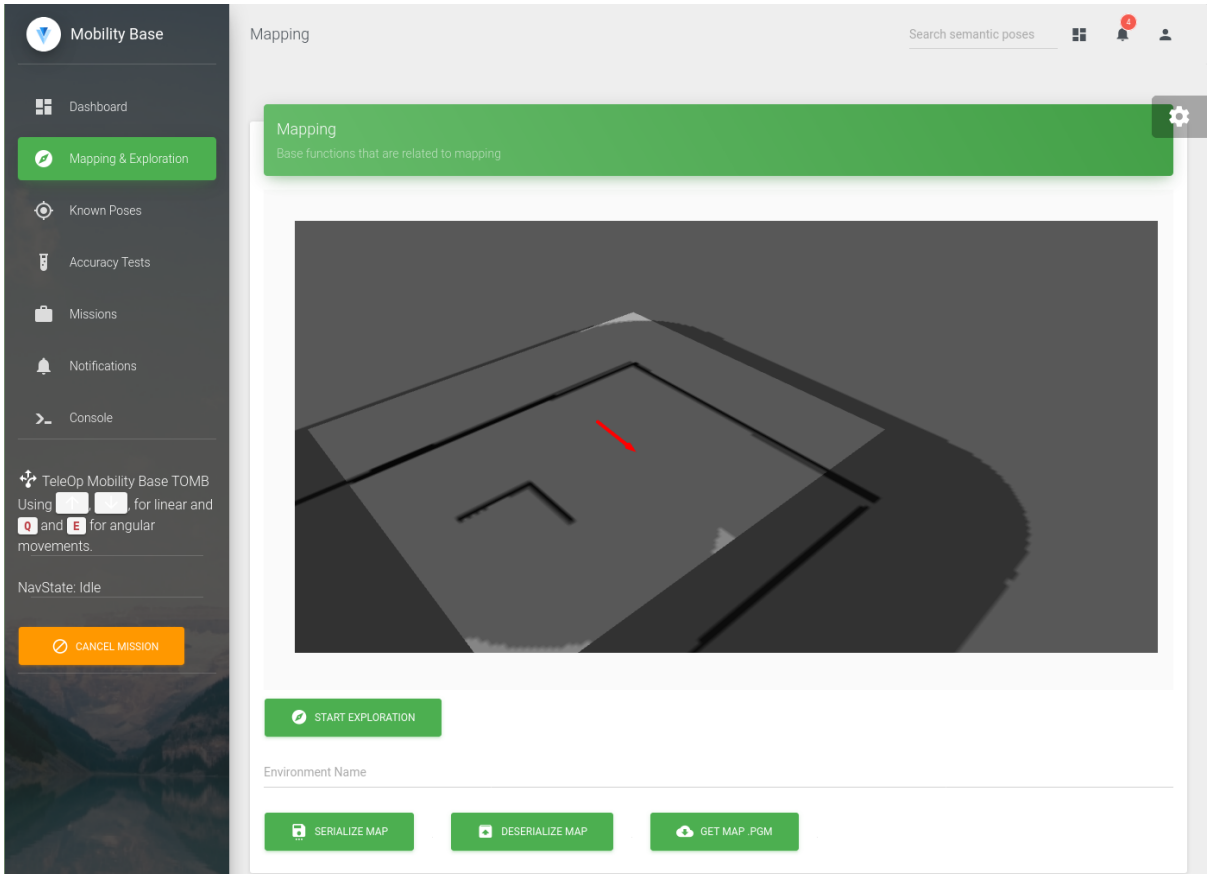


Fig. 3.18.: GUI for mapping, exploration and navigation

GUI for Creating Known Pose The GUI in Figure 3.19 allows the addition of semantic poses using a simple interface. A descriptive name can be entered in the text box and clicking the button to create the pose at the current location will show 2 options. The user then can choose the accuracy of the final pose, which will save the pose in the Known Pose DB. The newly created pose will be shown in the Table below together with other information gathered from the ROS environment. These created poses can be set as targets that will be added to the goal of the navigation stack. Using the bin, a pose can be deleted from the database.

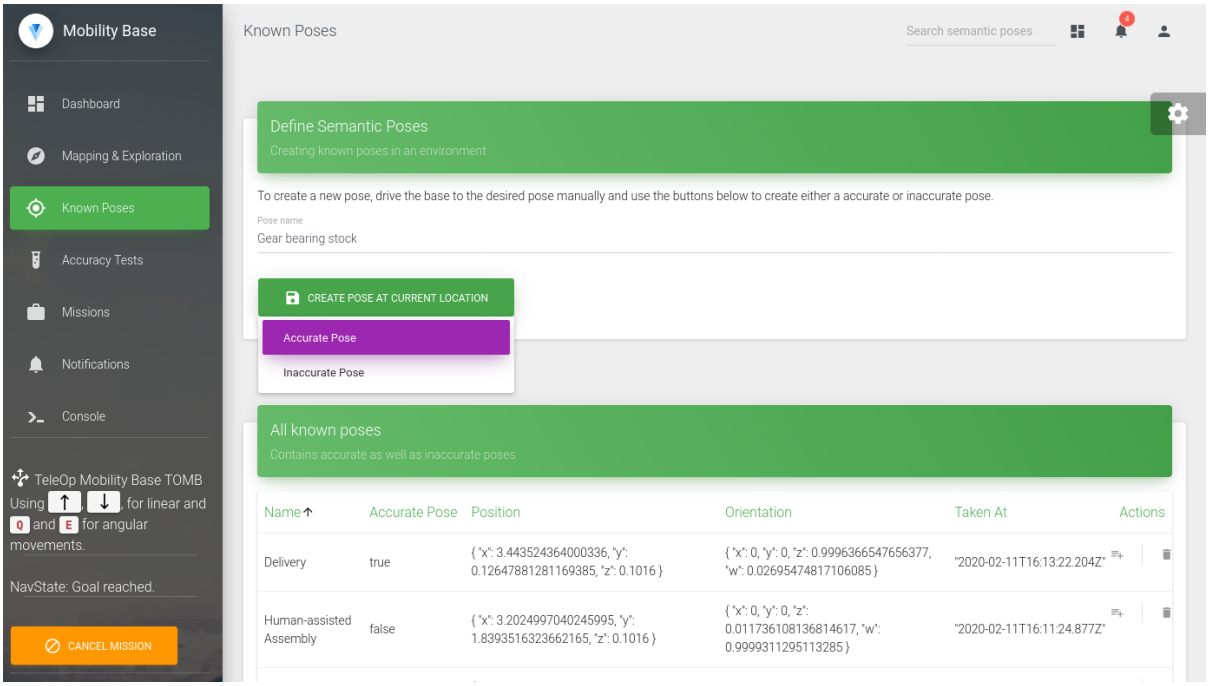


Fig. 3.19.: GUI for creating a known pose

Known Poses API

A `Known Poses API`[5] was developed to define and retrieve (accurate) semantic poses in a 3D environment. Because of some features that are specified in the OpenAPI Specification 3.0 but not yet implemented, some design compromises were made. One features that are not yet implemented include a `anyOf` response type, which can then be used to return different subschemas.[6] Another feature that was missing is the `discriminator` keyword, which facilitates polymorphism by allowing consumers to detect the object type.[7]

The data schema in 3.20 shows how known poses are stored in the database. The workaround is that the sub-components BasicPose and InaccuratePose are reference objects inside their respective parent model.

HTTP Method	Resource	Description
GET	/poses	Searches poses with optional search parameters
POST	/poses	Adds an inaccurate pose item
DELETE	/poses/{id}	Deletes a pose from the system using the `uuid`
GET	/poses/accurate	Searches accurate poses with optional search parameters
POST	/poses/accurate	Adds an accurate pose item

Tab. 3.14.: Operations of the Known Poses API available to developers

[5]Documentation available publicly here: https://app.swaggerhub.com/apis/kw90/known-pose-api/1.1.0
[6]More information at: https://swagger.io/docs/specification/data-models/oneof-anyof-allof-not
[7]More information at https://swagger.io/docs/specification/data-models/inheritance-and-polymorphism

Table 3.14 shows the 5 operations that are available to developers in the interface. Separate endpoints for the different data types were created as a workaround to the missing response type feature. Using the `HTTP GET` method on `/poses` to retrieve inaccurate poses will actually return all poses, including the accurate ones. The accurate poses amongst will have the `InaccuratePose` type with the `boolean` attribute `isAccurate` set to `True`.

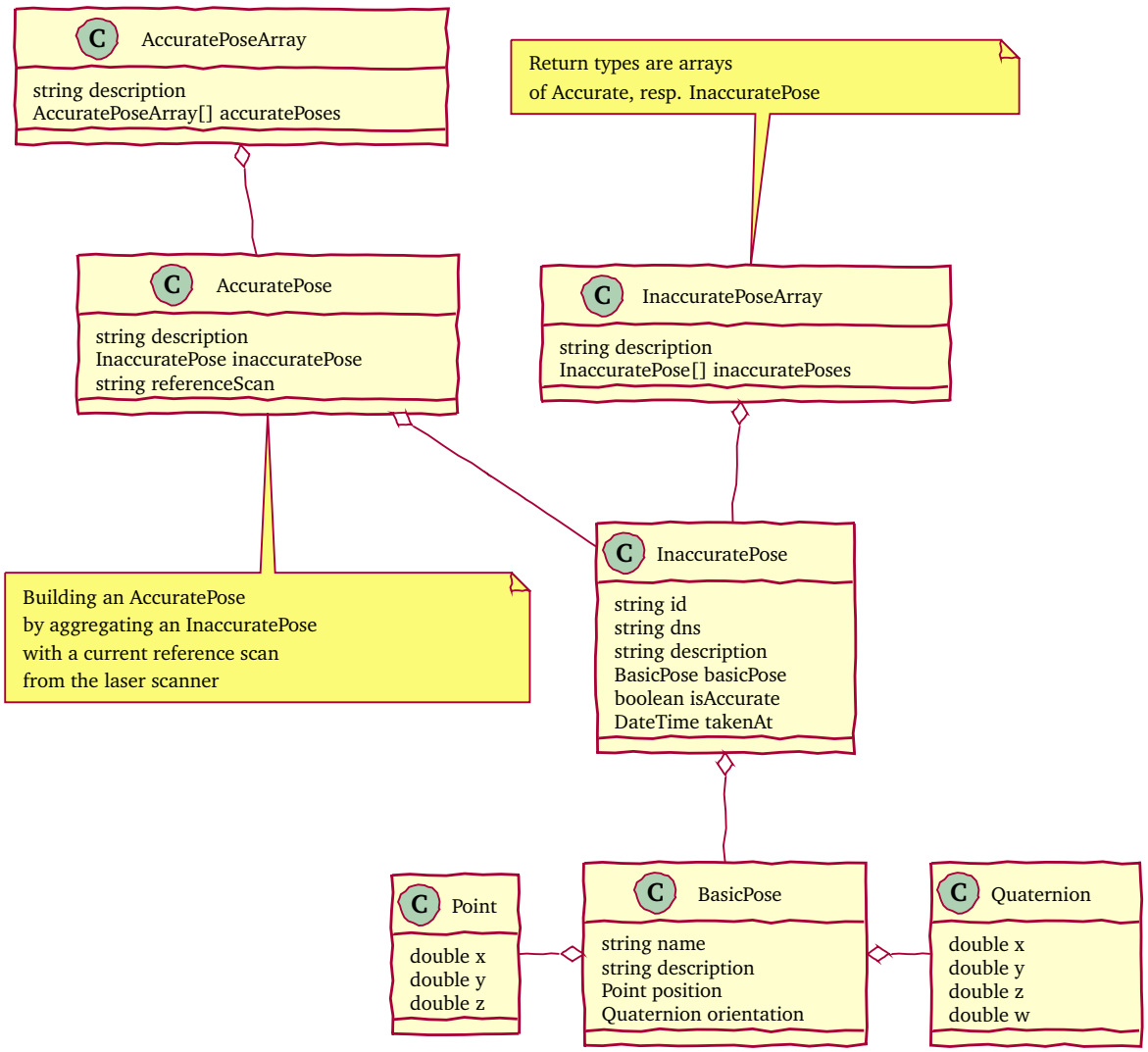


Fig. 3.20.: Data schema for known poses stored in the Known Poses DB

Rapid Prototyping with Jupyter

Dealing with ROS tools normally requires the widget toolkit Qt to get a GUI and either C++ or Python. The tedious process of building, running and shipping ROS application for non-experts in ROS and the formidable learning curve as described in Chapter 2.1 requires a simpler ecosystem. A simpler ecosystem using Jupyter Notebooks

is proposed by [65, 19] to bring the user closer to the code.

Using the template of [65], a virtualized version using Docker was created in this thesis, allowing the service to run anywhere. With this approach, fast initial testing of our approach presented in the next chapter was possible due to the interactive nature of these notebooks. As the code can be written in different cells that can be run independently, it is easy to execute, modify seperate lines and execute again to see the effects instantly. Also, the notebooks can embed data and markdown[8] text besides the code. Providing this component to other developers makes their life easier, as they can perform experiments rapidly. There's no need to install anything but a browser locally as everything runs on the server. The component can also be deployed on a local development machine and can be connected to the robot over the network if need be. Only the Docker engine is required in that case. This approach can also be used together with the simulator as can be seen in Figure 3.21. In that Figure it is seen, how a navigation goal is given to the base using `rospy`.

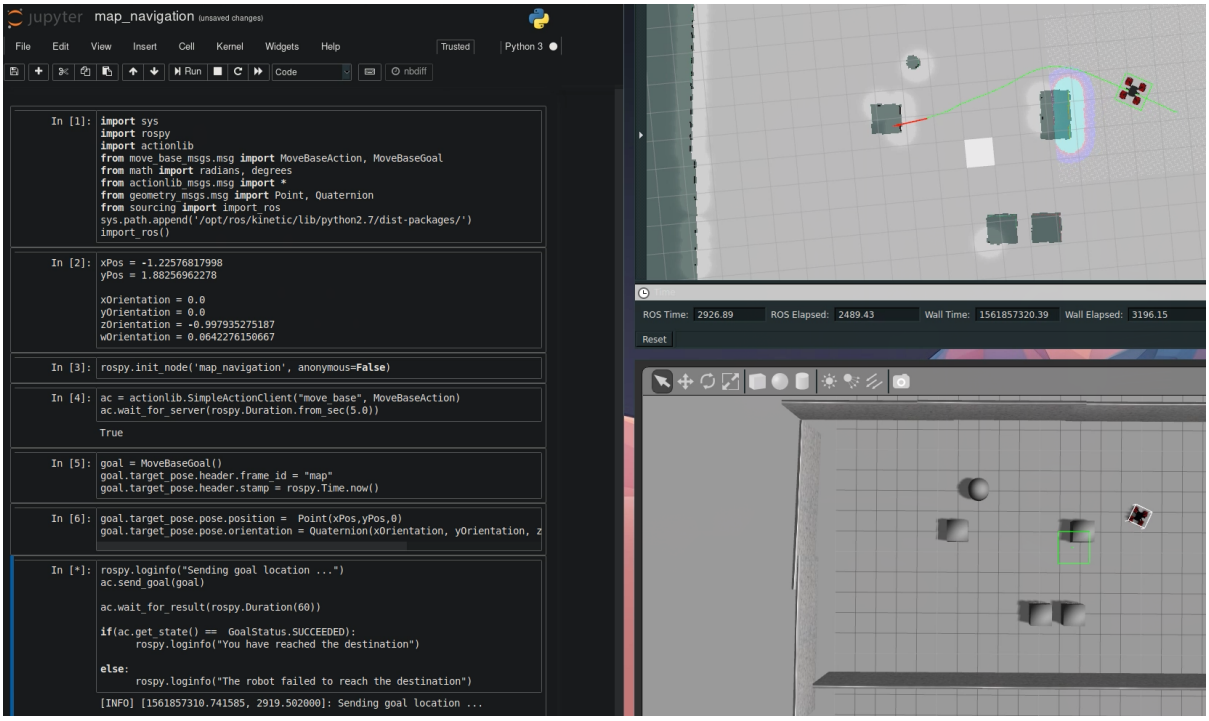


Fig. 3.21.: Jupyter Notebook showing an interactive experience for robotics research

ROS Core System

The ROS Core System implements the Mobile Base Task Control Architecture discussed in the previous chapter. This pipeline architecture processes data as it comes, transforms it and passes it through the components discussed in Table 3.15. In the following subsections, the important and contributed components are discussed in more

[8] A lightweight markup language with easy syntax

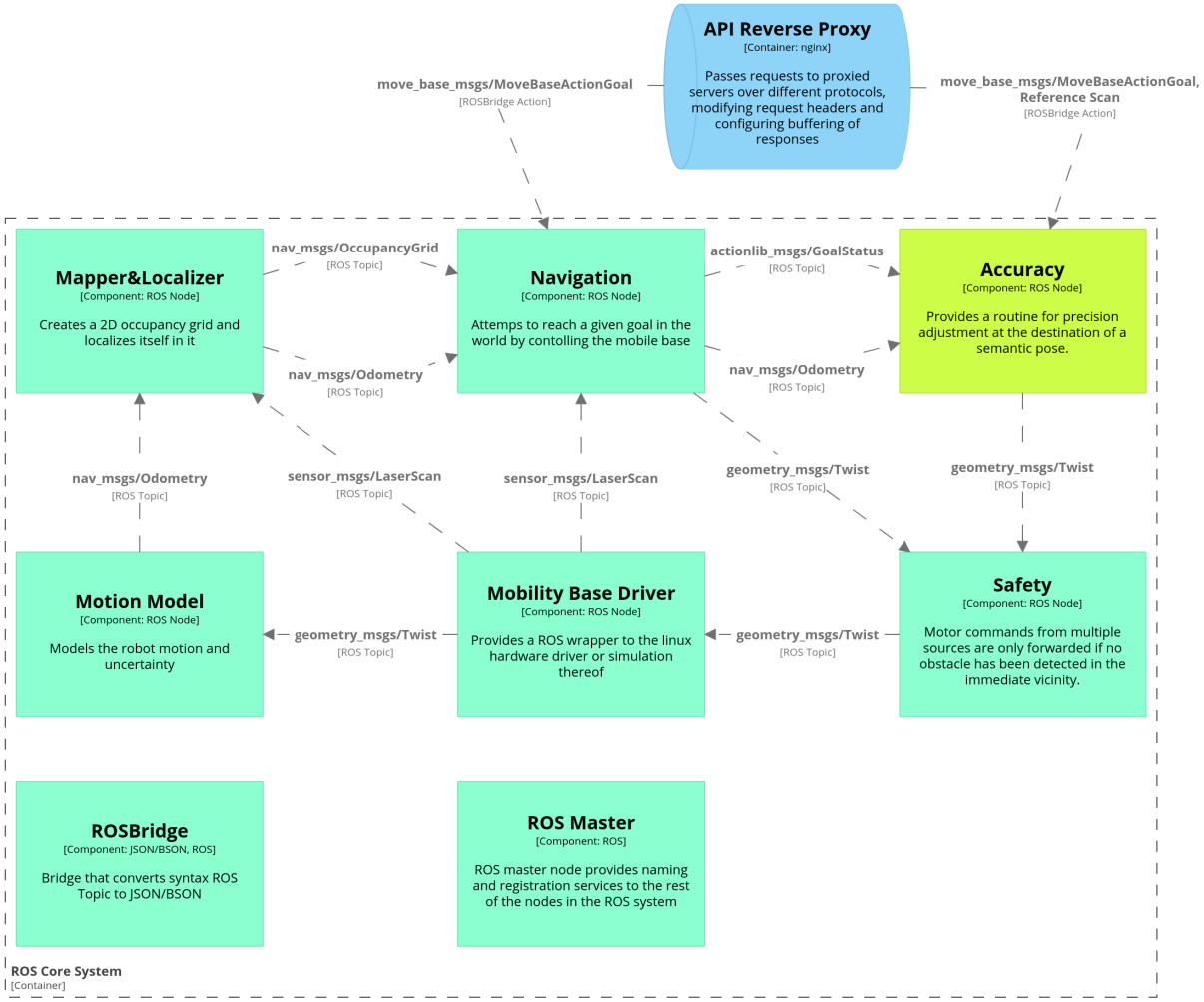


Fig. 3.22.: Component diagram zooms into ROS Core subsystem showing the building components

detail. Isolating the ROS Core System into its own module and network allows the rest of the system to be independent from ROS, use different programming langues and tools, and prevent changes to propagate out. A WebSocket API called `ROS Bridge` bridges communications between networks. It allows clients to subscribe to topics using simple JSON or BSON objects. Even though the connections from the `API Reverse Proxy` are drawn directly into the `Navigation` and `Accuracy` components, they are first routed through the `ROS Bridge`.

`Component`	Purpose	Technologies
`ROS Master`	Mandatory ROS node that provides naming and registration services to the rest of the nodes in the ROS system. This component is maintained by the ROS community and requires no configuration.	ROS Kinetic Docker 19.03.5
`Mobility Base Driver`	Provides a ROS wrapper to the Linux hardware driver or simulation thereof. This component is maintained by Dataspeed Inc and respective device manufacturer.	ROS Kinetic C++11 Docker 19.03.5

Component	Purpose	Technologies
Motion Model	Models the robot's motion and uncertainty by computing the odometry given the velocities of the robot stemming from the geometry_msgs/Twist topic. Additionally, this node provides a continuous transform at a frequency of 30 Hz to the /robot_pose topic for a fluid motion of the model in AR/VR and the web interface.	ROS Kinetic C++11 Docker 19.03.5
Mapper&Localizer	Component runs the slam_toolbox that provides lifelong mapping and localization. Using the laser and motion data, it creates a 2D occupancy grid and creates the ICP-based pose-graph for improving localization and multi-session mapping.	ROS Melodic Docker 19.03.5
Navigation	This component plans and executes paths using the Mobility Base and a map as OccupancyGrid and attempts to reach a given goal with a tolerance of approximately 5 cm.	ROS Kinetic Docker 19.03.5
Accuracy	If a pose is declared accurate, this component performs fine adjustments, once the Navigation component triggers the event Goal reached. For this, it uses the precise odometry pose and goal pose and tries to match them up.	ROS Kinetic C++11 Docker 19.03.5
Safety	This safety component takes motor control commands from multiple sources and checks them for obstacles within 1.5 m. If no obstacle is detected the controls are forwarded to the driver to be executed.	ROS Kinetic C++11 Docker 19.03.5
ROS Bridge	Bridge that converts syntax ROS Topic into JSON/BSON over a WebSocket API.	ROS Kinetic Docker 19.03.5

Tab. 3.15.: Components of the ROS Core System

Mobility Base Driver

This containerized package contributed in our work contains all necessary drivers and dependencies. It is however required to install the SDK from Dataspeed Inc. on the platform beforehand. The container also needs to be run in privileged mode, in order to have access to the devices exposed under /dev. A script that takes care of the SDK installation for a fresh install is provided. During initial testing with the laser scanner, a problem was identified, where some range readings are considered twice. The fix described in chapter 4.1 was reported on the official GitHub project using a Pull Request PR.

An integrated simulation environment can be used in order to test coding increments faster without being bound to hardware. This simulation should reflect the real world

as closely as possible such that the coding increments can be deployed with only minimal changes to the real robot. Testing social navigation behavior in the official Gazebo simulator is therefore required. Scripting actors such as humans to move around and interact in the simulation environment requires at least Gazebo version 8.x[9]. Dataspeed provides a plugin of the Mobility Base for Gazebo that supports versions up to 7.x. A major API change was introduced in version 8.x that switched the internal math library to *Ignition Math*[10]. This API change required to migrate the Dataspeed plugin to the new software version using the migration guide[11] from Gazebo. The migrated plugin we developed is available publicly on GitHub[12].

Mapper & Localizer

Component runs the `slam_toolbox` that provides lifelong mapping and localization. Using the laser and motion data it creates a 2D occupancy grid and the ICP-based posegraph for improving localization and allowing multi-session mapping.

The Kinetic branch of that package does not compile due to a dependency on a newer version of a visualization plugin for the *ROS Visualization RViz*. Therefore, the Melodic branch has to be used. The message definitions did not change between both ROS versions, therefore this should be fine. The containerization of that component is allowed to use a different ROS version isolated for that component without affecting other components. The second contribution adds configuration files to start a new mapping or to continue from a previous mapping session. To allow continuing a previous mapping session, a persistent storage in form of a file system has to be mapped inside the container. A tutorial for setting this node up can be found in the Appendix A.6.1

Navigation

This component plans and executes paths using the Mobility Base and an OccupancyGrid as map. Currently, the containerization developed in this thesis offers configurations and tunings for 3 global and 3 local planners. During startup time, the planners can be defined and the parameters can be tuned during runtime. Provided are the following global planners: `GlobalPlanner`, `Navfn` and `CarrotPlanner` as well as the following local planners: `Timed Elastic Band TEB`, `Dynamic Window Approach DWA` and `Trajectory Planner TP`.

They all provide strength and weaknesses in different situations. The evaluation of the planners is beyond the scope of this thesis. Nevertheless, a stable navigation was needed to test it's accuracy reliably and further improve upon it. Some tuning of the configurations was done following the guide from [54] and using the navigation config from [66] as a template. The final configurations used for this thesis can be found in the Appendix A.6.1. The modifications were done purely to make the robot differential

[9] More information at http://gazebosim.org/blog/gazebo8
[10] More information at https://ignitionrobotics.org/api/math/6.4/index.html
[11] Available at https://bitbucket.org/osrf/gazebo/src/default/Migration.md
[12] Available at https://github.com/kw90/mobility_base_simulator

drive and reduce the velocities and accelerations in all linear and angular movements, and goal tolerances.

The costmaps configurations in the Appendix A.10 until A.12 specify the parameters used in the navigation stack. The main changes to the configuration files happened to the inflation layer and the global costmap. The inflation layer is inflating obstacles, in order to propagate cost values out from occupied cells that decrease with distance. For the costmap configuration a rolling window approach has to be defined using a window that is at least the size in meters of the environment the robot has the potential to map. This is a workaround in order to allow exploration of unknown territory at the same time as doing SLAM and navigation.

To add social navigation, the social navigation plugin layers from [67] were added to the costmaps using the `social_navigation_layers`, namely `ProxemicLayer` and `PassingLayer`. First, the proxemic layer adds a Gaussian Distribution around detected people in the direction of their velocity. An example of this can be seen in Figure 3.23. The cost is increased in proportion to the detected people's velocities. Second, the passing layer adds cost to the right of a detected person, such that the robot always prefers to pass a human on the left side. While such modifications can lead to suboptimal situations as discussed in section 2.3, it is better than having no modifications. In order to make the movements of the base more predictable and more natural, the local path planner was adjusted, such that movements in forward direction are preferred by decreasing the maximum backward velocity.

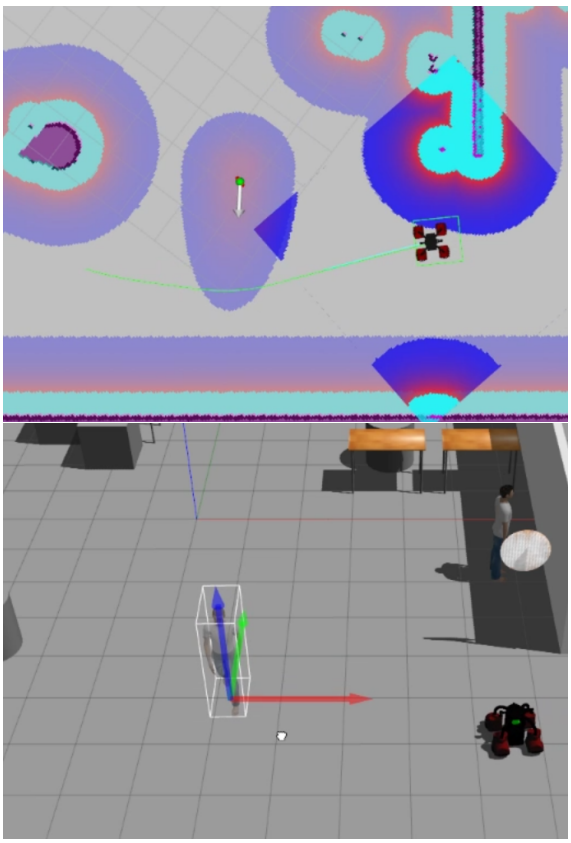


Fig. 3.23.: Social navigation plugin layer added to costmaps seen in RViz above and the scenario in Gazebo in the image below

Good performance with respect to robust navigation was achieved using the GlobalPlanner in combination with either DWA or TEB using differential contraints. The TEB planner had real troubles reaching destinations in tight spaces. This could also be due to some costmap or planner misconfiguration. Testing and tuning of the local planners has to be done to find the optimal values, as it was not in the scope of this thesis. In the end, the combination GP and DWA planner was set.

Accuracy

The accuracy node's responsibility is to drive to target poses as accurately as possible af-

ter the navigation stack has reached the goal with a tolerance of $5\,\mathrm{cm}$ and $0.1\,\mathrm{rad}$. Only a short overview is given here, as the contributed odometry-based algorithm that runs in this node is explained in the next chapter in section 4.4. The node interacts with 3 other nodes as depicted in Figure 3.25.

Interactions The node acts on one side indirectly for the ROS Bridge as an endpoint that provides point clouds. On the other side it receives a trigger event from the navigation that it has reached its destination and that the fine adjustment can begin. The accuracy node then sends motion controls over the safety node to reach the goal precisely.

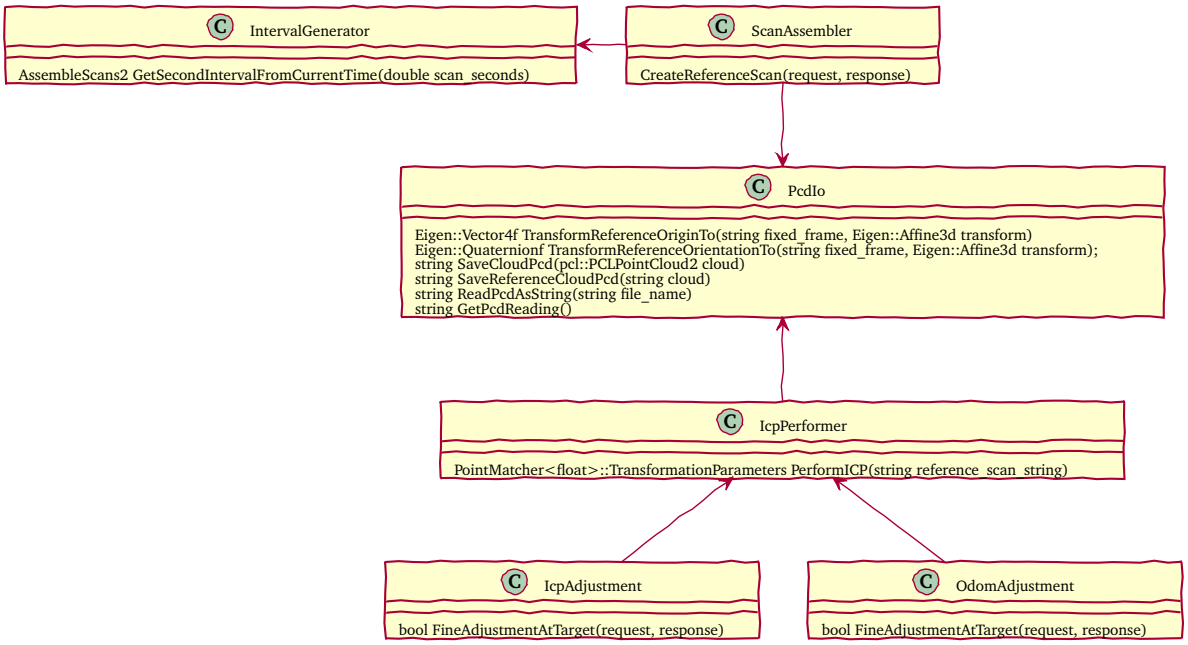


Fig. 3.24.: Class diagram of the accuracy node

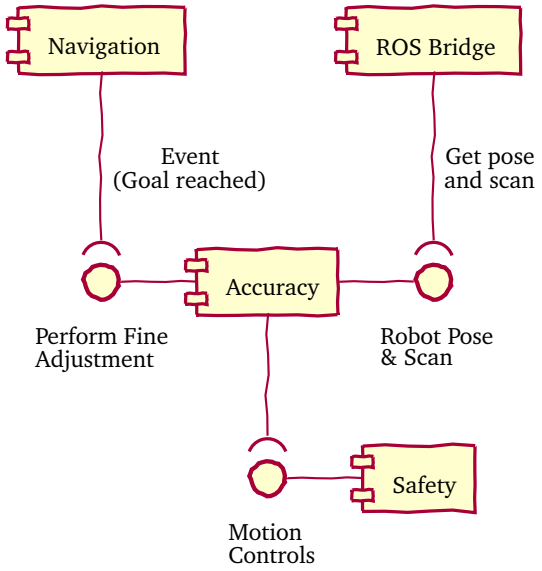


Fig. 3.25.: Accuracy node interactions

As a new known pose gets created, it requires a reference scan as PCD file. The reference scan is used for estimates of how good the robot can estimate the positioning error itself. At the same time, the known pose requires the current pose that is with reference to the map coordinates. This current pose with map reference is the only input the odometry-based method needs. For an ICP-based method, which was not developed in this thesis, the reference scan is additionally needed. When the user wants to navigate to an accurate pose previously defined and the navigation stack reaches it, an event will occur in that triggers the fine ad-

justment. In order to do this, the node listens to the `current_goal` topic of the navigation stack, which returns the 6D goal pose. The reference scan for the ICP-based method has to come over the ROS Bridge from the Known Pose API. This reference scan is then matched with a current reading from the laser scanner after the fine adjustment finished. This matching step yields an error estimate that is returned over the rosbridge to the user.

Safety

The safety node is an example of a reactive component that checks if an obstacle is in the proximity and adapts its speed to that distance. This check ensures that every node that produces motion controls such as `Navigation` and `Accuracy` do not collide with any abstacles. Also, it adapts to speed based on the distance of the nearest obstacle. The measure of slowing down when close to obstacles is computed by the *logistic curve* $f(x) = \frac{1.5}{1+2 \cdot \exp(-nx)} - 0.5$ where x is the nearest distance, n is a steepness factor of the curve and y is the speed in $\mathrm{m\,s^{-1}}$.

A logistic curve with $n = 2$ is plotted in Figure 3.26. The plot shows distance ranging from $0\,\mathrm{m}$ to $2\,\mathrm{m}$ on the x-axis and $\mathrm{m\,s^{-1}}$ in the y-axis. This function has the property to slow down conservatively in the beginning and reducing faster when getting closer. When the MB is driving with a speed of $1\,\mathrm{m\,s^{-1}}$ and an obstacle is approaching below $2\,\mathrm{m}$, this function needs to be evaluated with a high frequency ($120\,\mathrm{Hz}$) in order to avoid accidents. Tests using a cardboard box have shown that collisions can be avoided when producing motion cotrolls that drive at $1\,\mathrm{m\,s^{-1}}$ straight in direction of the box.

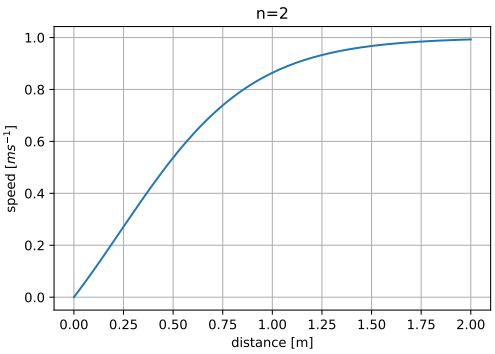


Fig. 3.26.: Logistic curve to compute a safe speed

For people standing or walking next to the robot, this slowing down should give some comfort. This is especially the case when the base is driving around people, as it would not drive with $1\,\mathrm{m\,s^{-1}}$, but rather with a speed somewhere between $0.5\,\mathrm{m\,s^{-1}}$ and $0.7\,\mathrm{m\,s^{-1}}$.

3.3.2 Runtime View

To show how the building blocks of the system behave concretely and interact, 2 UML activity diagrams are modeled. The first diagram in Figure 3.27 shows the interaction of all components discussed when an administrator adds a new semantic pose using the web dashboard. In the next diagram in Figure 3.28, the interactions between components are shown, which need to cooperate in order to reach a given pose by the worker.

Creating a New Known Pose

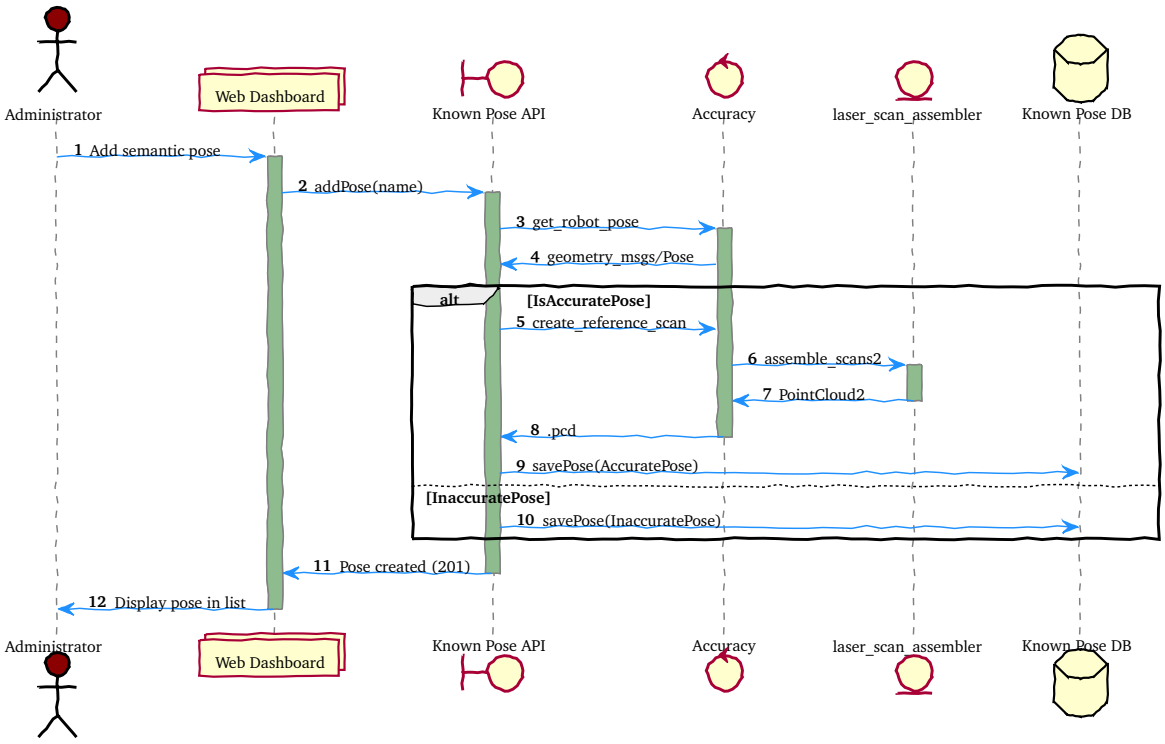


Fig. 3.27.: UML Activity diagram for adding an (in)accurate pose

Sending a Known Pose as Action Goal

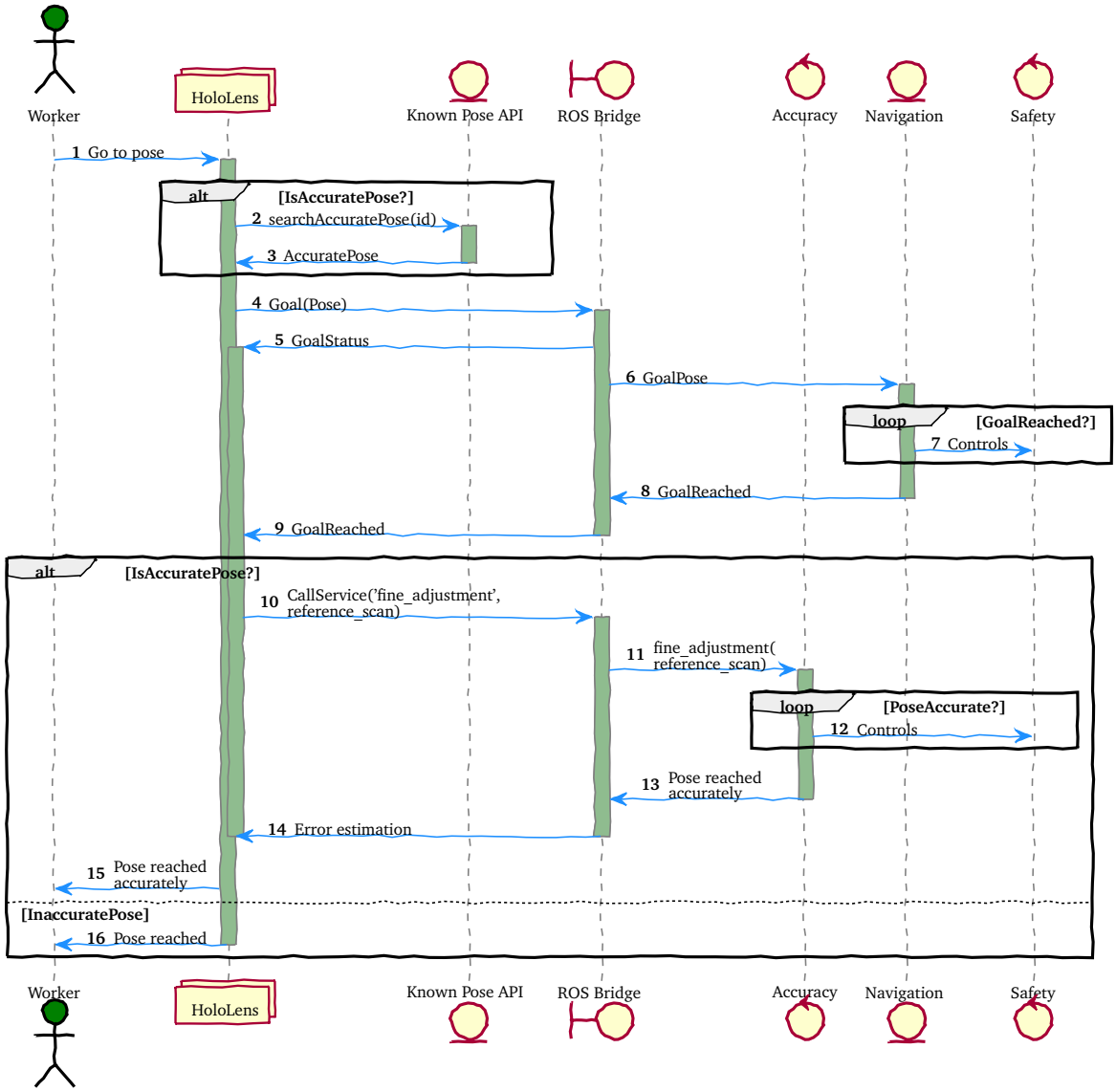


Fig. 3.28.: UML Activity diagram for reaching a pose (in)accurately

3.3.3 Deployment View

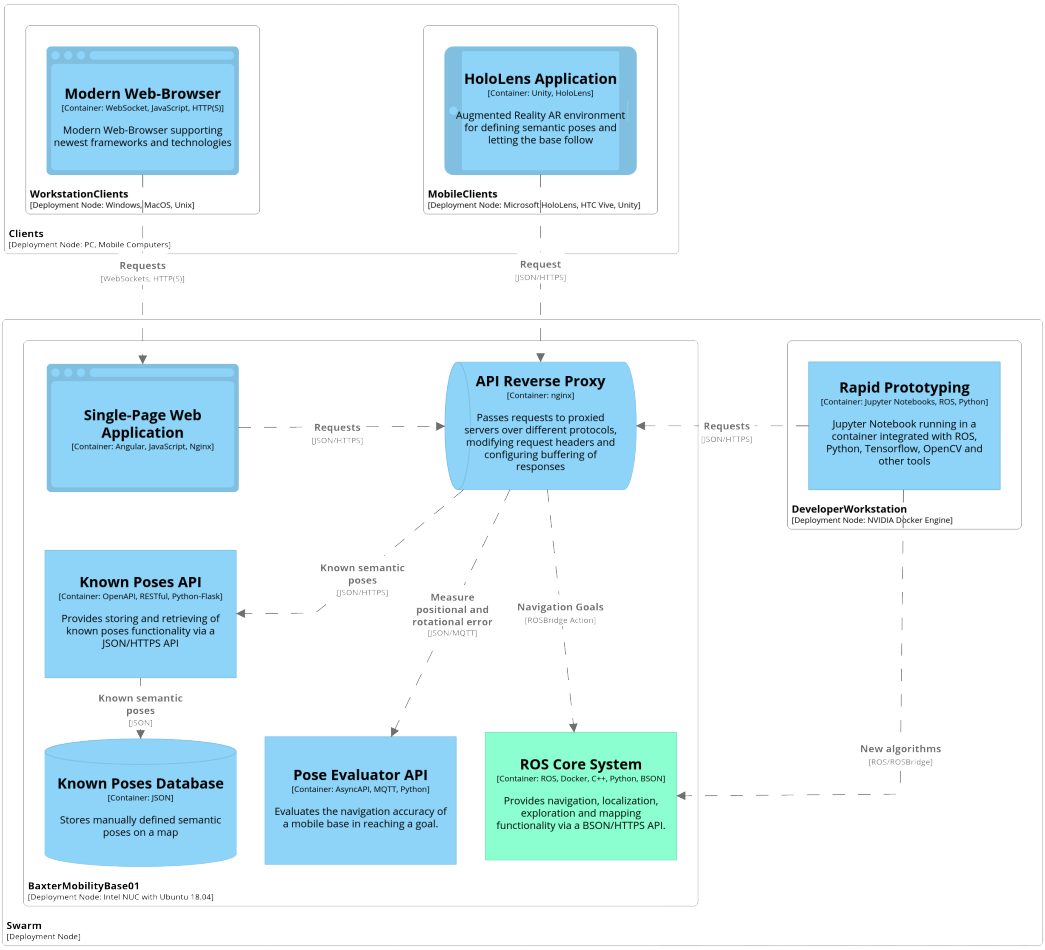


Fig. 3.29.: Deployment diagram illustrating how containers in the static model are mapped to infrastructure

Extensibility Examples

The extensability and flexibility of the Architecture is briefly discussed using 4 examples that have occurred in the past or could happen in the near future.

1. Add a 3D RGBD camera for SLAM
    - only affects the ROS Core System
    - adds 2 nodes (driver and a 3D point cloud to 2D laserscan converter)
2. SLAM container on a central server for enabling multi-mapping using multiple robots
    - only affects the ROS Core System
    - with Docker SDN the server can be added to the virtual network
    - a new SLAM node runs on the server building a shared map
    - the current SLAM node has to be replaced

3. Platforms without possibility to deploy containers
    - cross-compile all ROS files inside a container using the CPU architecture of the platform
    - the compiled files can be run onboard the robot without tethered computer or superuser privileges
4. Platforms without access to OS
    - run containers on a remote machine
    - configure navigation to the robots kinematic constraints
    - connect to robot's ROS master over a network connection
    - listen to sensors and send control commands over the network

A deployment as described in Item 4 can be seen in Figure 3.30. The MiR platform does not provide access to the OS but exposes the ROS Master over port 11311 on the robot's IP. This means we can deploy all containers on a remote controlling machine minus the MB driver node, which is provided by the robot itself.

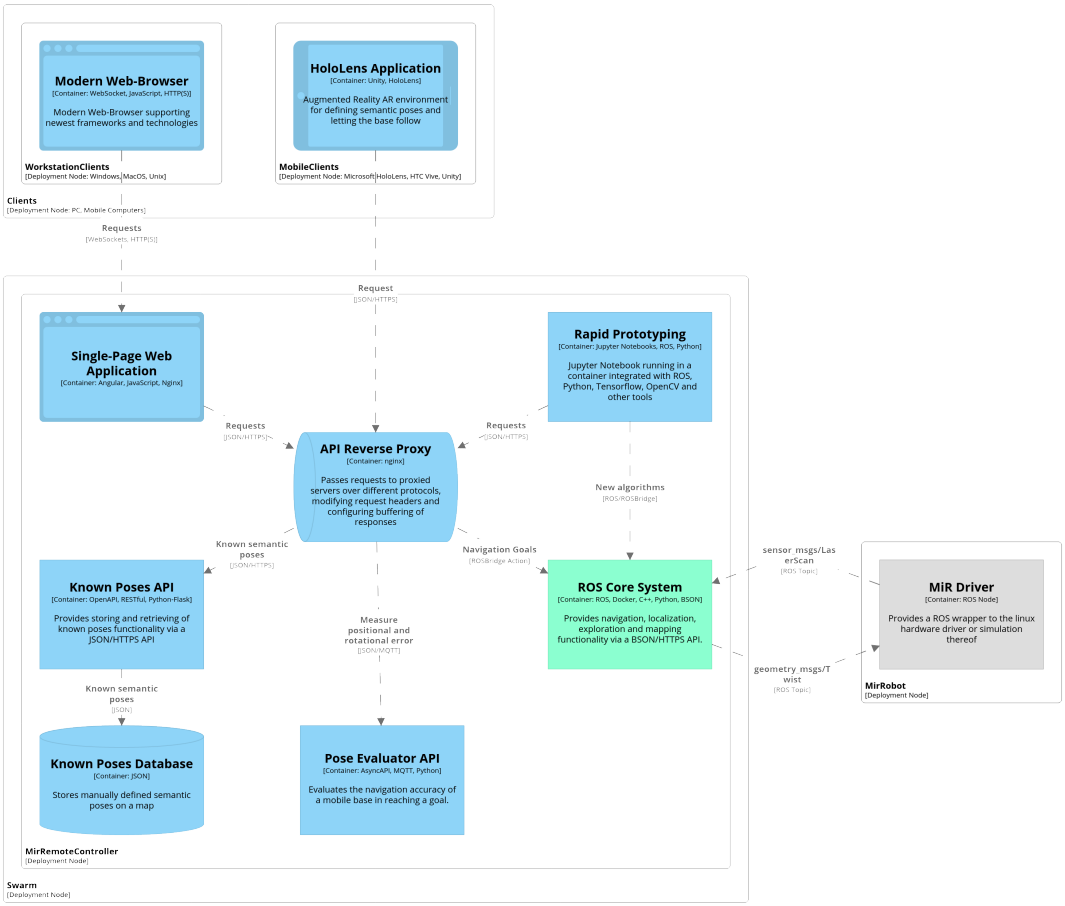


Fig. 3.30.: Deployment diagram illustrating how containers in the static model are mapped to another infrastructure that does not provide access to its OS

3.3.4 Technologies

Various technology decisions had to be considered, of which some were constrained by the environment. For example, using ROS for controlling the robot at high frequencies is constrained to using C++11 or Python3.6. All decisions concerning technology are documented below.

Container Runtime Environments

Development and operations are not isolated concerns, as software should work in development as well as in operations. Initially, this comes at some cost of setting up all required nuts and bolts such as the build and delivery pipeline of the infrastructure, which is discussed in the next Section. Providing the same runtime environment in development on a local machine using the simulator as in production on the robot, makes this process more efficient. Different alternatives that offer container runtime environments nowadays mainly include

- **Docker**
- Linux Containers LXC
- CoreOS rkt (Rocket)
- Mesos Containerizer

Docker still seems to be the quasi-standard and many tools exist as a result of that. Because the simulation environment and visualization components require a GPU accelerated runtime environment, the *NVIDIA Container Toolkit*[13] is needed. This toolkit requires Docker containers to automatically configure the leverage of a *NVIDIA GPU*. While alternatives for AMD exists as *Radeon Open Compute Platform ROCm*[14] for Docker, most computers in the MRK 4.0 lab at DFKI are equipped with *NVIDIA* graphics cards. Therefore, this technology combination of `Docker` with `nvidia-docker` was chosen.

Programming Languages

A multitude of programming languages exist with different strength and weaknesses. Choosing the right programming for a problem is an important aspect in solving the problem efficiently. The ROS Kinetic environment in the ROS Core Subsystem sets a constraint that limits these choices to either C++11 or Python3.6. For components outside the ROS ecosystem, developers are free to choose their language of choice. The developers can possess know-how in the following programming languages.

- Java
- Python
- C++
- Go
- JavaScript

To achieve a certain modifiability accross developers, it should be restricted to the languages mentioned above if possible.

[13] More information at https://github.com/NVIDIA/nvidia-docker
[14] More information at https://github.com/RadeonOpenCompute/ROCm-docker

Web Servers

For Python the following web servers were investigated

- **Flask**
- CherryPy
- Tornado
- Django

As Django offers a complete solution from web server to database – it is not light-weight enough. The rest are light-weight web-server only technologies with different request handling mechanisms. Tornado works with a multi-threading request handler, which is not needed for this project. Flask and CherryPy are somewhat similar in the sense, that they serve requests using a single thread. Flask has more community traction and many resources are availble for it. A web server with Flask is easy to set up and needs only few lines of code to get it working. Therefore, Flask is chosen as the web server technology for python services.

Frameworks for Web Applications

The frameworks considered for implementing web applications were the following

- Angular
- **Vue**
- React

Angular is a complete solution that has a steep learning curve and uses TypeScript as its programming language. Vue is a light-weight framework that is also easy to learn. Vue claims to have more performance which is important for a good user experience and it is driven entirely by the open-source community without contributions from big companies such as Google and Facebook. For these reasons, the Vue framework was chosen for the web application part.

3.3.5 Infrastructure

For this project, various resources from the Enterprise Lab of the Hochschule Luzern HSLU is used. All links and information about code, artifacts, docker images, API documentation, API clients and container runtime can be found in the Appendix A.7.

DevOps

The use of Docker Swarm makes it possible to implement a complete DevOps[15] toolchain. Such a chain makes it possible, that a developer only has to push his code to the GitLab repository. The rest is taken care of by the Continuous Integration / Continuous Deployment CI/CD toolchain that automates the process of software delivery and infrastructure changes. These set of tools aid in delivery, development and in the end management of application throughout the life cycle of software development.

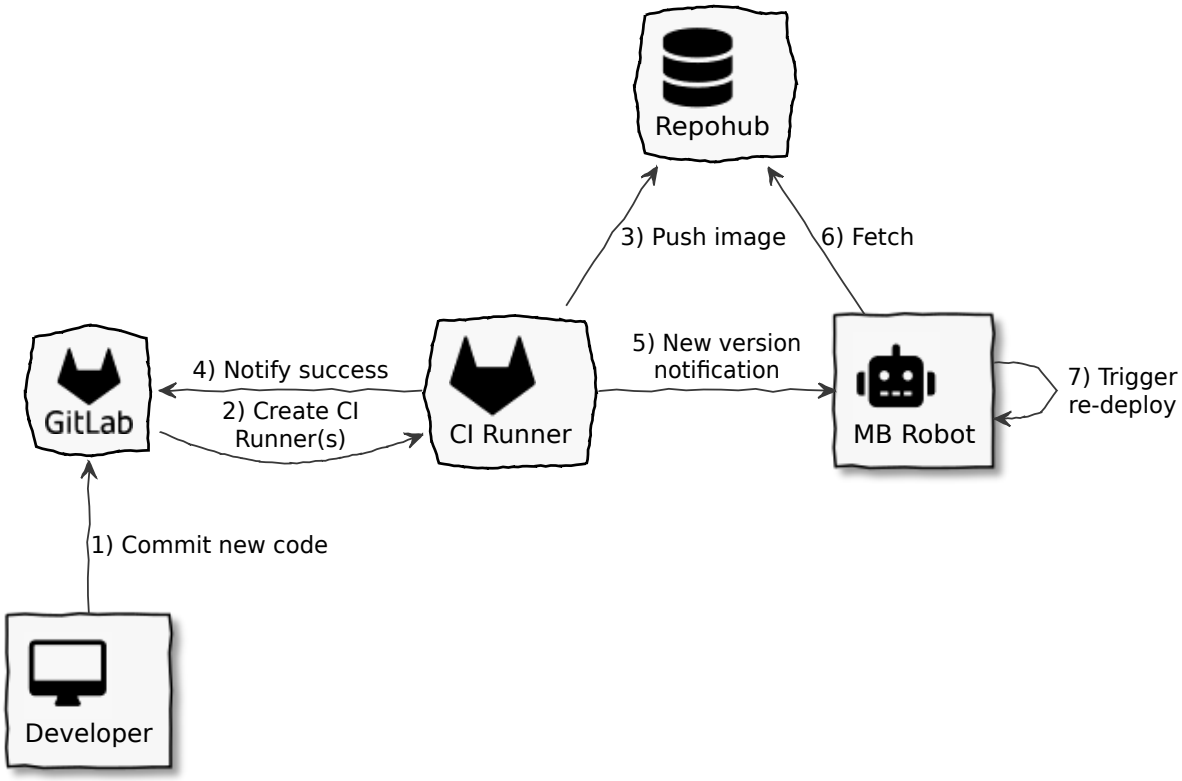


Fig. 3.31.: Continuous Integration / Continuous Deployment Overview

[15] Practices that automates the processes between software development and IT teams; stands for **Dev**elopment and **Op**eration**s**

Updating the API Spec

Creating a new version of an API requires re-building all client and server packages to reflect the new interface. As the official image from OpenAPI did not allow easy calling from an CI environment such as GitLab CI Runners, the image was modified to allow that using the official Java binary and mounting volumnes inside the CI runner. In order for the Runner to be allowed to push to a registry, a key has to be generated and stored in the registry. Updating the Spec and with it the Client SDKs as well as Server Stubs works as follows

1. Edit the yaml at SwaggerHub for convenience and validation and save the changes
2. Commit and push the changes to the repository
3. Magic
4. Find the client SDKs and server stubs over at the registries

The magic part will build all new clients defined in the CI chain using the new API documentation. Once done, the artifacts are pushed to the corresponding registries such as PyPI or NPM.

4 Precise Positioning of Mobile Robots at Semantic Poses based on Pose Graph Localization

> *Programs must be written for people to read, and only incidentally for machines to execute*
>
> — **Abelson and Sussman**
> (Structure and Interpretation of Computer Programs)

In this chapter the findings from the previous chapters are summarized and improvement opportunities are presented. A contributed bugfix that improves the accurate conversion of scan points from the laser scanner into 2D coordinates for the laser scanner is discussed. Finally, the approach proposed in this thesis for improving the positioning using an pose graph-based approach that can deal with changes in the environment when converging accurately to a goal pose is described.

4.1 Fencepost error correction

The 2D laser scanner in use is the `R2000 ODMM30M` from Pepperl+Fuchs [68]. This scanner is designed to periodically measure distances within a full 360° field of view FoV while rotating with a constant frequency defined by `scan_frequency`. Measurements are aggregated into scans where a single scan corresponds to one revolution of the sensor head around the FoV. One such scan yields a sequence of scan points (or samples) that can also be defined using the `samples_per_scan` parameter. Scan data acquisition is performed sequentially in the direction of head rotation around the origin of the scan plane.

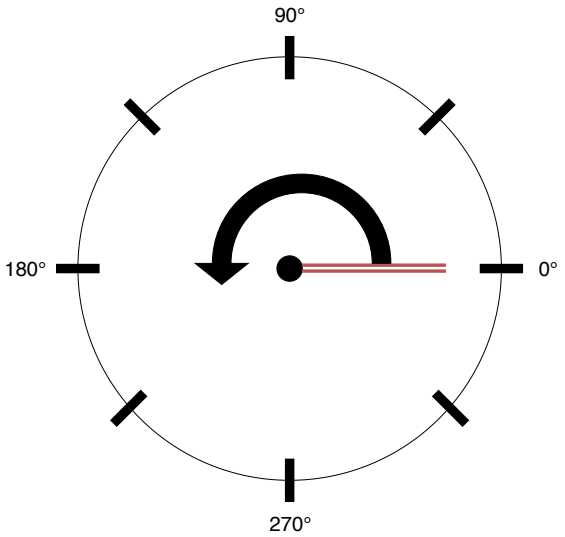


Fig. 4.1.: Polar coordinate system with direction of scan head rotation

As shown in Figure 4.1, this data is typically represented within a polar coordinate system. In this coordinate system it shows the sensor from the top-down view. The origin is located at the point of intersection of the axis of rotation and the axis of the laser beam. The pole of this coordinate system where the sensor sits is defined by the axis of rotation. Scan points are continuously recorded during operation using an uniform `angle increment` and direction of rotation. By default the scan head rotates in mathematically positive direction, which is called counterclockwise (abbreviated with ccw). The angle increment has a positive value between two subsequent scan points because of the scan direction.

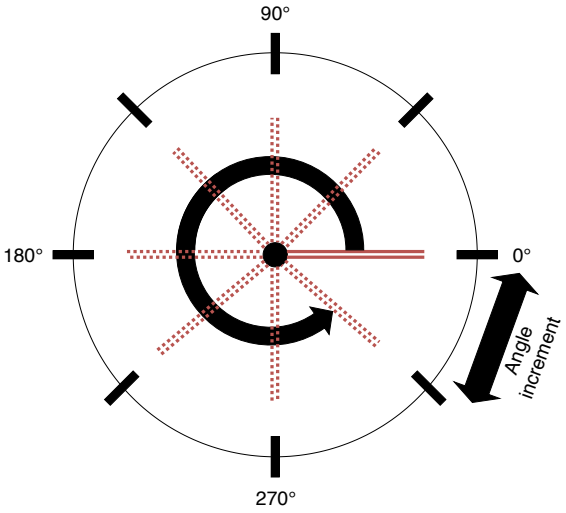


Fig. 4.2.: LiDAR scanning from 0 ($0°$) to 2π ($360°$) with an angle increment of $\frac{\pi}{4}$ ($45°$)

The package `pepperl_fuchs`[1] supplies a driver and ROS wrapper for this scanner. The current `r2000_node.cpp` creates scans within the range of $(0, 2\pi)$ with a dynamic `angle_increment` that is defined in `samples_per_scan` variable. Figure 4.2 shows an example for a simplified laser scan with an angle increment of $45°$.

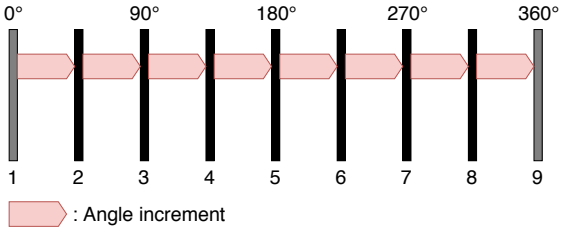


Fig. 4.3.: LiDAR scanning from 0 to 2π

Using this full range leads to a *fencepost error*, where the scan in $0°$ direction is registered two times. The error becomes more evident looking at the fence sections and posts in Figure 4.3 where 9 scans (posts) are separated by 8 angle increments (fence sections). The scans at $0°$ and $360°$ actually represent the same scan. Some ROS libraries (e.g. `navigation_2d`) complain about that error and others ignore it. Ignoring it leads to an incorrect conversion of the scan points into 2D coordinates. This conversion, however, is important for doing accurate SLAM.

$$angle_{min} = 0 \tag{4.1}$$

$$angle_{max} = 2\pi - angle_{increment} \tag{4.2}$$

Therefore, we redefined the maximal angle, keeping the minimal angle as is. Equations 4.1 and 4.2 show the new range for my adapted ROS driver node. In the example the range would be from $0°$ to $315°$.The pull request PR[2] was submitted to the official repository and can be tracked there. To fix the issue for the time being my fork[3] has to be used, because the pull request has not been merged into the master branch yet.

[1] Found at https://github.com/dillenberger/pepperl_fuchs
[2] Pull request at https://github.com/dillenberger/pepperl_fuchs/pull/11
[3] Fork at https://github.com/kw90/pepperl_fuchs

4.2 Mapping and Localization

Mapping sensor measurements into internal representations of the environment is the task of robotic perception. As defined in the previous chapter decribing the robotic agent and environment in 3.2.1, this perception is difficult because of noisy sensors, and the partial observability, unpredictability and dynamics of the environment. This section summarizes the navigation aspects and its accuracy incorporated so far in the proposed architecture. With these insights, an opportunity to leverage the benefits of lifelong mapping and accurate localization using the pose-graph SLAM is also discussed.

4.2.1 SLAM and Map Resolution

Mapping The most promising mapping results were achieved with the `slam_toolbox` described in [69]. This solution produced qualitatively good maps in a larger environment such as a $80 \cdot 42 = 3360\,\mathrm{m}$ hall area with demonstrators for research and testing purposes. This package also offers the ability for lifelong mapping, which is the concept of mapping a space completely, and over time to refine and update that map as the robot continues to interact with that space. All environments tend to change more or less over time, such as displacing or rotating furniture, opening and closing doors. Updating the internal map with changes in the environment helps the robot to better localize itself in that map and produce better paths. At this time, the lifelong mapping feature in the slam_toolbox package is still highly experimental and could substantially impact the computational performance.

Map Resolution All SLAM solutions tested provides a map resolution parameter, that sets the size of the occupancy grid block. An occupancy grid map discretizes the world around the robot into individual blocks. Such a block defines a state of the environment at that location in space. The states can either be assumed to be occupied or free. Increasing the map resolution means reducing the size of the blocks.

Unfortunately, reducing the size of the blocks has a drastic impact on the amount of memory consumed that is required to track the occupancy grid. At the same time, this increases the computation, which is required to keep those grid cells updated. Together with the people tracking, this leads to a lot of computation that has to be done for each individual cell. The increment in memory is easy to calculate, as halving the grid granularity from $5\,\mathrm{cm}$ to $2.5\,\mathrm{cm}$ squares would quadruple the memory requirement.

A second issue in reducing block sizes stem from sensor precision. Especially low cost LiDAR can return noisy distance readings, which can vary over time. If this noise exceeds the defined map resolution, the occupancy grid starts to get fuzzy. This can lead a solid wall to being no longer a single surface, but several nearby surfaces. This shows, a trade-off and finding the right map granularity for each robot platform could be investigated further.

Localization Odometry-like localization is provided by the same `slam_toolbox` package. Taken from the idea of KartoSLAM, the localization is built upon point cloud *registration* (process of aligning two or more point clouds of the same scene) along the robot trajectory. These point clouds are then used to build a graphical model called the `pose graph` on top that uses the `Iterative Closest Points ICP` algorithm. Results from ICP provide transformations between registrations used to connect reference frames are defined along the robots' trajectory. Using this information to build a pose graph enables the correction of the path of the robot as well as the map of the environment, for the occurrence of a loop closure. These loop closures at graph level also helps in reducing the overall drift of the system.

As opposed to (A)MCL with KDL sampling used in [3], the localization approach using the pose graph is not bound to a static map image as a `.pgm` file. Thus, it can incorporate changes in the environment and use them for localization. This should result in better localization accuracy than (A)MCL if the ICP algorithm is well tuned for the task. For the Navigation Stack, a map as occupancy grid is required for both path planning and tracking. As this map defines a resolution of $5\,\mathrm{cm}$ as described, the Navigation Stack is only able to ideally achieve a final pose accuracy inside that defined block size. The localization using a rolling buffer of recent scans in the pose graph can be more accurate than that and be used for a fine adjustment at target poses using the algotithm developed in this thesis (described in section 4.4).

4.3 Robust Navigation Strategy

4.3.1 Goal Tolerances

Local planners, which adhere to the BaseLocalPlanner[4] interface, provides a controller that drives a mobile base in the plane. The wiki page in [70] describes this controller as a service to connect to the path planner. A kinematic trajectory using a map is created for the robot, to get from a start to a goal location.

For the local planner to determine if a goal has been reached with success, two tolerance parameters can be set. A parameter `xy_goal_tolerance` sets the positional tolerance and another `yaw_goal_tolerance` sets the angular tolerance for goal precision. Lower tolerances give bad results in terms of planning success rates. This can make to the robot continually rotating about its goal. Setting the `xy_goal_tolerance` too small, the robot may try endlessly to make small adjustments around the goal position. Similarly, setting `yaw_goal_tolerance` too small, may cause the robot to oscillate near the goal.

```
1   # Goal Tolerance Parameters
```

[4]Found at http://wiki.ros.org/base_local_planner

```
2   xy_goal_tolerance: 0.05
3   yaw_goal_tolerance: 0.1
4   latch_xy_goal_tolerance: false
```

Listing 4.1: Local planner parameters for goal tolerances

Generally, a lower value for both means a closer and harder to achieve range and makes it difficult for the navigation stack to conclude that it has reached the goal. The parameters shown in listing 4.1 are probably not optimal. They seemed to work in the lab environment as well as in the simulator, however, no rigorous testing has been done to evaluate those parameters empirically. One general guideline for the positional tolerance is the used map resolution. That resolution sets the minimum possible tolerance. The map resolution was fixed at $0.05\,\mathrm{m}$, which was used as positional goal tolerance. As for the angular tolerance, there is no such constraint and has to be found through experimentation.

4.4 High Accuracy Pose Convergence

A method for converging accurately to a goal pose has been developed. The method uses a motion model approach using a odometry-like localization from different sources. On the integrated platform, the Mobility Base this odometry stems from the laser scanner and calculates distances from velocity commands. These sources could be combined with an IMU to get more accurate estimations. The IMU on the MB however can not be used due to technical reasons explained in chapter 3.2. The pose-graph approach from the `slam_toolbox` provides a combined odometry from all sources that is used in the algorithm presented next.

The x and y tolerances that trigger the *goal reached* event for a goal pose has been configured to $5\,\mathrm{cm}$. Therefore, the goal can either be in front, to the back or sideways to the current odometry pose as seen in Figure 4.4. This thesis proposes the computation described in Algorithm 1 that solves the problem of aligning a given goal pose with the current odometry pose frame that corresponds to the robot pose. This computation gets triggered once the navigation stack reports that it has arrived within the tolerances of the goal pose. Figures 4.5 to 4.12 show step-by-step the computations and executions of the Algorithm 1 in a visual fashion. Because movements in y direction (left, right) are either impossible (with non-holonomic wheel drives) or more inaccurate (with holonomic robots) than movements in x direction (forward, backward), the algorithm only uses movements in x direction and turns around the z axis. The following computations assume a right-handed coordinate system when viewed from the z axis the system is counter-clockwise CCW.

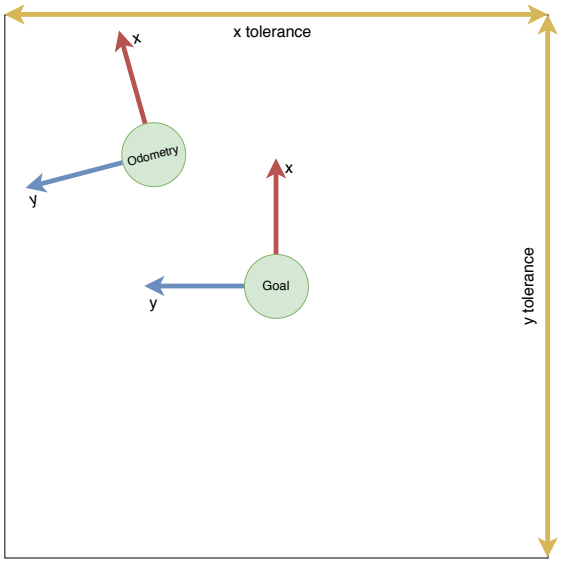


Fig. 4.4.: Navigation reaches the goal with a tolerance

Algorithm 1 Pose Graph Odometry Accurate Goal Pose Convergence

Require: 6D Pose of the goal to be achieved $Pose_{goal}$ and odometry $Pose_{odometry}$
Ensure: Align odometry pose with goal pose in 3D

1: **function** ODOMETRYFINEADJUSTMENT
2: TURNTOWARDSANGLETOGOAL
3: DRIVEALONGROBOTXTOGOAL
4: FINALGOALORIENTATIONALIGNMENTROTATION

After the navigation stack triggers the *Goal reached* event, the angle to the goal θ is determined by first calculating the difference in x and y between both positions and then computing $\tan^{-1}(\frac{y}{x})$. Computing the inverse tangent if x is negative or equal to 0 requires special case consideration. By using the function $atan2(y, x)$ from the `C++11 cmath` standard library, it is possible to abstract away from that. This function returns the four-quadrant inverse tangent of the difference in y and x. Computing θ as the angle from the odometry pose orientation to the goal can be seen in Figure 4.5. The full odometry is implemented by the pose-graph localization approach. Once computed, this angle can help to determine if the odometry pose lies in front, to the side or behind the goal pose. This is helpful in order to minimize the amount of on spot rotation effort of the robot, which normally leads to drift and bad estimates.

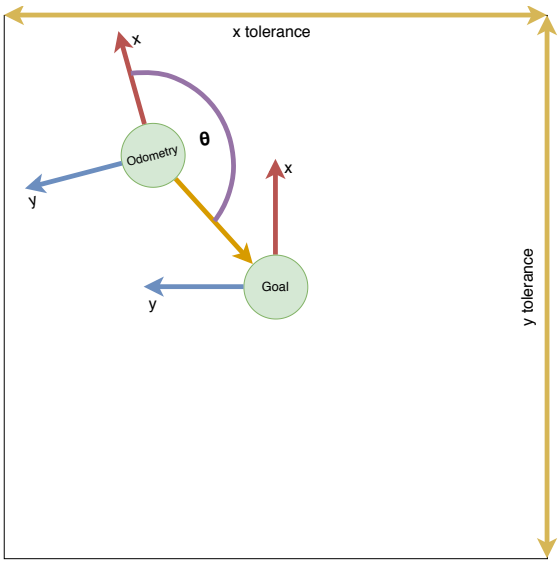


Fig. 4.5.: Compute the angle θ to the goal position

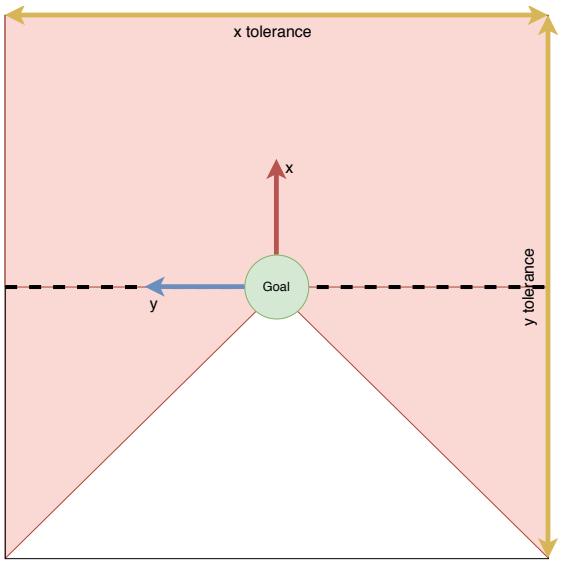


Fig. 4.6.: Sectors that if odom has perfect angle will trigger backward motion

In order to minimize those errors induced by long rotations, it is beneficial to restrict the angle below some threshold. This threshold is set to $\frac{\pi}{4}$ resulting in a defined bad sector as illustrated in bright red in Figure 4.6. Although this sector only holds true if the odometry pose orientation exactly matches the goal pose orientation, it does help in visualizing the problem. With differing angles between goal and odometry pose of $-1\,\mathrm{rad}$ and $1\,\mathrm{rad}$, the borders between the sectors behind the goal pose is able to move.

In case the odometry pose starts in the bad sector, it first backs up until the angle θ is below the defined threshold. Figure 4.7 shows an example where the odometry lies in front of the goal and has to back up into the good sector. The result of that motion together with the new angle θ, which is below $\frac{\pi}{4}$ is illustrated in Figure 4.8. Having a small angle, the rotation towards the goal can now be executed. In case the odometry lies to the left, clockwise rotation needs to be applied and vice-versa. Due to the right-hand coordinate system, this means increasing the z compo-

nent (also called *yaw*) of the angular vector. The computation and matching is defined in Algorithm 2. With frequent updation of the odometry pose and a single goal pose, the function `TurnTowardsAngleToGoal` ensures a matching odometry orientation and angle to goal with a sub milliradian accuracy.

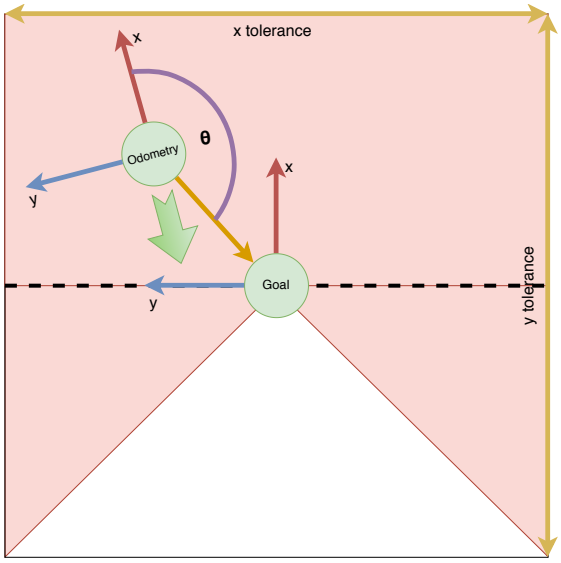


Fig. 4.7.: Navigation overshot target and landed in front and to the left of the goal

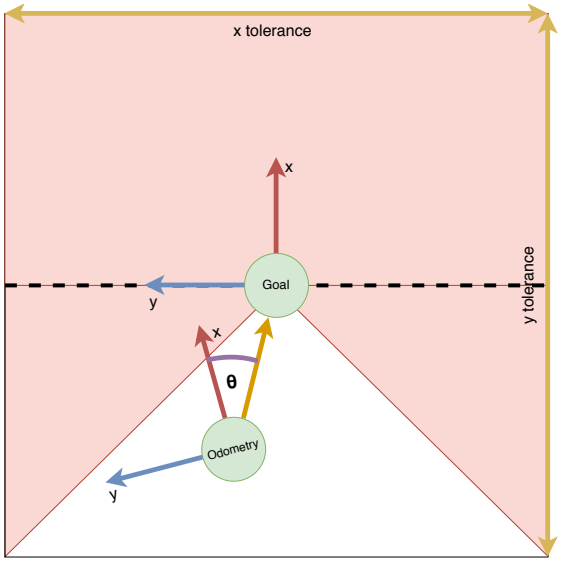


Fig. 4.8.: Difference has to be smaller than $\frac{\pi}{4}$

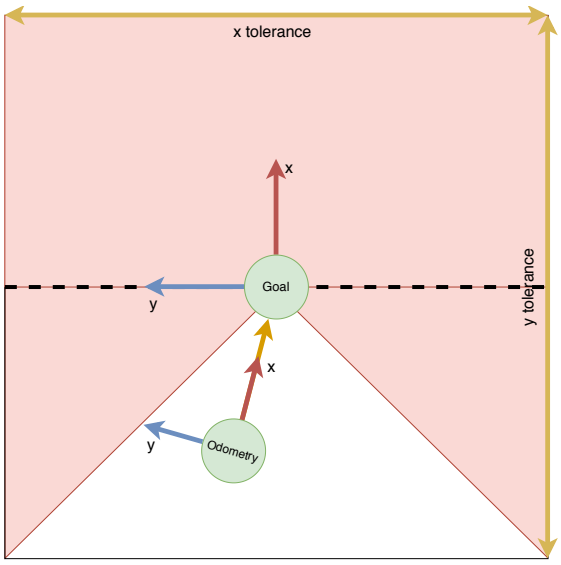


Fig. 4.9.: Turn base's x-axis towards goal pose

Finding the shortest relative rotation from the odometry orientation to the goal orientation it is simpler to convert to quaternions. Using quaternions, it is possible to find the shortest path of rotation by simply inverting the current orientation quaternion and right-multiplying it with the goal position orientation. To do that, we first compute the quaternion to the goal which is defined in Algorithm 3. This function computes the angle to the goal as described earlier using the `atan2` function and converts the resulting angle into the quaternion Q_g. The function named `ComputeQuaternionFromRPY` returns a quaternion using rotation around fixed axes roll, pitch and yaw respectively. To do that, it uses the `setRPY` from the *Bullet 3D physics* library, and further computes the angle using the current pose as well as the goal pose.

Algorithm 2 Align with angle to goal θ by minimizing rotation required

Require: 6D Pose of the goal to be achieved $Pose_{goal}$ and odometry $Pose_{odometry}$
Ensure: Odometry yaw matching the angle to the given goal with a sub milliradian accuracy

1: **function** TURNTOWARDSANGLETOGOAL
2: **while** $\mid \theta - \phi \mid > 0.001$ **do**
3: $Q_g \leftarrow$COMPUTEQUATERNIONTOGOAL($Pose_{goal}$)
4: $Q_o \leftarrow$COMPUTEQUATERNION($Pose_{odom}$)
5: $Q_\delta \leftarrow Q_g \cdot Q_o^{-1}$
6: $\theta \leftarrow$GETANGLE(Q_g)
7: $\phi \leftarrow$GETANGLE(Q_o)
8: $\delta_{rot} \leftarrow$GETYAW(Q_δ)
9: **if** $|\delta_{rot}| < \frac{\pi}{4}$ **then**
10: **if** $\delta_{rot} > 0$ **then** TURNCCW(speed)
11: **else**TURNCW(speed)
12: **else**DRIVEBACKWARDSALONGX(distance)

Computing the odometry quaternion in the `ComputeQuaternion` just requires to convert from the `geometry_msgs/Quaternion.msg` message definiton in ROS to a `tf2::Quaternion` using the provided `tf2::convert` function. With these quaternions, it takes simply the multiplication on line 5 to compute the shortest angle of rotation Q_δ. The functions `GetAngle` are provided by the `tf::Quaternion` and the `tf::Transform` ROS `Transform tf` library that returns the angle of rotation in the interval $[0, 2\pi]$. As seen in the illustrations before, if δ_{rot} is positive and below $\frac{\pi}{4}$, we rotate counter-clockwise and if its negative, clockwise. If however, δ_{rot} is above the threshold, we drive backwards along the x-axis and re-evaluate the situation until the threshold is reached.

Algorithm 3 Compute angle to goal from current pose

Require: 2D Positions of the goal to be achieved and odometry
Ensure: Angle to goal in radians

1: **function** COMPUTEQUATERNIONTOGOAL($Pose_{goal}$, $Pose_{odometry}$)
2: $\Delta_x \leftarrow goal_x - odom_x$
3: $\Delta_y \leftarrow goal_y - odom_y$
4: $\theta \leftarrow arctan2(\Delta_x, \Delta_y)$
5: $Q_g \leftarrow$COMPUTEQUATERNIONFROMRPY$(0, 0, \theta)$
6: **return** Q_g

Algorithm 4 Drive along robots' x-axis to goal position forwards

Require: 6D Pose of the goal to be achieved $Pose_{goal}$ and odometry $Pose_{odometry}$
Ensure: Drives the odometry position to the goal position with a sub millimetre accuracy

1: **function** DRIVEALONGROBOTXTOGOAL
2: **while** $\mid goal_x - odom_x \mid > 0.001$ &&
$\mid goal_y - odom_y \mid > 0.001$ **do** DRIVEALONGX(speed)

The rotations are performed with a constant and slow speed of $50\,\mathrm{mrad\,s^{-1}}$, which amounts to $\frac{3}{2\pi} \approx 0.477$ `rpm`. Two important factors that needs to be considered are – a) δ_{rot} is recalculated in each `while` iteration with updated odometry as drifts in odometry can occur anytime and b) that the executed angular movements exceed the tolerance of $1\,\mathrm{mrad}$ as defined. Both of these issues tend to affect the rotation direction.

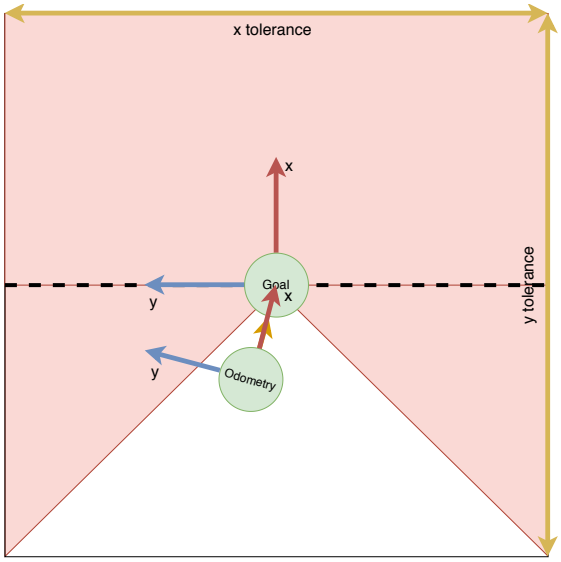


Fig. 4.10.: Drive forward towards goal

After aligning the odometry with the angle to the goal θ, x direction is further moved towards the goal position, with respect to its (x, y) on the map frame. This step can be seen visually in Figure 4.10 and its goal in Figure 4.11. The function is defined in Algorithm 4. In order to make this movement more precise, it is performed with a constant and slow speed. This speed is defined as $0.05\,\mathrm{m\,s^{-1}}$. When the goal pose reaches the position (x, y) with an odometry error of $1\,\mathrm{mm}$ on both axes, the final rotation is invoked. The Algorithm 5 takes care of aligning the odometry orientation with the goal pose orientation. This is similar to the task of Algorithm 2 in the sense that it has to find the angle of shortest rotation and the direction. To do this, it requires the current and goal poses. Using these poses, it creates the quaternions of orientation, inverts the current orientation Q_o and right-multiplies it with the goal quaternion Q_g to get the required minimal rotation needed to align both. Likewise, in a right-handed coordinate system, if δ_{rot} is positive, we have to apply a counter-clockwise rotation and vice-versa. The rotation is applied until the error in rotation exceeds $1\,\mathrm{mrad}$. Same as `TurnTowardsAngleToGoal`, this function also turns the robot with a fixed and slow speed of $\frac{3}{2\pi}$ `rpm` to reduce errors.

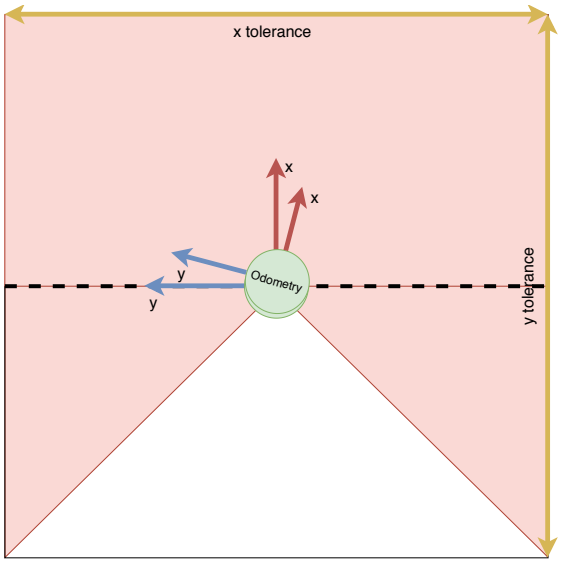


Fig. 4.11.: Reaching goal height with different orientation

Algorithm 5 Final rotation on spot to align odometry to goal pose

Require: 6D Pose of the goal to be achieved $Pose_{goal}$ and odometry $Pose_{odometry}$
Ensure: Rotates the angular odometry to the goal orientation with sub milliradian accuracy

function FINALGOALORIENTATIONALIGNMENTROTATION
  while $\mid \theta - \phi \mid > 0.001$ **do**
    $Q_g \leftarrow$COMPUTEQUATERNION$(Pose_{goal})$
    $Q_o \leftarrow$COMPUTEQUATERNION$(Pose_{odom})$
    $Q_\delta \leftarrow Q_g \cdot Q_o^{-1}$
    $\theta \leftarrow$GETANGLE(Q_g)
    $\phi \leftarrow$GETANGLE(Q_o)
    $\delta_{rot} \leftarrow$GETYAW(Q_δ)
    if $\delta_{rot} > 0$ **then** TURNCCW(speed)
    elseTURNCW(speed)

The final result of all these operations should yield a situation as depicted in Figure 4.12. As drift and estimation errors still can happen, this pose is not guaranteed to be perfect. The positional and rotation error that can be expected using this pose-graph odometry-based method using these algorithms that were explained in the current section are evaluated in the next chapter.

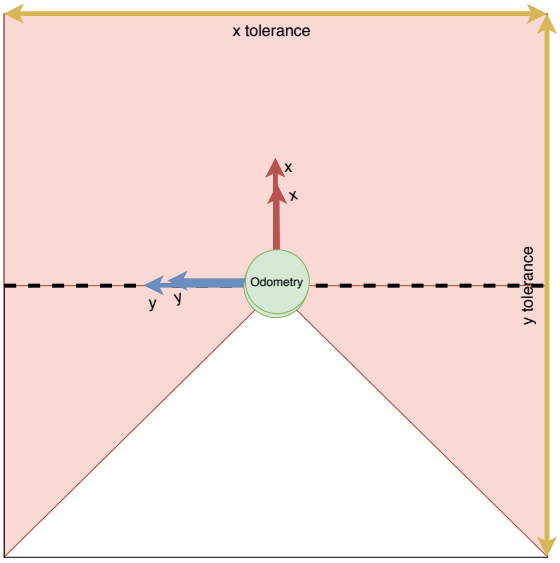


Fig. 4.12.: Final rotation to align with goal pose orientation

5 Experimental Evaluation

This thesis wants to answer the question of how accurate the standard navigation stack `move_base` can navigate to target poses and how it can be improved. The proposed solution should be able to deal with the dynamics in the environment while still being able to estimate its pose with a higher accuracy. An extensible solution architecture has been developed in this thesis that can be deployed to different platforms. Improvements for the accuracy at target poses are suggested by the pose-graph odometry-based algorithm. Also, some open software issues have been fixed and released to the community. While the accuracy in dynamic environments have not been evaluated due to time constraints, a solid groundwork has been laid for more in-depth experimentation as well as extending the solution with further sensors and algorithms. All of the code is open-sourced.

In this chapter, the algorithm developed in the previous chapter using the pose-graph odometry localization is evaluated using an external measurement. First, the methodology for real-world evaluation of the complete system is presented. Using this setup and the proposed architecture, the experiments are repeatable and results of this thesis reproducible. Second, the *OptiTrack Motion Capture System* that was used as external measurement system is described. Lastly, the results achieved with move_base and the proposed algorithm are presented and discussed.

5.1 Evaluation Methodology

Different authors have measured the positioning error of mobile robots in a variety of ways. In [3] they used an external motion capture system with 9 cameras to determine the pose of the robot operating in the environment. This setup was beneficial as it provided a means to evaluate the performance of the navigation system at different locations with precision. Using an external measurement system such as *OptiTrack Motion Capture System*, which is available in the MRK 4.0 Lab allows to determine the pose of the robot accurately up to about $1\,\mathrm{mm}$ if calibrated correctly.

The robot used for the evaluation is the Mobility Base from Dataspeed Inc. is described in 3. The pose-graph for the localization system has been built by steering the robot through the environment using the TOMB tool developed and integrated in the web dashboard. This pose-graph can be compiled into an occupancy grid seen in Figure 5.1 and is used by the move_base. The Figure also shows the 2 evaluation locations marked

with the red squares. These locations were used for the robot to repeatedly approach with both methods and each time the robot stops, the motion capture system would take a measurement. Figure 5.2 shows the lab environment, which is the evaluation site where the tests were run. The Figure below in 5.3 shows 3 out of 9 OptiTrack cameras that were used to track the robot.

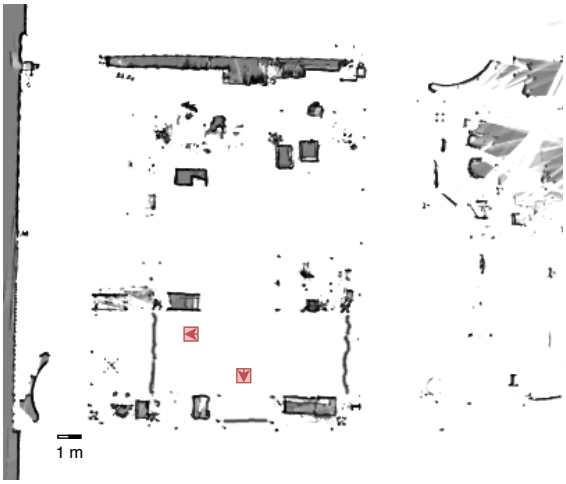


Fig. 5.1.: 2D Occupancy grid of the evaluation environment with a 5 cm resolution

In order for the motion capture system to work accurately, it has to be calibrated beforehand. If the calibration successfully runs and achieves an *excellent* rating from the control software, it should produce a positional error of less than 0.3 mm and rotational error less than 0.05°. The motion capture system provides a 6D pose of the marker configuration that is teached-in. The set of markers and their configuration is tracked by the system. These markers were hot-glued on top of the robot, where they can be easily seen by the OptiTrack cameras. The correct functioning of the system was tested by driving multiple times to the same location and seeing if the measurements match up.

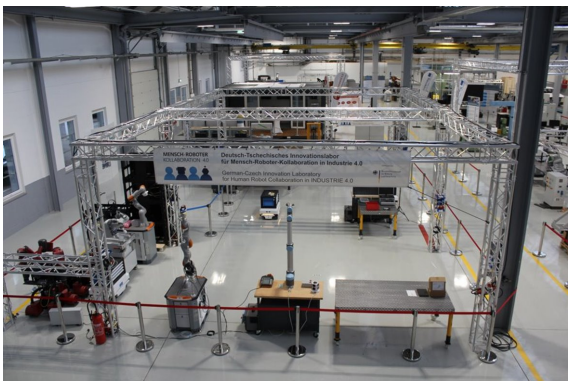


Fig. 5.2.: Lab environment that was used as evaluation site

Fig. 5.3.: Three of the 8 OptiTrack Motion Capture System cameras

As discussed in chapter 3 the navigation system has many individual components that work together to produce and execute a path robustly and to eventually reach the target pose. The dynamics of real world environments are difficult to capture and are largely ignored in this evaluation. To measure the accuracy quantitatively, an evaluation metric that measures the positional and rotational error of the robot relative to manual reference locations is used. For the position metric the translational error in the x, y-plane is computed using Equation 5.1.

$$PE = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \qquad (5.1)$$

The angles measured by the motion capture system are Euler angles with XYZ representation. This representation corresponds, respectively to pitch, roll and yaw. Therefore, only the z rotation component is of interest to us. This component has values in the interval $[-180°, 180°]$. Computing the rotational

error RE in degrees as the angle difference in the general case, the Equation in 5.2 was used. In this Equation, z_1 corresponds to the manual reference rotation and z_2 to the reached rotation. The order does not matter, but it's important to grasp that one variable is the fixed manual reference and the other represents the rotation that is measured in each test run.

$$RE = | z_1 | - | z_2 | \tag{5.2}$$

In this evaluation only the positioning error of the robot is determined. This positioning error means how accurate the robot can position itself. The error of the localization system is not evaluated in this thesis and should be a content for future work. This localization error gives a measure of how well the robot can estimate its own pose, which could indicate limitations of the motion execution in the low-level controller or hardware of the platform. The test procedure for the evaluation is described in Table 5.1. The procedure defines the purpose and the preconditions that have to be met along with the procedure steps to be executed. All steps can be executed in the developed web dashboard, which also includes a live evaluation of the errors at the test runtime. Figures A.9 and A.10 in the Appendix shows the view in action.

We start a test run by driving to a pose using TOMB and creating an accurate pose by giving it a distinguishable semantic name. This pose addition creates a reference scan that gets transformed to a point cloud and is saved to the Known Pose API. At the same time, this triggers an OptiTrack measurement which is saved in the local browser database for later evaluation. The next step involves driving the base to different random starting positions. This navigation to different locations includes random changes in position as well as orientation of the robot. During our tests, this was made using TOMB but could be imagined to be executed by a software component that sends the robot to random but reachable positions in the map. This second step should be repeated a lot of times to get a sample size that yields a significant result. Each test run also includes, besides an OptiTrack measurement, a final transformation estimate of the robot. This estimate can be used to determine the localization error, which is not included in this evaluation. After each run, the evaluation refreshes a Table containing both errors as well as a scatter plot, which shows the distribution of the test measurements including their rotation. After running a multitude of test runs, the test data can be downloaded as JSON to be processed further, if the evaluation in the web dashboard is not enough.

Test Procedure: Positioning Error	
Purpose	This test procedure creates an initial manual measurement at a reference location using OptiTrack. It then evaluates the positional and rotational error with repeating autonomous navigation runs of a method in reaching the exact pose again.

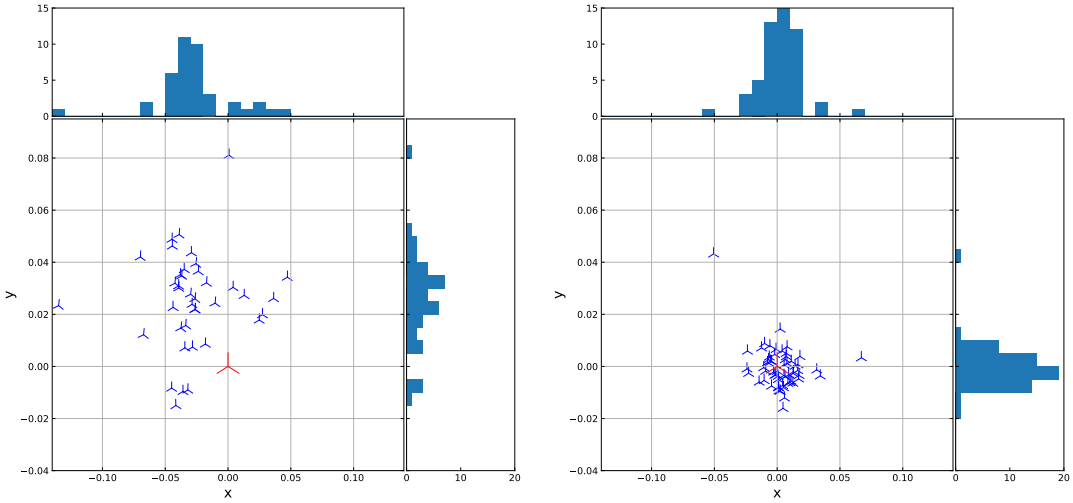
Test Procedure: Positioning Error	
Preconditions	1. Pose-graph of the environment created 2. Complete navigation stack up and running 3. Known Poses API running 4. OptiTrack system calibrated and tracking the robot
Procedure steps	1. Create new pose a) Drive to a desired pose inside of the range of the motion capture system accurately using a Joystick or TOMB b) Create new accurate pose on web dashboard by providing a semantic name • Get reference scan • Save OptiTrack measurement 2. Run test using the web dashboard (Repeat n times) a) Navigate to different starting positions randomly b) Give the robot the task to navigate back exactly to the manually defined goal pose c) Capture measurements once robot stops moving 3. Download data a) Save to a JSON file • Pose Name • ICP Transformation estimate • Reference OptiTrack measurement • Test OptiTrack measurement

Tab. 5.1.: Test protocol for evaluation

5.2 Benchmark Pose Accuracy of Odometry-Based Algorithm

After calibrating the OptiTrack system as described above with an *excellent* rating the experiments were executed. For the tests, the two poses marked in the map in Figure 5.1 were defined manually. Both move_base and odometry-based methods were tested with 142 runs. Thereof, 82 were conducted with the odometry-based method and 60 with the move_base. In this thesis, we focus on increasing accuracy (*trueness and precision*). Improving trueness requires decreasing systematic errors by getting the mean of the measurements closer to the actual reference value, in this case the manual measurement. In order to get better precision, the amount of random errors needs to be decreased to get more closeness between results. Simple statistical analysis was used to compare the results of both move_base and the odometry-based method.

Position Error



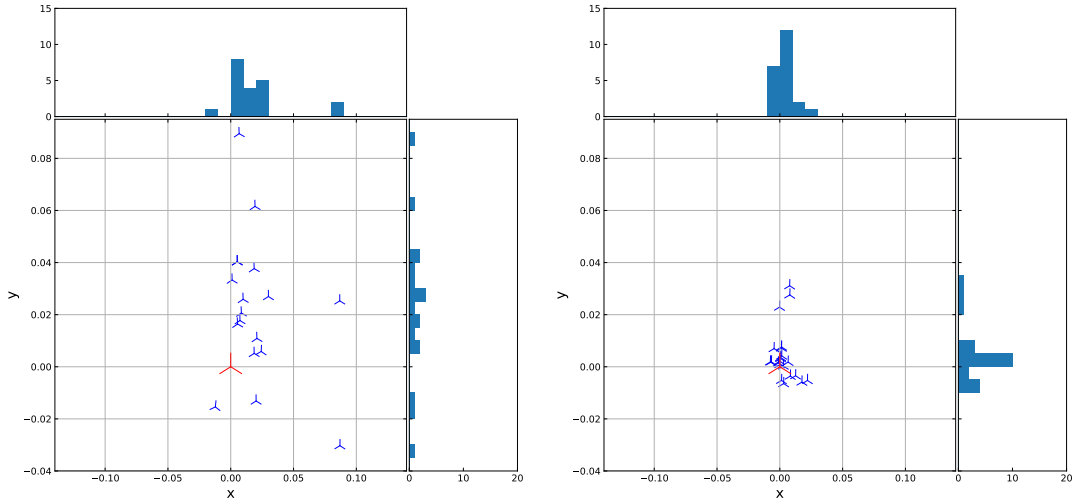
a: Positioning errors move_base

b: Positioning errors odometry-based method

Fig. 5.4.: Scatter and Histogram axis plots of the obtained positioning results at the first reference location for the two different methods

Figures 5.4 and 5.5 show a scatter plot of the positioning error of the robot at the two different locations. In both Figures, the positioning error of move_base is shown on the left side and the odometry-based method method on the right. In the move_base result, as can be seen from the graph in both the Figures there is a systematic error. This error affects all measurements in the same way. At the first pose above, most measurements overshot the target and landed slightly to the left. For the second pose it also overshot the target but landed slightly to the right (see histograms for x-axis above the scatter plot). The common denominator in this case is the clear overshooting of the

measurements. There is also some random error that affects the measurements in an unpredictable manner. The odometry-based method does not show such a clear systematic error. In Figure 5.5, some overshooting is noted, however, this is not visible in Figure 5.4, where the average and mean measurement is on target (although there are some outliers). There is also random error in both results, which is seen to reduce considerably in the results of the right-hand-sided plots. Random errors will always occur when doing measurements, however their magnitude can be minimized. Comparing the odometry-based method to move_base it can be said that it has *higher precision and medium trueness* meaning that it is more accurate. The odometry-based method is more accurate from a positional point of view. The rotational errors are also plotted but barely visible and are discussed in the next section.



a: Positioning errors move_base

b: Positioning errors odometry-based method

Fig. 5.5.: Scatter and Histogram axis plots of the obtained positioning results at the second reference location for the two different methods

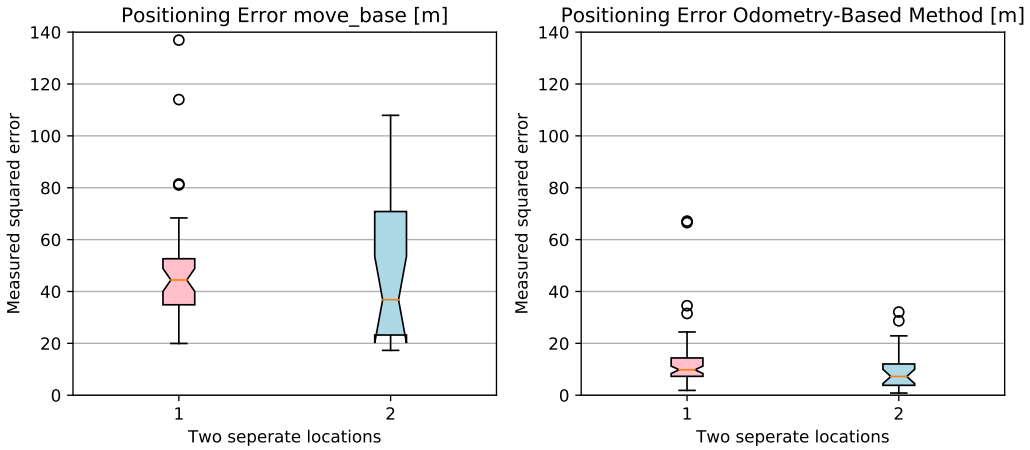


Fig. 5.6.: Box plot showing the position accuracy at both target locations using the two methods

Looking closer at the positional error of both methods at the two locations in the box plots in Figure 5.6 it can be seen, that the means and variances of both methods are clearly different. There are 6 outliers in the box plot of the odometry-based method, that lie in the move_base ranges with two suprassing the means of the move_base method. A more detailed view of the position error of the odometry-based method can be seen in Figure 5.7. We construct Table 5.2 by computing the measures of central tendency for both methods using the data of both locations. The plots in the Appendix A.8 and these measures show that both distributions have a positive *skew*, which comes from the fact that the mean constitutes the most weight, followed by the median and mode. This means, that there is a longer tail in the distribution to the right, which makes the mean less meaningful than with a symmetrical normal distribution. This skew is marked in the first location using the move_base approach, because of some outliers, which could be due to the small sample size.

The mean, median and mode *positional error* is seen to be quartered, and the range values have nearly been halved using the odometry-based method. Furthermore, 85% of positional errors using move_base and 7.3% using the odometry-based method are greater than 25 mm. The most frequently occurring position error for move_base is a *mode* of 4 cm, compared to 1 cm for that of the odometry-based method. This can also be observed in the histogram mentioned in the Appendix. Histograms plots the frequencies of positional square error in metres. The lack of symmetry as well as the clear outliers in both samples can also be observed in the *normal quantile-quantile Q-Q* plot mentioned in the Appendix. Because the ordered data points from our tests do not match up with the theoretical quantiles from the normal distribution.

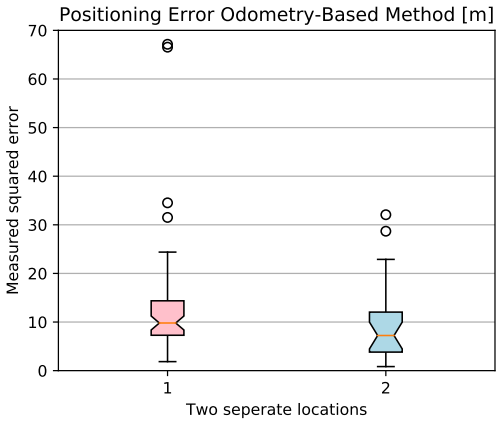


Fig. 5.7.: Box plot of the detailed position accuracy of the odometry-based method

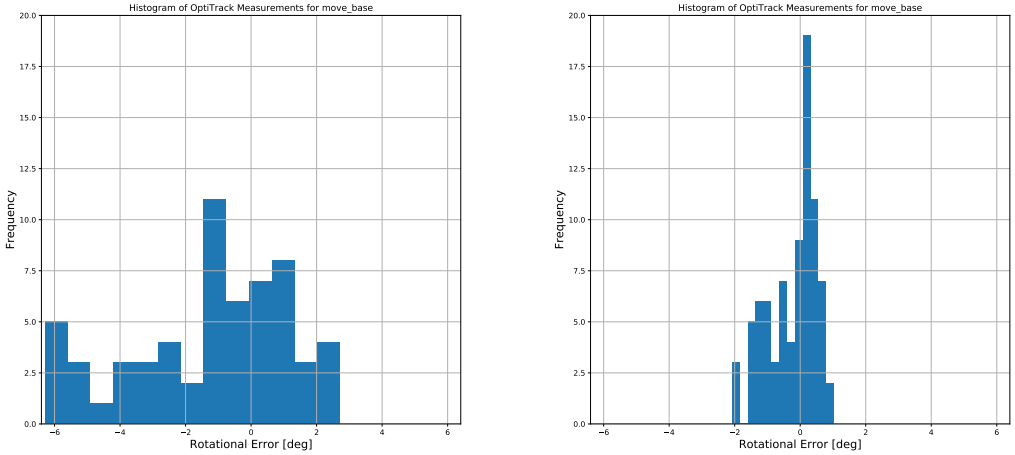
Measure	move_base	Odometry-based
Mean	4.86	1.24
Median	4.07	0.92
Mode	4.0	1.0
Range Interval	$[1.73, 13.69]$	$[0.08, 6.71]$
Range	11.96	6.63
Variance	0.07	0.01
Standard Deviation σ	2.59	1.12

Tab. 5.2.: Measures of central tendency for position errors

A way to test if the improvement is significant would be by conducting a statistical test. Due to the lack of symmetry in the normal distribution (*skewness*) and the possibility of outliers this can not be done easily and requires more work. A possible solution to both problems could be to employ a transformation. Such a transformation applies a function to every data point, such as a logarithm that reduces positive skewness. Removing the outliers in the datasets could be done by gathering more data, also at other locations with different orientations.

Rotation Error

In the scatter plots above it was difficult to grasp the rotation differences of both methods and compare them. Similar to the box plot in the previous section for the position error, Figure 5.8 shows histograms but for the rotation error over both target locations for both methods. For the move_base method there seems to be a systematic error towards a left rotation, which is reversed for the odometry-based method, as most measurements were inclined towards next to the origin on the right hand side. The rotation error is seen to be more prominent and wider spread for the move_base results. While the values on the left are contained within a big range interval of $[6.30, 2.71]$ with a total range of $9.01°$ as compared to $3.11°$ on the right. This range difference also suggests an improvement regarding random errors when using the odometry-based method.



a: Rotation errors move_base

b: Rotation errors odometry-based method

Fig. 5.8.: Histogram of the obtained rotation errors over both reference locations for the two different methods

The rotation mean, median and mode errors of both methods are different at the first location. Especially the variance at the first location is more prominent for the move_base method. At the second location, the means are same but the skewness and variance of

the move_base method is larger. It seems that there is one outlier present in the box plot for the move_base method that reports a rotation error of more than 6 degrees. While the variance has been divided by 13, the mean, median and mode are improved by a factor of 4.5 on average. The most frequently occurring angular error for the move_base navigation is a mode of 2.13°, as compared to 0.28° using the odometry-based method. This can also be observed in the histogram in the Appendix A.12. This histogram plots the frequencies of absolute angular error in degrees. However, the mean is high for both methods, the median is much lower.

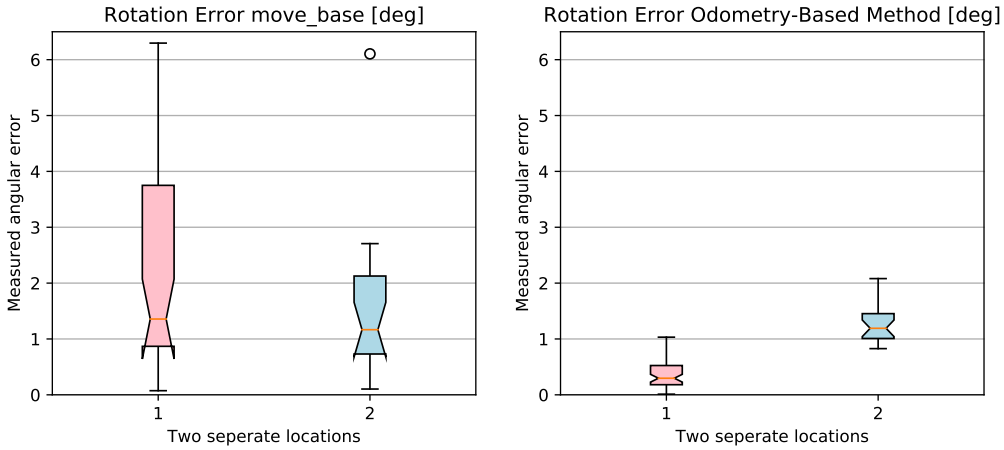


Fig. 5.9.: Box plot of the rotation accuracy at both target locations using the two methods

Measure	move_base	Odometry-based
Mean	2.05	0.61
Median	1.27	0.49
Mode	2.13	0.28
Range Interval	$[-6.30, 2.71]$	$[-2.08, 1.03]$
Range	9.00	3.11
Variance	3.21	0.24
Standard Deviation σ	1.79	0.49

Tab. 5.3.: Measures of central tendency for rotation errors

Comparing the methods using a 1.5° rotation error, we get a total of 45% of rotations that were bigger than the threshold for move_base. Using the same threshold for the odometry-based method yields a percentage rise by 6.1%. The distributions seen from the viewpoint of the histogram is less skewed than the ones in the position error plot. Although looking at the Q-Q plots in 5.11 and 5.12 we can clearly see the non-

linearity in the move_base distribution, the odometry-based method seems to fit more nicely to the quantiles of a normal distribution. The odometry-based method seems to have some skewness to the right and a thin tail, which can be seen in the Q-Q plot 5.12 as well. Therefore, it also requires more work to get a powerful test without violating a distribution assumption.

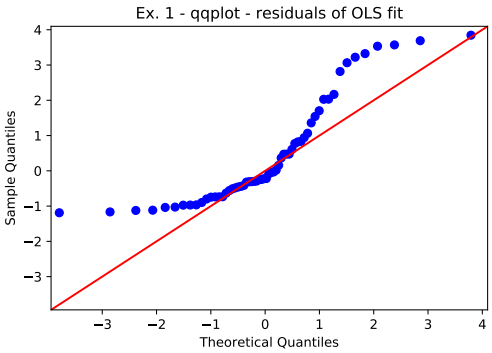


Fig. 5.11.: Normal Q-Q plot comparing rotation errors from move_base on the vertical axis to theoretical quantiles

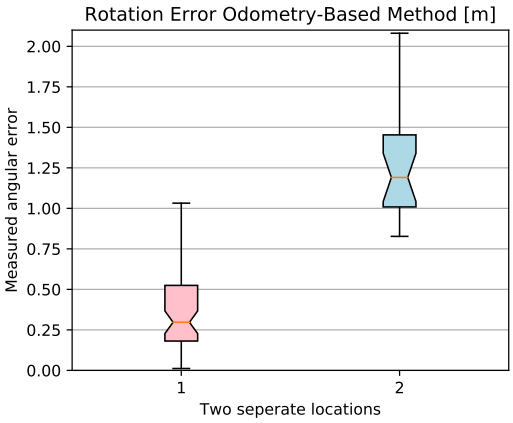


Fig. 5.10.: Box plot showing the detailed rotation accuracy of the odometry-based method

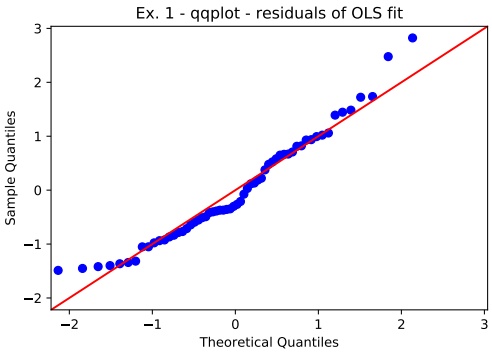


Fig. 5.12.: Normal Q-Q plot comparing rotation errors from odometry-based method on the vertical axis to theoretical quantiles

Discussion of Results

Both the position and rotation error is seen to improve for both *trueness and precision*, which results in an overall better accuracy using the odometry-based method. Both accuracy goals of $1°$ and $1\,\mathrm{cm}$ could not successfully be achieved. Although the requirements could not be achieved in this work, the improvements are considerable. The experimental evaluation in a industrial setting achieves a position error below $25\,\mathrm{mm}$ in 92.7%, and a rotation error below $1.5°$ in 93.9% of the tests. Although the significance of the improvement could not be statistically shown in this thesis due to skewed data and some possible outliers, the results show an improvement in both trueness and precision, which leads to an overall more accurate positioning. Furthermore, by using the pose graph-based approach for localization, we avoid manual teach-in of static reference scans that can degenerate in dynamic environments over time.

In oder to show a statistical significant improvement using the odometry-based method over the standard navigation stack requires some further statistical analysis. More experimental evaluation has to be done in order to determine how the methods can handle obstruction of the sensor and moving objects. The same experimental evaluation should also be carried out with other robotic platforms, especially robots with differential drives.

While testing we observed a possible benefit in the localization when driving further backwards as this would provoke a jump in the odometry pose, which probably is a correction from the pose-graph localization. Another benefit of driving further back is that the amount of rotation is reduced. In turn this could increase the error induced by the small amount of error that can happen when matching the angle to the goal, which ideally should be below $1\,\mathrm{mrad}$ but is not guaranteed. This needs to be investigated further and could lead to a improved version of the odometry-based algorithm.

6 Conclusion

This thesis project was undertaken to design and implement an extensible architecture for mobile robot navigation and evaluate the positioning accuracy of the complete system. It was expected, that the errors lie somewhere over $5\,\mathrm{cm}$ positioning and $1\,\mathrm{rad}$ rotational error, which can lead to process failures. Using the insights gained during design and implementation, the second goal of this thesis was to propose an algorithm that improves the final positioning accuracy and is able to handle changing environments. To show that the proposed algorithm improves the final positioning accuracy of the robot navigation an empirical experiment was conducted and discussed.

In this last chapter, the proposals and evaluations are discussed in retrospect. This allows to summarize the thesis project by discussing the achieved successes and what could be improved in future projects.

6.1 Summary of Thesis Findings

The proposed solution architecture in Chapter 3 has been designed and implemented with the architectural forces in mind. The forces were defined by defining the context of the system, stating its vision and functional requirements, and identifying the constraints of the environment. Most importantly, the quality attributes *modifiability, usability and testability* to be addressed were specified using scenarios to indicate the system's satisfaction of the stakeholders needs. For the solution architecture we then analyzed the robotic agents' task environment specification and specified requirements and agent types to get an appropriate architecture for navigation with final fine positioning. This appropriate *Mobile Base Task Control Pipeline Architecture* combines deliberative planning with reactive control in a layered hybrid approach, where multiple processes are executed in parallel.

After specifying the robotic agent and its environment, the architectural style micro services was chosen, mainly because of the positive consequences of the system subdivision. While this decision lead to a considerable time investment, in order to initially set up all the *nitty-gritty*[1] of this elaborate and distributed architecture, it payed off in important system qualities. The required quality attributes specified have then been achieved using tactics that were implemented in the system in order to achieve the quality goals specified earlier. All of the chosen tactics guided to a set of architectural decisions, out

[1] The specific practical details

of which the most important ones were discussed in detail.

The individual subsystems and individual components are explained using different views at varying levels of abstraction and from different perspectives. These perspectives show the divisions of the system (*building block view*), the behavior of important building blocks (*runtime view*) and different deployment (*deployment view*) strategies. Important interfaces and API's built are documented in detail. Four exensibility examples for the architecture are given, that modify or add behavior to the system. Along the way, technology decisions that had to be considered, which are documented together with the rationale. The architecture proposal finishes by showing the DevOps toolchain set up in the infrastructure of the Enterprise Lab, which enables CI/CD that automates the process of software testing, building and delivery, and also if needed infrastructure changes. Using this infrastructure and the containerized simulation solution, a developer can develop and test components locally in the simulator. Once a coding increment is done and the code committed to the repository, it gets deployed on the real robot.

This solution architecture for mobile robot navigation yields a robust navigation strategy that uses social navigation layers to plan paths from a starting to a target pose. These social navigation layers give the robot a more friendly behavior following the general preferences of human-robot interaction. To improve the final pose accuracy of mobile robots an algorithm for precise positioning at semantic poses based on the pose-graph localization is proposed in Chapter 4. This method improves the final pose accuracy of the standard navigation stack, which defines a $5\,\mathrm{cm}$ and $1\,\mathrm{rad}$ tolerance. The proposed algorithm converges an odometry pose with an goal pose using only minimal rotation and lateral movements in x direction, to be compatible with differential drive robots. During implementation an error with the laser scanner in use has been identified and fixed in a pull request that was submitted to the official driver GitHub repository.

Because of drift and estimation errors the proposed fine adjustment algorithm was evaluated experimentally in Chapter 5 using an external measurement system. The used evaluation methodology together with the proposed navigation solution architecture allows repeatable reproducibile results. Due to time constraints, only the positioning error of the robot was determined. Evaluation of the position error using the translational error in the x, y-plane yielded improvements in both *trueness and precision*. Achieving a positional error below $25\,\mathrm{mm}$ in 92.7% of the tests. The range of the position error was almost halved compared to the standard navigation. As for the rotational error, the angle difference in degrees was used. The results for rotation yielded better results, as 93.9% of the rotations ended up being less than $1.5°$ off. The range of the rotation error was almost divided by 3 compared to the standard navigation. There seem to be some outliers present in both errors, which increase the range intervals.

The position error of the standard navigation stack was determined to be below $13.69\,\mathrm{cm}$, but at best $1.73\,\mathrm{cm}$. As for the rotation error, the standard navigation achieved all poses with under $6.3°$ error. The odometry-based

strategy achieved all positions below $6.71\,\mathrm{cm}$, and at best with $0.08\,\mathrm{cm}$. For the rotation error, it achieved all poses with a maximum error of $2.08°$.

Compared to the results of [3] both errors seems rather large. However, it does not require to use a static reference scan and instead the desired outcome is achievable by updating the pose graph-approach continuously. Also, there are some limitations of the integrated platform that does not provide an IMU reading, nor a second laser scanner. The proposed algorithm works not only for holonomic wheel drives, but for all mobility types.

6.2 Future Work

The proposed solution architecture from Chapter 3 only lays a solid ground work to build upon. Although the system was designed as a generic solution for mobile robot navigation, it has only been deployed on the Mobility Base so far. For future work, we want to deploy it to other platforms that provide different sensors, such as 3D lidars or 3D cameras and different types of mobility types. Because of the modifiability and testability qualities of the architecture and support for rapid prototyping or template containers, the addition and evaluation of new sensors should be doable in one day.

This thesis has provided a deeper insight into the pose accuracy of a default ROS navigation stack designed using state-of-the-art algorithms. An ICP-based pose graph approach for simultaneous localization and mapping brings two major benefits from lifelong and continuous mapping to precise optimization-based localization. Future research should evaluate the accuracy of the localization system and compare it to the positioning error, in order to determine how well this system can estimate its own pose. Such an analysis provides insight into the limitations of the low-level controllers of the platform, similar to what [3] did. This should be a rather straight forward analysis, as most of the data is already captured. Furthermore, the impact of backwards movement along the x-axis should be investigated by extending the odometry-based strategy. This extended strategy should always perform that backwards movement for a greater distance and it should then be analyzed if that helps to improve the odometry-like localization. Finally, the SLAM system will be extended by fusing 3D cameras and an IMU with the laser scanner and see how much of an improvement this brings towards a better localization and positioning.

Bibliography

[1]Tobias Arndt, Jan Hochdoerffer, Emanuel Moser, Steven Peters, and Gisela Lanza. “Customer-driven Planning and Control of Global Production Networks–Balancing Standardisation and Regionalisation”. In: *Proceedings of the 18th Cambridge International Manufacturing Symposium*. Vol. 11. University of Cambridge Cambridge, UK. 2014, pp. 2014–12 (cit. on p. 1).

[2]Henning Kagermann, Johannes Helbig, Ariane Hellinger, and Wolfgang Wahlster. *Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Securing the future of German manufacturing industry; final report of the Industrie 4.0 Working Group*. Forschungsunion, 2013 (cit. on p. 1).

[3]Jörg Röwekämper, Christoph Sprunk, Gian Diego Tipaldi, et al. “On the position accuracy of mobile robot localization based on particle filters combined with scan matching”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 3158–3164 (cit. on pp. 1, 11, 12, 80, 89, 103).

[4]ISO Vim. “International vocabulary of basic and general terms in metrology (VIM)”. In: *International Organization* 2004 (2004), pp. 09–14 (cit. on p. 2).

[5]ISO ISO. “5725-1: 1994, Accuracy (trueness and precision) of measurement methods and results-Part 1: General principles and definitions”. In: *International Organization for Standardization, Geneva* (1994) (cit. on p. 2).

[6]Morgan Quigley, Ken Conley, Brian Gerkey, et al. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5 (cit. on p. 5).

[7]Manuel Serrano, Erick Gallesio, and Florian Loitsch. “Hop: a language for programming the web 2. 0”. In: *OOPSLA Companion*. 2006, pp. 975–985 (cit. on p. 5).

[8]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. 5).

[9]Kyle Johns and Trevor Taylor. *Professional microsoft robotics developer studio*. John Wiley & Sons, 2009 (cit. on p. 5).

[10]Sylvain Joyeux and Jan Albiez. “Robot development: from components to systems”. In: *6th National Conference on Control Architectures of Robots*. INRIA Grenoble Rhône-Alpes. Grenoble, France, May 2011, 15 p. (Cit. on p. 5).

[11]Richard Volpe, Issa AD Nesnas, Tara Estlin, et al. “CLARAty: Coupled layer architecture for robotic autonomy”. In: *NASA-JET PROPULSION LABORATORY*. Citeseer. 2000 (cit. on p. 5).

[12]Emmanouil Tsardoulias and Pericles Mitkas. “Robotic frameworks, architectures and middleware comparison”. In: *arXiv preprint arXiv:1711.06842* (2017) (cit. on pp. 5, 6).

[13]Wei Xie, Jiachen Ma, Mingli Yang, and Qi Zhang. “Research on classification of intelligent robotic architecture”. In: *Journal of Computers* 7.2 (2012), pp. 450–457 (cit. on p. 6).

[14]Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016 (cit. on pp. 6, 33, 39).

[15]Spyros G Tzafestas. “Mobile robot control and navigation: A global overview”. In: *Journal of Intelligent & Robotic Systems* 91.1 (2018), pp. 35–58 (cit. on p. 7).

[16]Ruffin White and Henrik Christensen. “ROS and Docker”. In: *Robot Operating System (ROS)*. Springer, 2017, pp. 285–307 (cit. on p. 8).

[17]Andrea Bauer, Dirk Wollherr, and Martin Buss. “Human–robot collaboration: a survey”. In: *International Journal of Humanoid Robotics* 5.01 (2008), pp. 47–66 (cit. on p. 8).

[18]Reza Abiri, Xiaopeng Zhao, Griffin Heise, Yang Jiang, and Fateme Abiri. “Brain computer interface for gesture control of a social robot: An offline study”. In: *2017 Iranian Conference on Electrical Engineering (ICEE)*. IEEE. 2017, pp. 113–117 (cit. on p. 8).

[19]Enric Cervera and Angel P Del Pobil. “ROSLab: sharing ROS code interactively with Docker and JupyterLab”. In: *IEEE Robotics & Automation Magazine* 26.3 (2019), pp. 64–69 (cit. on pp. 8, 62).

[20]Eugenio Guglielmelli. “Research Reproducibility and Performance Evaluation for Dependable Robots [From the Editor’s Desk]”. In: *IEEE Robotics & Automation Magazine* 22.3 (2015), pp. 4–4 (cit. on p. 8).

[21]Marina Indri, Fiorella Sibona, and Ludovico Orlando Russo. “P&P-Standard architecture to enable fast software prototyping for robot arms”. In: *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. IEEE. 2018, pp. 721–728 (cit. on p. 8).

[22]Pengfei Xu, Shaohuai Shi, and Xiaowen Chu. “Performance evaluation of deep learning tools in Docker containers”. In: *2017 3rd International Conference on Big Data Computing and Communications (BIGCOM)*. IEEE. 2017, pp. 395–403 (cit. on p. 9).

[23]Yan-Bin Jia. “Quaternions and rotations”. In: *Com S* 477.577 (2008), p. 15 (cit. on p. 9).

[24]MiR Robots. “MiRFleet Reference Guide”. In: (Jan. 2019) (cit. on p. 11).

[25]MiR Robots. “MiRHook User Guide”. In: (Feb. 2019) (cit. on p. 11).

[26]Ellon Mendes, Pierrick Koch, and Simon Lacroix. “ICP-based pose-graph SLAM”. In: *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE. 2016, pp. 195–200 (cit. on pp. 11, 15).

[27]Garry A Einicke and Langford B White. “Robust extended Kalman filtering”. In: *IEEE Transactions on Signal Processing* 47.9 (1999), pp. 2596–2599 (cit. on p. 11).

[28]Eric A Wan and Rudolph Van Der Merwe. “The unscented Kalman filter for nonlinear estimation”. In: *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*. Ieee. 2000, pp. 153–158 (cit. on p. 11).

[29]Dieter Fox, Wolfram Burgard, and Sebastian Thrun. “Markov localization for mobile robots in dynamic environments”. In: *Journal of artificial intelligence research* 11 (1999), pp. 391–427 (cit. on p. 11).

[30]Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. “Monte carlo localization for mobile robots”. In: *ICRA*. Vol. 2. 1999, pp. 1322–1328 (cit. on p. 11).

[31]JaeMu Yun, SungBu Kim, and JangMyung Lee. “Robust positioning a mobile robot with active beacon sensors”. In: *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*. Springer. 2006, pp. 890–897 (cit. on p. 11).

[32]Ilan Shimshoni and Igal Loevsky. *Localization method for mobile robots based on landmarks*. US Patent 8,930,127. Jan. 2015 (cit. on p. 11).

[33]José Jean-Paul Zanlucchi de Sousa, Rodrigo Hiroshi Murofushi, Lucas Henriques Silva, and Gustavo Rezende Silva. “Petri Net Inside RFID Database Integrated with RFID Indoor Positioning System for Mobile Robots Position Control.” In: *PNSE@ Petri Nets*. 2017, pp. 157–176 (cit. on p. 11).

[34]Robert Sim and Gregory Dudek. “Mobile robot localization from learned landmarks”. In: *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No. 98CH36190)*. Vol. 2. IEEE. 1998, pp. 1060–1065 (cit. on p. 11).

[35]Maren Bennewitz, Cyrill Stachniss, Wolfram Burgard, and Sven Behnke. “Metric localization with scale-invariant visual features using a single perspective camera”. In: *European Robotics Symposium 2006*. Springer. 2006, pp. 195–209 (cit. on p. 12).

[36]Pantelis Elinas and James J Little. “σMCL: Monte-Carlo Localization for Mobile Robots with Stereo Vision.” In: *Robotics: Science and Systems*. 2005, pp. 373–380 (cit. on p. 12).

[37]Morgan Quigley, David Stavens, Adam Coates, and Sebastian Thrun. “Sub-meter indoor localization in unmodified environments with inexpensive sensors”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 2039–2046 (cit. on p. 12).

[38]Paul J Besl and Neil D McKay. “Method for registration of 3-D shapes”. In: *Sensor fusion IV: control paradigms and data structures*. Vol. 1611. International Society for Optics and Photonics. 1992, pp. 586–606 (cit. on p. 12).

[39]Su Pang, Daniel Kent, Xi Cai, et al. “3D Scan Registration Based Localization for Autonomous Vehicles-A Comparison of NDT and ICP under Realistic Conditions”. In: *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*. IEEE. 2018, pp. 1–5 (cit. on p. 12).

[40]Kirill Krinkin, Anton Filatov, Art yom Filatov, Artur Huletski, and Dmitriy Kartashov. “Evaluation of modern laser based indoor slam algorithms”. In: *2018 22nd Conference of Open Innovations Association (FRUCT)*. IEEE. 2018, pp. 101–106 (cit. on p. 13).

[41]Sukkpranhachai Gatesichapakorn, Jun Takamatsu, and Miti Ruchanurucks. “ROS based autonomous mobile robot navigation using 2D LiDAR and RGB-D camera”. In: *2019 First international symposium on instrumentation, control, artificial intelligence, and robotics (ICA-SYMP)*. IEEE. 2019, pp. 151–154 (cit. on p. 14).

[42]Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. “Real-Time Loop Closure in 2D LIDAR SLAM”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1271–1278 (cit. on p. 14).

[43]Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. “Improved techniques for grid mapping with rao-blackwellized particle filters”. In: *IEEE transactions on Robotics* 23.1 (2007), pp. 34–46 (cit. on p. 14).

[44]S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. “A Flexible and Scalable SLAM System with Full 3D Motion Estimation”. In: *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE. Nov. 2011 (cit. on p. 14).

[45]Rauf Yagfarov, Mikhail Ivanou, and Ilya Afanasyev. “Map comparison of lidar-based 2d slam algorithms using precise ground truth”. In: *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. IEEE. 2018, pp. 1979–1983 (cit. on p. 14).

[46]Joao Machado Santos, David Portugal, and Rui P Rocha. “An evaluation of 2D SLAM techniques available in robot operating system”. In: *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE. 2013, pp. 1–6 (cit. on p. 14).

[47]Edwin B Olson. “Real-time correlative scan matching”. In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 4387–4393 (cit. on p. 14).

[48]Kurt Konolige, Giorgio Grisetti, Rainer Kümmerle, et al. “Efficient sparse pose adjustment for 2D mapping”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 22–29 (cit. on p. 15).

[49]Regis Vincent, Benson Limketkai, and Michael Eriksen. “Comparison of indoor robot localization techniques in the absence of GPS”. In: *Detection and Sensing of Mines, Explosive Objects, and Obscured Targets XV*. Vol. 7664. International Society for Optics and Photonics. 2010, 76641Z (cit. on p. 15).

[50]Uthman Baroudi, M Alharbi, K Alhouty, H Baafeef, and K Alofi. “Cloud-Based Autonomous Indoor Navigation: A Case Study”. In: *arXiv preprint arXiv:1902.08052* (2019) (cit. on p. 15).

[51]Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. “A tutorial on graph-based SLAM”. In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), pp. 31–43 (cit. on p. 15).

[52]Dieter Fox. “KLD-sampling: Adaptive particle filters”. In: *Advances in neural information processing systems*. 2002, pp. 713–720 (cit. on p. 16).

[53]Rodrigo Longhi Guimarães, André Schneider de Oliveira, João Alberto Fabro, Thiago Becker, and Vincius Amilgar Brenner. “ROS navigation: Concepts and tutorial”. In: *Robot Operating System (ROS)*. Springer, 2016, pp. 121–160 (cit. on p. 17).

[54]Kaiyu Zheng. “ROS Navigation Tuning Guide”. In: *CoRR* abs/1706.09068 (2017). arXiv: 1706.09068 (cit. on pp. 17, 65).

[55]Jonathan Mumm and Bilge Mutlu. “Human-robot proxemics: physical and psychological distancing in human-robot interaction”. In: *Proceedings of the 6th international conference on Human-robot interaction*. ACM. 2011, pp. 331–338 (cit. on p. 17).

[56]Javier V Gómez, Nikolaos Mavridis, and Santiago Garrido. “Social path planning: Generic human-robot interaction framework for robotic navigation tasks”. In: *2nd Intl. Workshop on Cognitive Robotics Systems: Replicating Human Actions and Activities*. 2013 (cit. on pp. 17, 18).

[57]Chi-Pang Lam, Chen-Tun Chou, Kuo-Hung Chiang, and Li-Chen Fu. “Human-centered robot navigationtowards a harmoniously human–robot coexisting environment”. In: *IEEE Transactions on Robotics* 27.1 (2010), pp. 99–112 (cit. on p. 17).

[58]David V Lu, Daniel B Allan, and William D Smart. “Tuning cost functions for social navigation”. In: *International Conference on Social Robotics*. Springer. 2013, pp. 442–451 (cit. on p. 18).

[59]Rachel Kirby, Reid Simmons, and Jodi Forlizzi. “Companion: A constraint-optimizing method for person-acceptable navigation”. In: *RO-MAN 2009-The 18th IEEE International Symposium on Robot and Human Interactive Communication*. IEEE. 2009, pp. 607–612 (cit. on p. 18).

[60]Jared Herzog. “Software Architecture in Practice Third Edition Written by Len Bass, Paul Clements, Rick Kazman.” In: *ACM SIGSOFT Software Engineering Notes* 40.1 (2015), pp. 51–52 (cit. on pp. 21, 26, 27, 44, 48, 49).

[61]ISO ISO. “IEC/IEEE Systems and Software Engineering: Architecture description”. In: *ISO/IEC/IEEE 42010: 2011 (E)(Revision of ISO/IEC 42010: 2007 and IEEE Std 1471-2000)* (2011) (cit. on p. 21).

[62]Andreas Spillner and Tilo Linz. *Basiswissen softwaretest*. dpunkt, 2012 (cit. on p. 26).

[63]Chris Richardson. *Microservices patterns*. Manning Publications Shelter Island, 2018 (cit. on pp. 41, 43).

[64]Budd Timothy. *Introduction to object-oriented programming*. Pearson Education India, 2008 (cit. on p. 45).

[65]*JupyterRos - GitHub ReadMe*. https://github.com/RoboStack/jupyter-ros. Online;Accessed: 2020-02-10 (cit. on p. 62).

[66]*summit_xl_common - GitHub ReadMe*. https://github.com/RobotnikAutomation/summit_xl_common. Online;Accessed: 2020-02-10 (cit. on p. 65).

[67]*navigation_layers - GitHub ReadMe*. https://github.com/DLu/navigation_layers. Online;Accessed: 2020-02-10 (cit. on p. 66).

[68]*Operation Instructions: OMDxxx-R2000 Ethernet communication protocol - Protocol version 1.04*. Pepperl+Fuchs GmbH. 2019 (cit. on p. 77).

[69]*slam_toolbox - GitHub ReadMe*. https://github.com/SteveMacenski/slam_toolbox. Online;Accessed: 2020-01-16 (cit. on p. 79).

[70]*base_local_planner - ROS Wiki*. http://wiki.ros.org/base_local_planner. Online;Accessed: 2020-01-13 (cit. on p. 80).

List of Figures

1.1 Co-bot scenario: Precisely visit different workplaces and interact with humans, objects and other mobile or gripping robots 1

2.1 Rotation around x-axis then around y-axis 10
2.2 Rotation around y-axis then around x-axis 10
2.3 Two layers in the ICP-based pose-graph SLAM system 15

3.1 Context Diagram shows how the software system in scope fits into the world around it 23
3.2 Modifiability scenario for improving an existing component 27
3.3 Modifiability scenario for developing a coding increment as ROS node 28
3.4 Usability scenario for using the system productively 29
3.5 Usability scenario for deploying the system in a new environment 29
3.6 Testability scenario for executing the test suite 30
3.7 Testability scenario for executing system tests 30
3.8 Mobility Base with and without Baxter Robot mounted on top 33
3.9 Overview goal-based agent `move_base` and both utility-based agents `global` and `local planners`. Image courtesy of Open Source Robotics Foundation OSRF downloaded from https://wiki.ros.org/move_base in January 2020. 38
3.10 Mobile Base Task Control Architecture 40
3.11 Tree overview of the applied modifiability tactics 44
3.12 Tree overview of the applied usability tactics 47
3.13 API gateway that is the single entry point for all clients 47
3.14 Tree overview of the applied testability tactics 48
3.15 Coordination model: interactions between elements 51
3.16 Container view zooms into navigation system in scope showing the high-level technical building blocks at level 1 56
3.17 GUI Dashboard 58
3.18 GUI for mapping, exploration and navigation 59
3.19 GUI for creating a known pose 60
3.20 Data schema for known poses stored in the Known Poses DB 61
3.21 Jupyter Notebook showing an interactive experience for robotics research 62
3.22 Component diagram zooms into ROS Core subsystem showing the building components 63

3.23 Social navigation plugin layer added to costmaps seen in RViz above and the scenario in Gazebo in the image below . . . 66
3.24 Class diagram of the accuracy node . . . 67
3.25 Accuracy node interactions . . . 67
3.26 Logistic curve to compute a safe speed . . . 68
3.27 UML Activity diagram for adding an (in)accurate pose . . . 69
3.28 UML Activity diagram for reaching a pose (in)accurately . . . 70
3.29 Deployment diagram illustrating how containers in the static model are mapped to infrastructure . . . 71
3.30 Deployment diagram illustrating how containers in the static model are mapped to another infrastructure that does not provide access to its OS . . . 72
3.31 Continuous Integration / Continuous Deployment Overview . . . 75

4.1 Polar coordinate system with direction of scan head rotation . . . 77
4.2 LiDAR scanning from 0 ($0°$) to 2π ($360°$) with an angle increment of $\frac{\pi}{4}$ ($45°$) . . . 78
4.3 LiDAR scanning from 0 to 2π . . . 78
4.4 Navigation reaches the goal with a tolerance . . . 82
4.5 Compute the angle θ to the goal position . . . 83
4.6 Sectors that if odom has perfect angle will trigger backward motion . . . 83
4.7 Navigation overshot target and landed in front and to the left of the goal . . . 84
4.8 Difference has to be smaller than $\frac{\pi}{4}$ . . . 84
4.9 Turn base's x-axis towards goal pose . . . 84
4.10 Drive forward towards goal . . . 86
4.11 Reaching goal height with different orientation . . . 86
4.12 Final rotation to align with goal pose orientation . . . 87

5.1 2D Occupancy grid of the evaluation environment with a $5\,\mathrm{cm}$ resolution . . . 90
5.2 Lab environment that was used as evaluation site . . . 90
5.3 Three of the 8 OptiTrack Motion Capture System cameras . . . 90
5.4 Scatter and Histogram axis plots of the obtained positioning results at the first reference location for the two different methods . . . 93
5.5 Scatter and Histogram axis plots of the obtained positioning results at the second reference location for the two different methods . . . 94
5.6 Box plot showing the position accuracy at both target locations using the two methods . . . 94
5.7 Box plot of the detailed position accuracy of the odometry-based method . . . 95
5.8 Histogram of the obtained rotation errors over both reference locations for the two different methods . . . 96
5.9 Box plot of the rotation accuracy at both target locations using the two methods . . . 97
5.10 Box plot showing the detailed rotation accuracy of the odometry-based method . . . 98
5.11 Normal Q-Q plot comparing rotation errors from move_base on the vertical axis to theoretical quantiles . . . 98

5.12 Normal Q-Q plot comparing rotation errors from odometry-based method on the vertical axis to theoretical quantiles 98

A.1 Related work analysis for pose accuracy with laser and wireless receivers 121
A.2 Related work analysis for pose accuracy with 2D/3D Cameras 122
A.3 Context Diagram shows how the software system in scope fits into the world around it 129
A.4 Container View zooms into navigation system in scope showing the high-level technical building blocks 130
A.5 Component diagram zooms into ROS Core showing the building components . . . 131
A.6 Deployment diagram illustrating how containers in the static model are mapped to infrastructure 132
A.7 Starting an accuracy measurement 133
A.8 Dashboard ROS Topics 134
A.9 Scatter Plot for OptiTrack measurements 135
A.10 Computed position and rotation error 136
A.11 Histogram position error move_base 149
A.12 Histogram rotation error move_base 149
A.13 Histogram position error odometry-based method 149
A.14 Histogram rotation error odometry-based method 149
A.15 Normal Q-Q plot comparing position errors from move_base on the vertical axis to theoretical quantiles 150
A.16 Normal Q-Q plot comparing position errors from odometry-based method on the vertical axis to theoretical quantiles 150
A.17 Q-Q plot comparing position errors from move_base method on the vertical axis to odometry-based method 150
A.18 Normal Q-Q plot comparing rotation errors from move_base on the vertical axis to theoretical quantiles 150
A.19 Normal Q-Q plot comparing rotation errors from odometry-based method on the vertical axis to theoretical quantiles 150
A.20 Q-Q plot comparing rotation errors from move_base method on the vertical axis to odometry-based method 150
A.21 Navigation stack pipeline 151

List of Tables

3.1 Worker Persona Specification . . . 24
3.2 Administrator Persona Specification . . . 25
3.3 Developer Persona Specification . . . 25
3.4 Constraints of the Environment . . . 32
3.5 PEAS: Task environment specification for the Mobility Base . . . 36
3.6 Requirements for an appropriate architecture for the navigating agent . . . 37
3.7 Application subdivision in many subprograms . . . 50
3.8 Coordination between two environments: ROS and non-ROS programs . . . 51
3.9 API Design: Communication Mechanism for the Known Pose API . . . 52
3.10 Representation of point clouds for adaptability . . . 53
3.11 Representation of measurement data . . . 55
3.12 Binding time of values that are likely to change . . . 55
3.13 Overview of microservice components and subsystems . . . 58
3.14 Operations of the Known Poses API available to developers . . . 60
3.15 Components of the ROS Core System . . . 64

5.1 Test protocol for evaluation . . . 92
5.2 Measures of central tendency for position errors . . . 95
5.3 Measures of central tendency for rotation errors . . . 97

A.1 Source code resources . . . 146
A.2 Artifact resources . . . 146
A.3 Docker images . . . 146
A.4 Documentation resources . . . 147
A.5 Container runtime resources . . . 148

List of Listings

4.1 Local planner parameters for goal tolerances 80

A.1 Build Docker image for slam_toolbox 137
A.2 YAML config for running the image 137
A.3 Call serialization service with map name 138
A.4 YAML config for deserializing a specified map 138
A.5 ENUM of ProcessType 138
A.6 Call deserialization service with yaml config 138
A.7 GlobalPlanner configuration 139
A.8 Dynamic Window Approach DWA local planner configuration 139
A.9 Timed Elastic Band TEB local planner configuration 140
A.10 Common Costmap Parameters 142
A.11 Global Costmap 144
A.12 Local Costmap 144

List of Algorithms

1 Pose Graph Odometry Accurate Goal Pose Convergence 82
2 Align with angle to goal θ by minimizing rotation required 85
3 Compute angle to goal from current pose 85
4 Drive along robots' x-axis to goal position forwards 85
5 Final rotation on spot to align odometry to goal pose 87

Appendix

A

A.1 Related Work Analysis for Pose Accuracy

YoP	Paper		Technology	Accurate Localization & Positioning (0-3)	Dealing with Changes (0-3)	Planning Path & Accurately Executing it (0-3)	Explore Autonomously (0/1)	Define Missions & Semantic Poses (0/1)	Continuous (Lifelong) Mapping (0/1)	Benchmark Accuracy (0/3)	Flexible Architecture (0-3)	Markerless (0/1)	Strategy & Autonomous Navigation Control (0-3)
Laser													
2015	[1]	Rowekamper et al.	Laser/ICP	✔✔✔ < 15 mm < 0.15°	✔✔✔ Heavy changes	✔✔✔	✘	✘	✘	✔✔✔ Nice benchmark protocol	✘	✔	✘
RFID/WiFi/GPS or other wireless receivers													
2012	[2]	Shimshoni et al.	Landmark bearings	✔✔ < 3.8 cm	✔ Isolated changes	✔ Restricted due to localization	✔	✘	✘	✔✔ No orientation	✘	✘	✘
2006	[3]	Yun et al.	Active Beacons	✔✔✔ < 5mm	✔✔✔ Heavy changes	✔✔✔	✘	✘	✘	✔✔ No orientation	✘	✘	✘
2013	[4]	de Sousa et al.	RFID IPS	✔✔ < 4.7 cm	✔ Isolated changes	✔ Only lines	✘	✘	✘	✔✔ No orientation	✘	✘	✔ Only ↑/↓ movements
			Table Key	0-1 cm ✔✔✔ 1-10 cm ✔✔ 10+ cm ✔	Heavy ✔✔✔ Moderate ✔✔ Isolated ✔	Accurate ✔✔✔ Moderate ✔✔ Restricted ✔	Yes ✔ No ✘	Yes ✔ No ✘	Yes ✔ No ✘	Everything ✔✔✔ Just Position ✔✔ Restricted ✔	Yes ✔ No ✘	Yes ✔ No ✘	Yes ✔ No ✘

Fig. A.1.: Related work analysis for pose accuracy with laser and wireless receivers

YoP	Paper		Technology	Accurate Localization & Positioning (0-3)	Dealing with Changes (0-3)	Planning Path & Accurately Executing it (0-3)	Explore Autonomously (0/1)	Define Missions & Semantic Poses (0/1)	Continuous (Lifelong) Mapping (0/1)	Benchmark Accuracy (0/3)	Flexible Architecture (0-3)	Markerless (0/1)	Strategy & Autonomous Navigation Control (0-3)
2D Cameras													
2011	[5]	Sim et al.	Landmark camera	✔ < 6.8 cm	✘	✘ No motion	✘	✘	✘	✔✔ No orientation	✘	✔	✘
2006	[6]	Bennewitz et al.	SIFT Monocular Camera	✔ < 39 cm < 4.5°	✘	✔✔ Grows over time	✘	✘	✘	✔✔ No external sensors	✘	✔	✘
2005	[7]	Elinas et al.	SIFT Stereo Camera	✔ < 20 cm < 6°	✘	✘	✘	✘	✘	✔✔ No external sensors	✘	✔	✘
2010	[8]	Quigley et al.	Low Cost Cameras (fusion)	✔ < 0.73 m	✔✔ Moderate changes	✘ No motion	✘	✘	✘	✔✔ No orientation	✘	✔	✘
3D Cameras													
2018	[9]	Pang et al.	3D Scan ICP/ NDT	✔✔✔ < 40 mm	✔✔✔ Heavy changes	✔✔✔	✘	✘	✘	✔✔ No orientation	✘	✔	✘
Table Key				0-1 cm ✔✔✔ 1-10 cm ✔✔ 10+ cm ✔	Heavy ✔✔✔ Moderate ✔✔ Isolated ✔	Accurate ✔✔✔ Moderate ✔✔ Restricted ✔	Yes ✔ No ✘	Yes ✔ No ✘	Yes ✔ No ✘	Everything ✔✔✔ Just Position ✔✔ Restricted ✔	Yes ✔ No ✘	Yes ✔ No ✘	Yes ✔ No ✘

Fig. A.2.: Related work analysis for pose accuracy with 2D/3D Cameras

A.2 Guiding Quality Design Decisions: Design Checklists

A.2.1 Modifiability

This checklist supports the design and analysis process for modifiability.

Allocation of Responsibilities

Determine which changes or categories thereof are likely to occur. To make that change,

- which responsibilities would need to be added, modified or deleted?
- what responsibilities need to be considered for impact?

Add *3D Mapping or Gesture Control* Requires some responsibilities that need to be added to the ROS environment. Defined interfaces would not change. No other responsibilities are affected by that change.

Change Navigation Behavior Requires to modify only responsibilities in the ROS node *move_base*. Defined interfaces would not change. No other responsibilities are affected by that change.

Improve Fine Adjustment at Target Pose Requires to modify only responsibilities in the ROS node *accuracy*. Defined interfaces would most likely not change. Depending on what the improved algorithm needs as additional data. No other responsibilities are affected by that change.

Coordination Model

Which functionality or quality attribute can change at runtime? How does it affect coordination?

Only information being communicated can change at runtime over the fixed defined interfaces and communication protocols.

Which devices, protocols and communication paths for coordination are likely to change?

The platform on which NavAjust runs can change as well as the deployment structure. This requires a coordination model that reduces coupling, such as dynamic lookup of modules, masking interface identity changes, convert syntax of a service into another form, removing producer's knowledge of its consumers and defer binding to as late as possible.

- Reducing coupling: Use an intermediary
  - Dynamic lookup of modules: software defined networking adds deployment flexibility (Docker SDN)
  - Masking interface identity changes: broker to make connection between modules (ROS Master)
  - Convert syntax of a service into another form: prevent changes from inside ROS environment to propagate out (ROSBridge)

  - Removing producer's knowledge of its consumers: publish-subscribe wherever possible (ROS Topics, MQTT)

- Defer binding to as late as possible:
  - Configuration files at deployment time (Environment Variables)
  - Startup time binding of values (Docker/Docker-Compose)

Data Model

Which changes can occur to the data abstractions, operations and their properties? Which changes involve the creation, initialization, persistence, manipulation, translation, or destruction of data abstractions?

Add 3D Information to Point Cloud Data for Known Poses This change would be made by a developer. The change would involve switching from 2D to 3D point clouds. For the point cloud representation, `Point Cloud Library Format PCD`, was chosen in version 0.7 (PCD_V7) for the JSON database, which specifies a point cloud using a header that identifies and declares certain properties of the point cloud data. PCD can already store n-D point clouds. Other formats such as PLY, STL, OBJ, X3D and many others do not offer the flexibility and speed of PCD files. The data abstractions can be left unchanged. Only the algorithm in the accuracy ROS node would be affected.

Mapping among Architectural Elements

The current mapping of functionality to computational elements (e.g. processes, threads, processors) is at runtime. Some execution dependencies are present in the ROS environment. The ROS environment can be seen as a data transformation and control pipeline. Data that comes from the sensors is processed and mapped to internal models. Decisions are made based on that models and a users goals, and converted into motion controls. Adding, deleting or modifying a node inside that chain therefore needs careful consideration.

Resource Management

Adding *3D Mapping or Gesture Control* might require more graphical processing resources. In case of this requirement, the system or single modules can be moved to a more powerful computer, as all modules are encapsulated and bindings are deferred to an extent that allows different deployments. Network resources have to be checked, if big amounts of data or time-critical data is transferred.

Binding Time

Each foreseen change that adds adds the functionality mentioned or changes the fine adjustment without changing the interfaces can be made at runtime.

Choice of Technology

The modifications are made easier by the technology choices ROS, Docker, MQTT, OpenAPI. ROS, MQTT and OpenAPI are open standards. Docker furthermore allows relocatable containers that can be rebuilt anytime and deployed at a whim, which is critical to protect from vendor lock-in.

A.2.2 Usability

This checklist supports the design and analysis process for usability.

Allocation of Responsibilities

Additional system responsibilities have been allocated to assist the user in

- learning how to use the system
- efficiently achieving the task at hand
- adapting and configuring the system
- recovering from user and system errors

Change User Interface Separating the user interface aids in rapid UI prototyping and facilitate experimentation with UI. An API gateway as a single entry point for clients handles requests in two ways.

Coordination Model

How do the system coordination properties *timeliness, currency, completeness, correctness and consistency* affect the user?

Long-running events should be canceled instantly if a user choses to abort the task.

Data Model

Determine the major data abstractions that are involved with user-perceiveable behavior.

The data abstractions are designed to support *cancel and aggregate* operation. Cancel operation for long-running navigation and exploration tasks. In order to avoid repetitive actions of the user, an aggregate operation to delete multiple not needed known poses is abstracted. No need for an *undo* operation could be envisioned in the system.

Mapping among Architectural Elements

For the end user the architectural elements seem to be cloud-based and should not affect the ways in which the user interacts with the system.

Resource Management

The user is not able to adapt or configure the system's use of resources.

Binding Time

The user can decide at run-time the system's deployment strategy and configuration.

Choice of Technology

The usability is made easier by separating the different UIs from the rest using an nginx API Gateway. HoloLens and VueJS are specifically designed with usability in mind by experts in the field. Measurment results are outputted as JSON for increased compatibility with analysis tools.

A.2.3 Testability

This checklist supports the design and analysis process for testability.

Allocation of Responsibilities

All system responsibilities are to be tested on their own. The most critical system responsibilities that need to be most thorougly tested are all related to the pose accuracy subsystem. Allocation of additional responsibilities for:

- execution of test suite
- log all activities that could lead to a fault or unexpected behavior
- control and observe relevant system state for testing

Testing modules Provide templates to self-test different module types such as ROS Nodes (Python3.6, C++11) or pure Python3.6 services using unit tests. Isolation of modules is key, to exclude external influences.

Testing Pose Accuracy with Navigation Goes hand-in-hand with modifiability to make sure that the allocation of functionality provides high cohesion, low coupling, strong separation of concerns, and low structural complexity. This allocation of functionality comes as an external test that requires a user to manually define a precise pose to be re-created in N runs. The test suite then drives to different locations in known map randomly and tries to re-create the manuelly defined pose as accurately as possible.

Coordination Model

Coordination and communication mechanisms for the system support the execution of the test suite and capture the results within a system. Activity that resulted in a fault within a system are captured in log files of the concerned module (container logs). Injection and monitoring of state into the communication channels within a system is supported by echoing ROS topics or message queues. The mechanisms do not introduce needless nondeterminism.

Data Model

Modules should get and produce data as defined by their interfaces. Data abstractions for the known pose API, final measurement data, the localization estimations and navigation paths are the data abstractions that have to be tested in order to ensure correct operation of the system. All these values are capturable and injectable into the system, such that a fault may be recreated (Record/playback ROS bag, Measurements and Known

Poses as JSON output). Also, all creation, initialization, persistence, manipulation, translation and destruction of instances of these data abstractions can be exercised and captured.

Mapping among Architectural Elements

Possible mapping of architectural elements not considered.

Resource Management

Sufficient resources to execute the test suite and capture the results is available if the system can run on the deployed hardware. The test environment has to be representative of the environment in which the system will run.

Binding Time

Modules that are bound at runtime can be tested in that late-bound context. Late bindings can be captured in the log files of the concerned module.

Choice of Technology

ROS with Gazebo and Docker as Technologies allows sandbox testing. Using such a virtualized resource that simulates the imortant aspects of the real world system offers the benefit of building a version of the resource whose behavior is under the developers control. Virtualization provides the ability to operate the system that it has no permanent consequences or that any consequences can be rolled back. JSON as data-interchange format allows humans to read and write and machines to parse and generate data easily.

A.3 Cohesion Analysis

A.3.1 Known Pose API

"This module's purpose is to provide known poses to various clients by gathering the robot current state and sensor readings."

This module has following cohesion bindings

1. compound sentence: sequential, due to a particular order to be used when adding a pose and gathering current state and sensor readings
2. words relating to time: none
3. sentence predicate without object: gathering different objects has functional binding because there is a cooperation of 2 elements to carry out a single function
4. No temporal binding

A.3.2 Web Dashboard Application

"This module's purpose is to serve the UI to the administrator to create and evaluate semantic poses, and see the robot current state."

This module has following cohesion bindings

1. compound sentence: sequential, due to a particular order to be used when evaluating a pose
2. words relating to time: none
3. sentence predicate without object: none
4. No temporal binding

A.3.3 Accuracy ROS Node

"This module's purpose is to perform a qualitative fine adjustment after reaching the target pose of a mobile robot."

This module has following cohesion bindings

1. compound sentence: none
2. words relating to time: after reaching the target pose has sequential binding
3. sentence predicate without object: none
4. No temporal binding

A.3.4 Pose Evaluator

"This module's purpose is to compute the positional and rotational error of the final pose."

This module has following cohesion bindings

1. compound sentence: none, positional and rotation error has no sequence
2. words relating to time: after reaching the target pose has sequential binding
3. sentence predicate without object: none
4. No temporal binding

A.4 Detailed Architectural Views

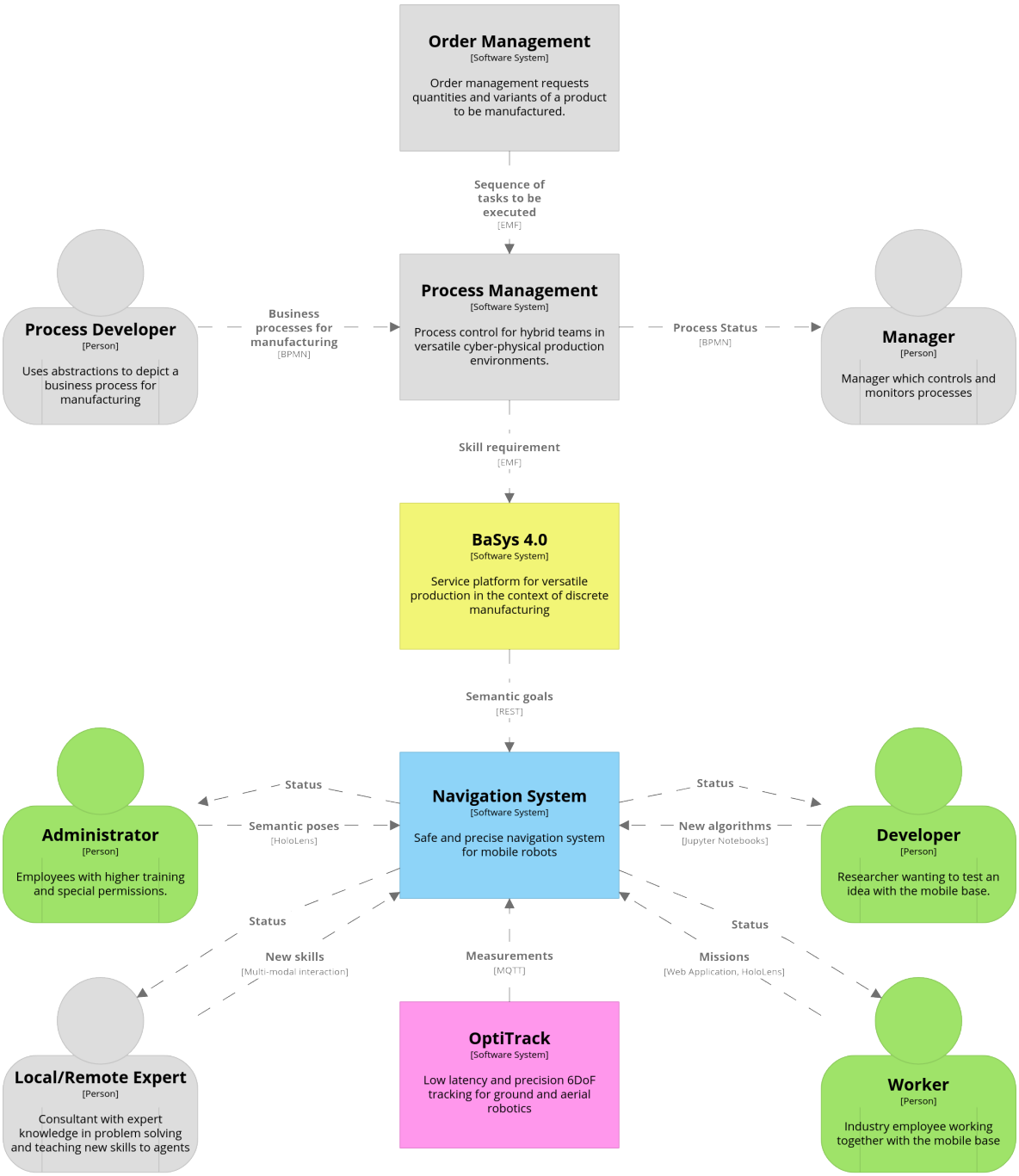


Fig. A.3.: Context Diagram shows how the software system in scope fits into the world around it

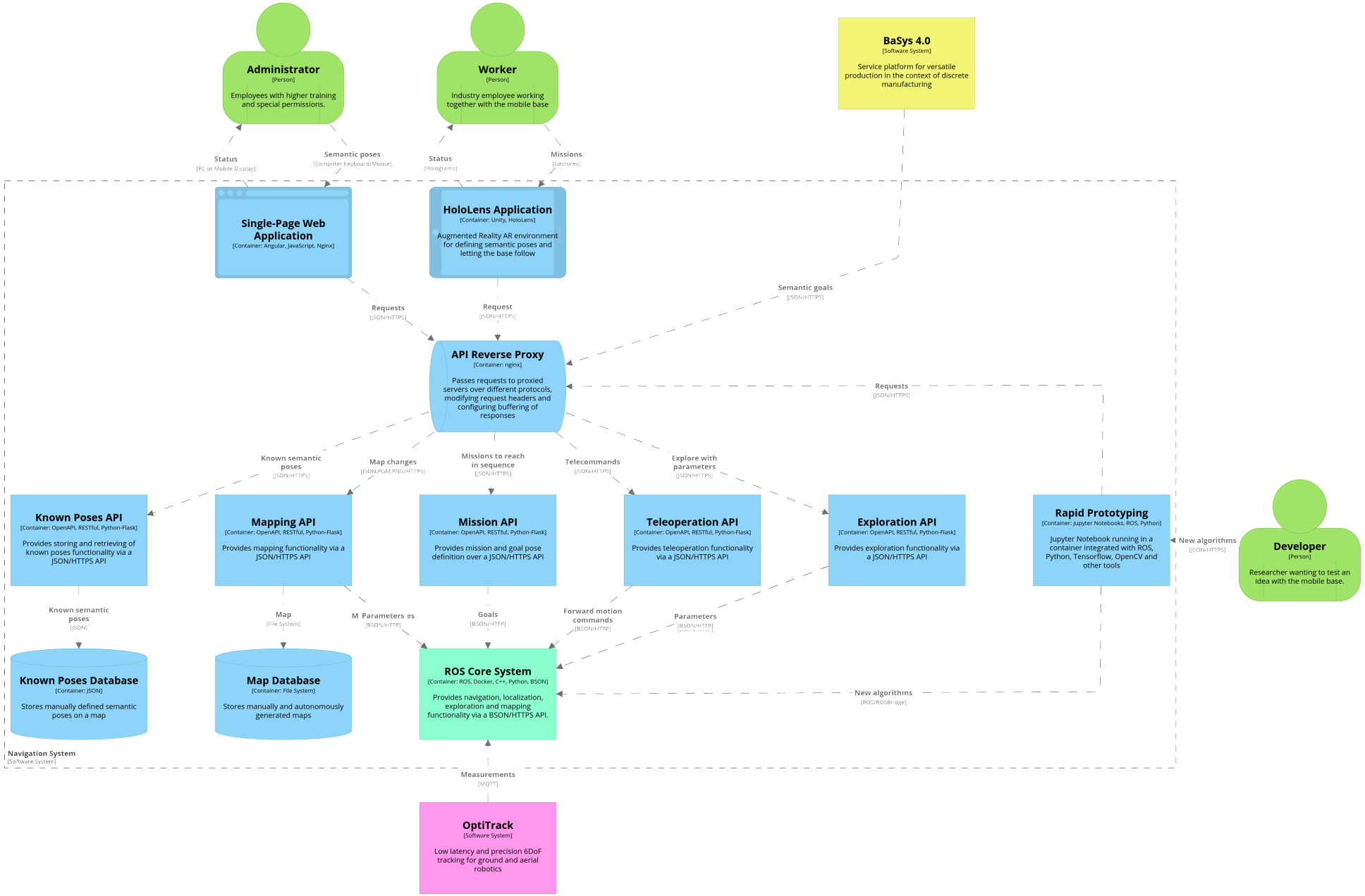


Fig. A.4.: Container View zooms into navigation system in scope showing the high-level technical building blocks

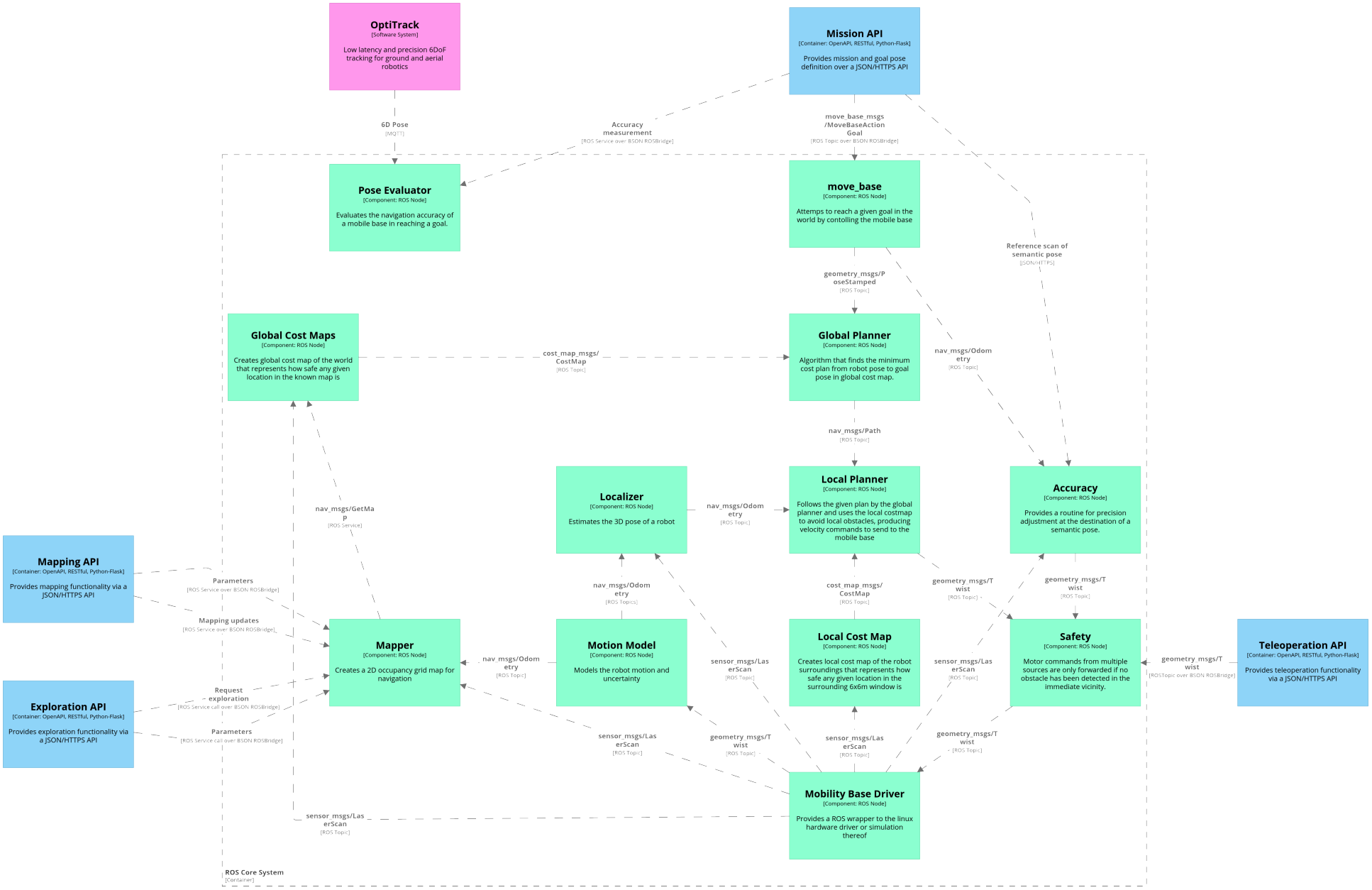


Fig. A.5.: Component diagram zooms into ROS Core showing the building components

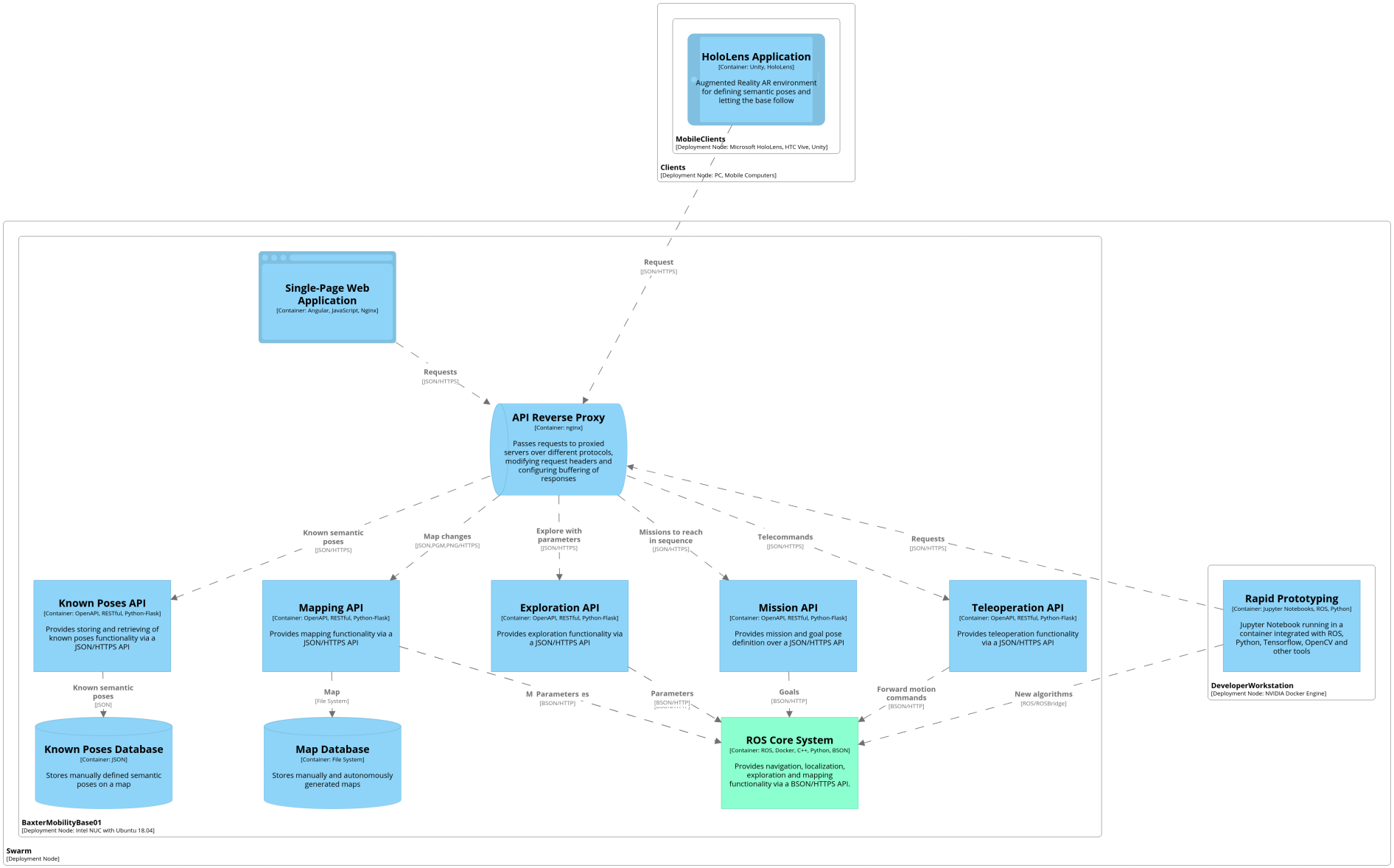


Fig. A.6.: Deployment diagram illustrating how containers in the static model are mapped to infrastructure

A.5 More Web Dashboard Screenshots

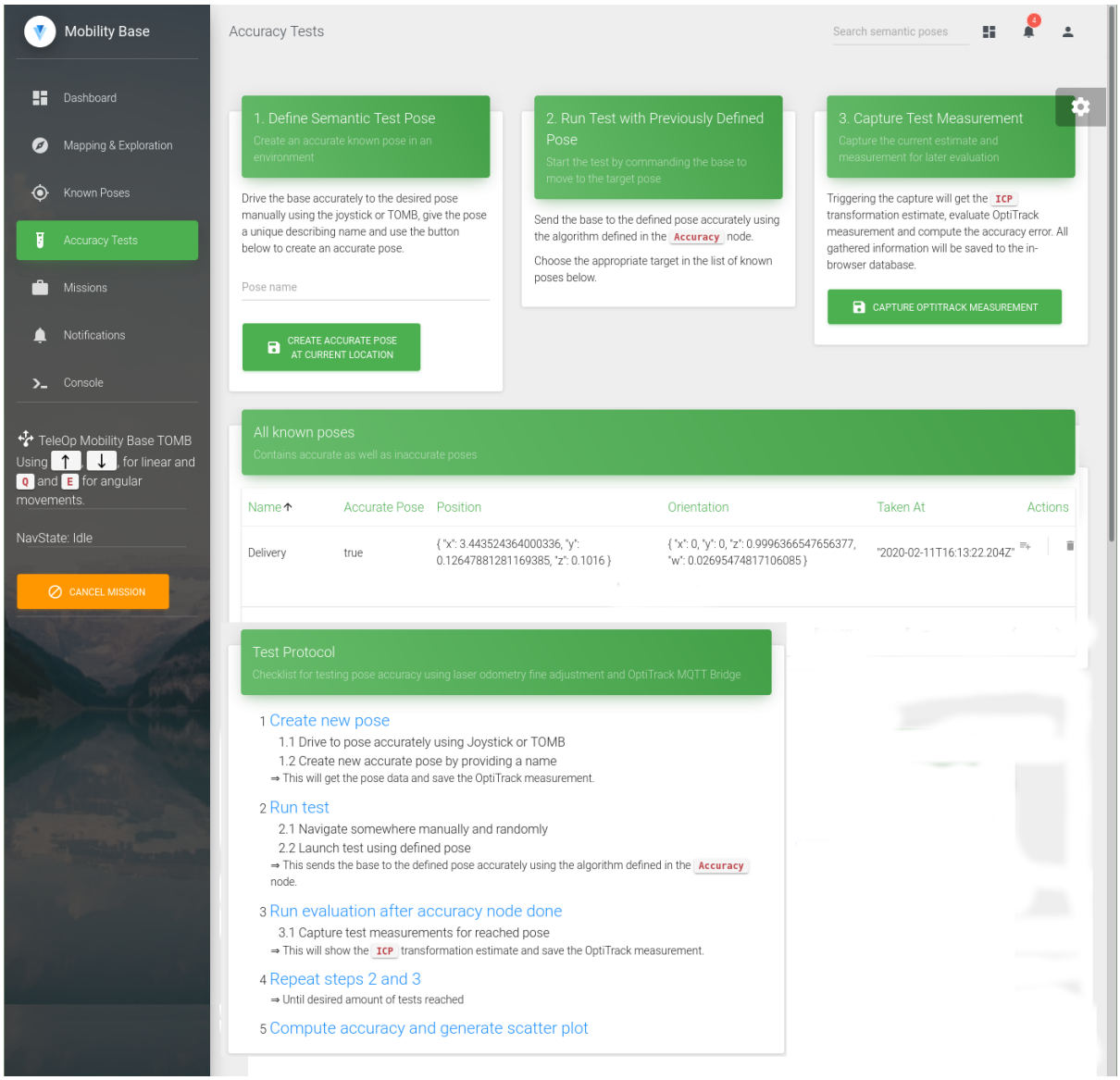


Fig. A.7.: Starting an accuracy measurement

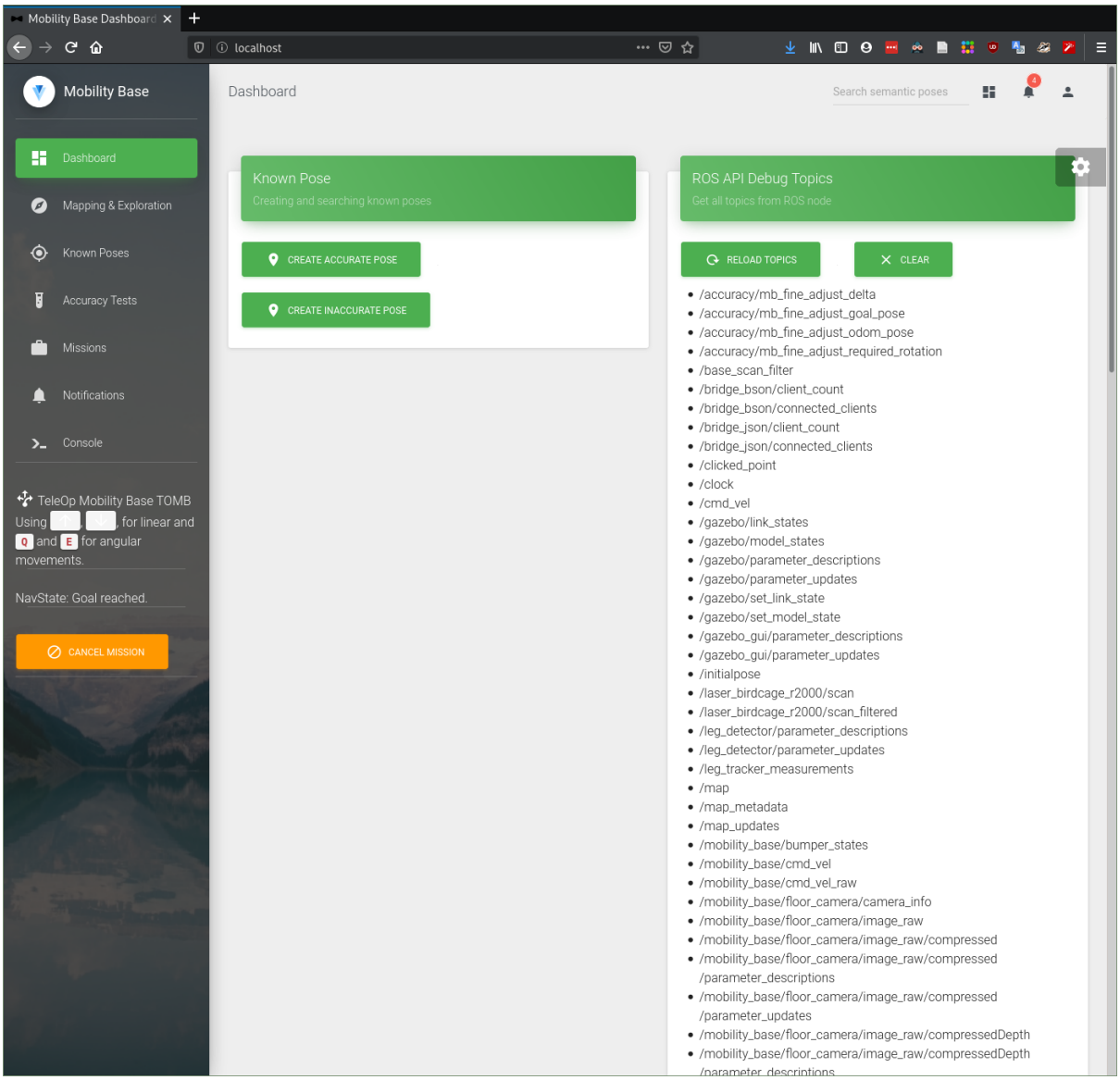


Fig. A.8.: Dashboard ROS Topics

A.5.1 Accuracy Tests UI

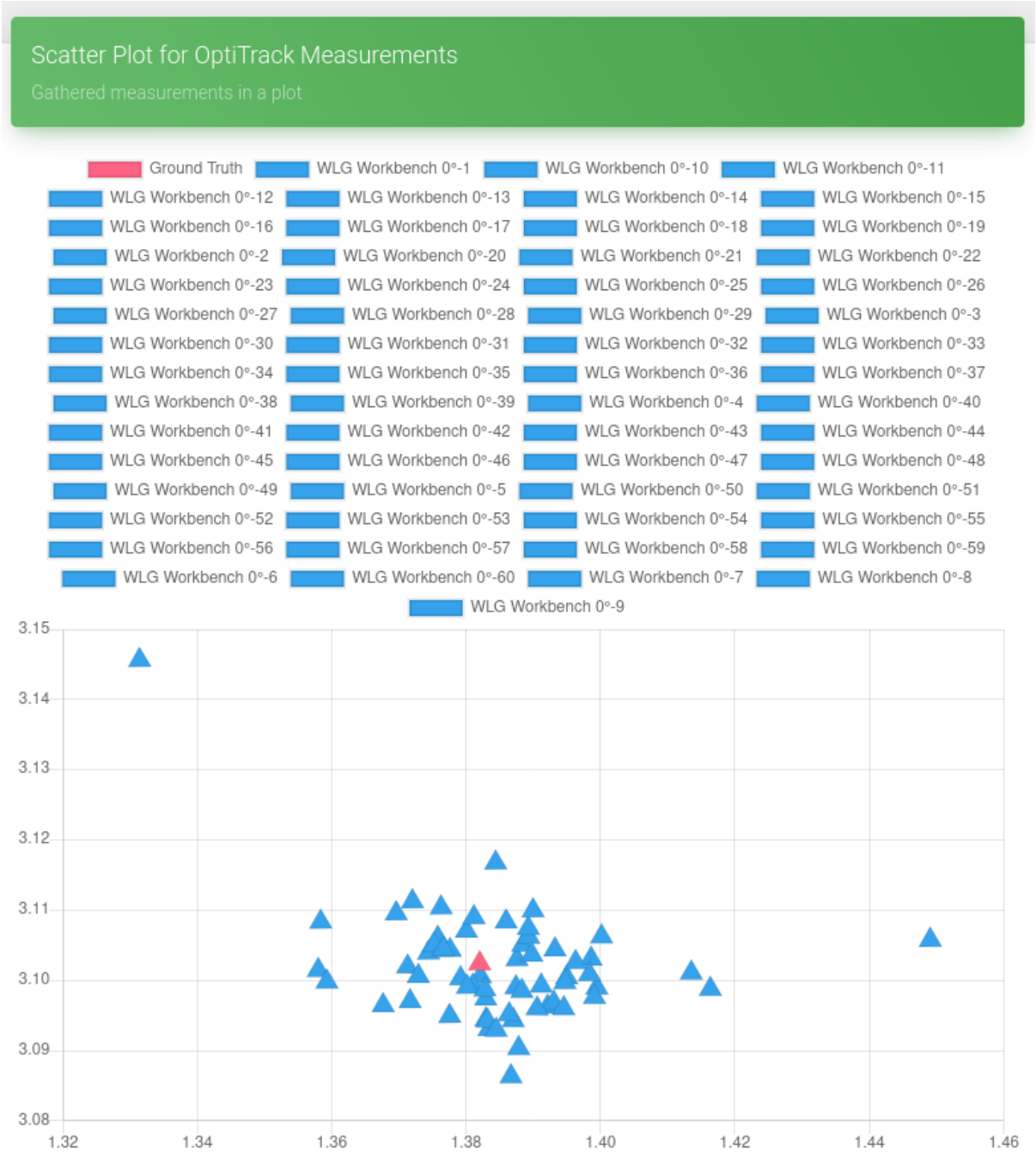
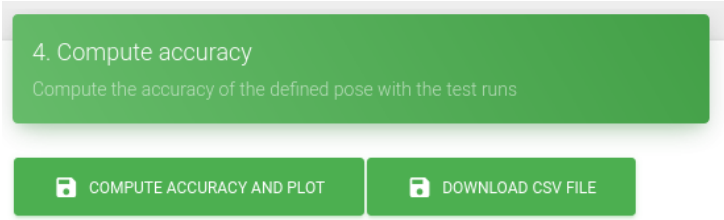


Fig. A.9.: Scatter Plot for OptiTrack measurements



Measurement Data

Measurement Name	Position Error (cm)	Rotation Error (deg)
WLG Workbench 0°-1	0.58	0.58
WLG Workbench 0°-10	0.98	1.03
WLG Workbench 0°-11	0.93	0.27
WLG Workbench 0°-12	0.74	0.18
WLG Workbench 0°-13	0.38	0.61
WLG Workbench 0°-14	1.07	0.53
WLG Workbench 0°-15	1.09	0.31
WLG Workbench 0°-16	1.17	0.52
WLG Workbench 0°-17	0.71	0.15
WLG Workbench 0°-18	2.44	0.37
WLG Workbench 0°-19	0.38	0.77
WLG Workbench 0°-2	6.65	0.95
WLG Workbench 0°-20	2.29	0.13
WLG Workbench 0°-21	6.71	0.69
WLG Workbench 0°-22	3.45	0.27
WLG Workbench 0°-23	1.43	0.28
WLG Workbench 0°-24	1.33	0.45
WLG Workbench 0°-25	0.79	0.59
WLG Workbench 0°-26	1.77	0.5
WLG Workbench 0°-27	0.88	0.28
WLG Workbench 0°-28	0.82	0.29
WLG Workbench 0°-29	0.69	0.02
WLG Workbench 0°-3	0.34	0.71
WLG Workbench 0°-30	3.15	0.03
WLG Workbench 0°-31	1.43	0.16
WLG Workbench 0°-32	1.3	0.35
WLG Workbench 0°-33	1.41	0.56
WLG Workbench 0°-34	1.07	0.61
WLG Workbench 0°-35	1.85	0.28
WLG Workbench 0°-36	1.31	0.05
WLG Workbench 0°-37	0.98	0.01
WLG Workbench 0°-38	0.82	0.72
WLG Workbench 0°-39	0.64	0.41
WLG Workbench 0°-4	0.66	0.52
WLG Workbench 0°-40	1.24	0.05
WLG Workbench 0°-41	0.97	0.12
WLG Workbench 0°-42	0.5	0.12

Fig. A.10.: Computed position and rotation error

A.6 Tutorials

A.6.1 LifeLong Mapping Using slam_toolbox

Extensive 2D SLAM library for creating and maintaining mobile robot navigation maps. Check out the project on GitHub for more information.

Introduction This project creates a Docker images to be used in any environment. It was originally built for the Mobility Base for Baxter but should be easily portable to other platforms using the provided image and compose files.

Clone the repository using git and build the image using

```
1 docker build -t ros-slam-toolbox .
```

Listing A.1: Build Docker image for slam_toolbox

Use It in Your Compose To use this node in conjunction with the rest of the environment add it to the same network. Define the dependencies of the node, which usually should be some odometry source. Further, to be able to (de-)serialize maps create and add a volume to the node. Also give some names and set the ROS environment variables. Finally, run the lifelong mapping node using the shell script provided in the container or create your own.

```
1  ros-slam-toolbox:
2          image: ros-slam-toolbox:latest
3          depends_on:
4                  - ros-odom
5          container_name: ros-slam-toolbox
6          hostname: ros-slam-toolbox
7          networks:
8                  - ros-overnet
9          environment:
10                 - ROS_HOSTNAME=ros-slam-toolbox
11                 - ROS_MASTER_URI=http://ros-master:11311
12         volumes:
13                 - maps:/root/.ros/
14         command: /run-shells/lifelong.sh
```

Listing A.2: YAML config for running the image

Serializing a Map Serializing a map to the file system can be done by simply calling the ROS service `/slam_toolbox/serialize_map` with a file name as argument.
NOTE: Set a known pose as a dock where SLAM starts

```
rosservice call /slam_toolbox/serialize_map map_name
```

Listing A.3: Call serialization service with map name

Deserializing a Map Deserialize a map from the file system by calling the ROS service `/slam_toolbox/deserialize_map` with the following `YAML` definition.

```
{
        filename: map_name,
        match_type: 2,
        initial_pose:
        {
                x: 0.0,
                y: 0.0,
                theta: 0.0
        }
}
```

Listing A.4: YAML config for deserializing a specified map

Call map deserialization using a approximate starting location defined as dock using the match / process type `PROCESS_NEAR_REGION` from the enumeration type `ProcessType`.

```
  enum ProcessType
  {
    PROCESS = 0,
    PROCESS_FIRST_NODE = 1,
    PROCESS_NEAR_REGION = 2,
    PROCESS_LOCALIZATION = 3
  };
```

Listing A.5: ENUM of ProcessType

This will localize the mobile base near the initial pose provided and apply scan matching using the actual laser scans. The mapping will continue from this pose in the graph.
NOTE: Fill in a previously saved known pose as a dock for SLAM starts

```
$ rosservice call /slam_toolbox/deserialize_map \
    "{filename: map_name, match_type: 2, \
    initial_pose: {x: 0.0, y: 0.0, theta: 0.0}}"
```

Listing A.6: Call deserialization service with yaml config

Tuning move_base

```
GlobalPlanner:
  old_navfn_behavior: false
  use_quadratic: true
  use_dijkstra: true
  use_grid_path: false

  allow_unknown: true

  planner_window_x: 0.0
  planner_window_y: 0.0
  default_tolerance: 0.0

  publish_scale: 100

  lethal_cost: 253
  neutral_cost: 66
  cost_factor: 0.55
  publish_potential: true
```

Listing A.7: GlobalPlanner configuration

```
DWAPlannerROS:
# Robot Configuration Parameters
  max_vel_x: 1.0
  min_vel_x: -0.5

  max_vel_y: 0.0
  min_vel_y: 0.0

# The velocity when robot is moving in a straight line
  max_trans_vel: 1.5
  min_trans_vel: 0.11

  max_rot_vel: 2.75
  min_rot_vel: 1.37

  acc_lim_x: 1.75
  acc_lim_y: 0.0
  acc_lim_theta: 3.2

# Goal Tolerance Parametes
```

```
  xy_goal_tolerance: 0.05
  yaw_goal_tolerance: 0.1
  latch_xy_goal_tolerance: false

# Forward Simulation Parameters
  sim_time: 1.5
  vx_samples: 20
  vy_samples: 0
  vth_samples: 40
  controller_frequency: 10.0

# Trajectory Scoring Parameters
  path_distance_bias: 32.0
  goal_distance_bias: 20.0
  occdist_scale: 0.02
  forward_point_distance: 0.325
  stop_time_buffer: 0.2
  scaling_speed: 0.25
  max_scaling_factor: 0.2

# Oscillation Prevention Parameters
  oscillation_reset_dist: 0.05

# Debugging
  publish_traj_pc : true
  publish_cost_grid_pc: true
```

Listing A.8: Dynamic Window Approach DWA local planner configuration

```
TebLocalPlannerROS:
  odom_topic: odom
  map_frame: map

  teb_autosize: True
  dt_ref: 0.3
  dt_hysteresis: 0.1
  min_samples: 3
  global_plan_overwrite_orientation: True
  global_plan_viapoint_sep: 0.5
  max_global_plan_lookahead_dist: 3.0
  force_reinit_new_goal_dist: 1.0
  feasibility_check_no_poses: 5
  publish_feedback: false
```

```
  shrink_horizon_backup: true
  allow_init_with_backwards_motion: true
  exact_arc_length: false
  shrink_horizon_min_duration: 10

  max_vel_x: 1.2
  max_vel_x_backwards: 1.0
  max_vel_theta: 1.0
  max_vel_y: 0 # not used, is differential
  acc_lim_y: 0 # not used, is differential
  acc_lim_x: 0.2
  acc_lim_theta: 0.2
  min_turning_radius: 0.0
  wheelbase: 0.0 # not used, is differential
  cmd_angle_instead_rotvel: false # not used, is differential
  footprint_model:
    type: "polygon"
    vertices: [[0.5, 0.5],[-0.5, 0.5],[-0.5, -0.5],[0.5, -0.5]]

  xy_goal_tolerance: 0.05
  yaw_goal_tolerance: 0.1
  free_goal_vel: False #False

  min_obstacle_dist: 0.435
  include_costmap_obstacles: True
  costmap_obstacles_behind_robot_dist: 1.0
  obstacle_poses_affected: 30
  inflation_dist: 0.7
  legacy_obstacle_association: false
  obstacle_association_force_inclusion_factor: 1.5
  obstacle_association_cutoff_factor: 5.0
  costmap_converter_plugin: "costmap_converter::
     CostmapToPolygonsDBSMCCH"
  costmap_converter_spin_thread: True
  costmap_converter_rate: 5

  no_inner_iterations: 5
  no_outer_iterations: 4
  optimization_activate: True # optimize
  optimization_verbose: False
  penalty_epsilon: 0.1
  weight_max_vel_x: 1
```

```
  weight_max_vel_y: 0
  weight_max_vel_theta: 1
  weight_acc_lim_x: 1
  weight_acc_lim_y: 0
  weight_acc_lim_theta: 1
  weight_kinematics_nh: 10000 # is a nonholonomic robot
  weight_kinematics_forward_drive: 10
  weight_kinematics_turning_radius: 1
  weight_optimaltime: 10
  weight_obstacle: 50
  weight_inflation: 1
  weight_viapoint: 1
  weight_adapt_factor: 2

  enable_homotopy_class_planning: False
  simple_exploration: False
  enable_multithreading: True
  max_number_classes: 4
  selection_cost_hysteresis: 1.0
  selection_obst_cost_scale: 4.0
  selection_viapoint_cost_scale: 1.0
  selection_alternative_time_cost: False
  roadmap_graph_no_samples: 15
  roadmap_graph_area_width: 6
  h_signature_prescaler: 1.0
  h_signature_threshold: 0.1
  obstacle_heading_threshold: 0.45
  roadmap_graph_no_samples: 15
  roadmap_graph_area_width: 5
  obstacle_keypoint_offset: 0.1
  visualize_hc_graph: False
```

Listing A.9: Timed Elastic Band TEB local planner configuration

```
global_frame: /map
robot_base_frame: /base_footprint

obstacle_range: 8.0
raytrace_range: 8.0
footprint: [[0.5, 0.5], [-0.5, 0.5], [-0.5, -0.5], [0.5, -0.5]]

update_frequency: 5.0
publish_frequency: 1.0
```

```
transform_tolerance: 10.0

resolution: 0.05

static_layer:
  map_topic: /map
  #  subscribe_to_updates: true
  unknown_cost_value: 99
  #  track_unknown_space: true
  lethal_cost_threshold: 150
  trinary_costmap: true

#robot_radius = ir_of_robot

obstacle_layer:
  observation_sources: laser_scan_sensor
  unknown_threshold: 15
  mark_threshold: 0
  combination_method: 1
  track_unknown_space: false    # true needed for disabling
     global path planning through unknown space
  obstacle_range: 4.0
  raytrace_range: 5.0
  laser_scan_sensor: {
    sensor_frame: laser_birdcage_r2000,
    data_type: LaserScan,
    topic: laser_birdcage_r2000/scan_filtered,
    marking: true,
    clearing: true
  }

inflation_layer:
  cost_scaling_factor: 2.58 #1 #2.58
  inflation_radius: 1.75

proxemic_layer:
  amplitude: 150.0
  covariance: 0.3
  factor: 7.0
passing_layer:
  enabled: false
```

Listing A.10: Common Costmap Parameters

```yaml
global_costmap:
  global_frame: /map
  robot_base_frame: /base_footprint
  update_frequency: 2.0
  publish_frequency: 1.0
  map_type: costmap
  static_map: false
  rolling_window: true
  width: 70.0
  height: 70.0
  always_send_full_costmap: true

  plugins:
  - {name: static_layer, type: "costmap_2d::StaticLayer"}
  - {name: obstacle_layer, type: "costmap_2d::ObstacleLayer"}
  - {name: inflation_layer, type: "costmap_2d::InflationLayer"}
  - {name: proxemic_layer, type: "social_navigation_layers::
     ProxemicLayer"}
  - {name: passing_layer, type: "social_navigation_layers::
     PassingLayer"}
```

Listing A.11: Global Costmap

```yaml
local_costmap:
  global_frame: /odom
  robot_base_frame: /base_footprint
  update_frequency: 5.0
  publish_frequency: 2.0
  static_map: false
  rolling_window: true
  width: 6.0
  height: 6.0
  resolution: 0.05

  plugins:
  - {name: obstacle_layer, type: "costmap_2d::ObstacleLayer"}
  - {name: inflation_layer, type: "costmap_2d::InflationLayer"}
  - {name: proxemic_layer, type: "social_navigation_layers::
     ProxemicLayer"}
  - {name: passing_layer, type: "social_navigation_layers::
     PassingLayer"}
```

Listing A.12: Local Costmap

A.7 Infrastructure Resources

Contributed Source Code Resources

The code is managed with Gitlab (https://gitlab.enterpriselab.ch) of the Enterpriselab. All components are categorized in different GitLab Groups and subcomponents are stored in different GitLab Projects.

Project	URL
ROS Indigo	Subgroup for ROS Indigo based projects (these are old projects, as I moved to ROS Kinetic). Contains 16 Git projects from containerized drivers and simulation environments to different SLAM approaches and custom built ROS nodes. The whole navigation stack in ROS Indigo with various configurations. https://gitlab.enterpriselab.ch/mt-kawa/ros-indigo-mb-docker
ROS Kinetic	Subgroup for ROS Kinetic based projects. Contains 17 Git projects from containerized drivers and simulation environments to different SLAM approaches and custom built ROS nodes. The whole navigation stack in ROS Kinetic with various configurations. https://gitlab.enterpriselab.ch/mt-kawa/ros-kinetic-mb-docker
Message Definitions	Subgroup for ROS message definitions. Contains 2 Git project defining message definitions that are used with the custom built ROS nodes such as Accuracy and a Known Pose Converter. https://gitlab.enterpriselab.ch/mt-kawa/ros-kinetic-mb-docker
Web Frontends	Subgroup for web frontends containing 2 Git projects for different interfaces. https://gitlab.enterpriselab.ch/mt-kawa/web-frontends
Proxies	Subgroup for proxy configuration containing 1 Git project. https://gitlab.enterpriselab.ch/mt-kawa/proxies
Templates	Subgroup containing 2 templates for ROS Kinetic and Indigo integrated with Python and C++. https://gitlab.enterpriselab.ch/mt-kawa/ros-templates
Rapid Prototyping	Subgroup for enabling rapid prototyping using 4 different Jupyter Notebook integrations. https://gitlab.enterpriselab.ch/mt-kawa/rapid-prototyping
Tools	Subgroup for simulation tools such as a containerized blender OBJ to gazebo DAE file converter. This is handy to convert models and worlds designed with a more appropriate tool such as Blender and import them into the Gazebo Simulator. https://gitlab.enterpriselab.ch/mt-kawa/simulation-environment

Project	URL

Tab. A.1.: Source code resources

Artifacts

The artifacts for the thesis as Python Package Index PyPI or Node Package Manager NPM packages are stored in the corresponding public artifact repositories. All artifacts are BSD licensed, which make them easier to reuse, especially in the development environment and Continuous Integration / Continuous Deployment CI/CD chain.

Project	URL
Known Pose Clients	Generated Python client publicly available on Python Package Index PyPI https://pypi.org/project/known-pose-client Generated JavaScript client publicly available on Node Package Manager NPM https://www.npmjs.com/package/known-pose-client Other clients for java, csharp on GitLab https://gitlab.enterpriselab.ch/mt-kawa/api-clients Server stubs for the Known Pose API on GitLab https://gitlab.enterpriselab.ch/mt-kawa/api-servers

Tab. A.2.: Artifact resources

Docker Images

Project	URL
Docker Registry	51 Docker images hosted in the Docker Registry of the Enterprise Lab available at repohub.enterpriselab.ch:5002/kawa. There are 7 out of those 51 that are variation of the original image used to differentiate environments and configurations. List of all Docker images

Tab. A.3.: Docker images

API Documentation

REST APIs are documented using the OpenAPI 3 specification. The documentation is hosted publicly on SwaggerHub https://app.swaggerhub.com/apis/kw90/known-pose-api/1.1.0. The Documentation shows the available operations of the API as well as the defined schema. This documentation allows to generate client SDKs and server stubs for nearly all thinkable languages.

Thesis Documentation

Documentations like this thesis and various presentations can be found in the `mt-docs` group at https://gitlab.enterpriselab.ch/mt-kawa/mt-docs.

`Project`	URL	Technologies
`Thesis`	Project for this thesis. Project adds filters for PlantUML diagrams, pandoc (for markdown) and watchexec to rebuild latex when changes are saved to file. This thesis build on the CleanThesis theme.	CleanThesis 0.4.0 Pandoc 2.7.2 TexLive 2019 Docker 19.03.5
`Intermediate Presentation`	Project for making presentations fastly using markdown then using Pandoc to convert it to a nice LaTeX Beamer presentation. Supports PlantUML, GanttCharts, revealjs, and other plugins. Can also be used to create LaTeX documents using simple markdown syntax.	Pandoc 2.7.2 TexLive 2019 Docker 19.03.5

Tab. A.4.: Documentation resources

Container Runtime

The configurations for the desired deployment tool, which for this thesis was simply `Docker-Compose` are defined for multiple scenarios. These configurations become complex rather quickly for multipart applications. For simulation environments, a script is provided to forward the display environment inside virtual containers. Different platforms require different runtime configurations. Therefore, different repositories were created in this thesis for simulation and real-world scenarios.

Project	URL	Technologies
MB Robot Runtime	Project hosting all docker-compose configurations for navigation and exploration using different algorithms, lifelong mapping and forwarding the robots desktop using a browser based VNC solution. This is for the real robot. https://gitlab.enterpriselab.ch/mt-kawa/ros-kinetic-compose/ros-kinetic-mb-sim-compose	Docker-Compose 1.25.0 Docker 19.03.5
Simulation Runtime	Project hosting all docker-compose configurations for navigation and exploration using different algorithms, lifelong mapping and forwarding the host display inside a container environment. This is for simulating a robot on a powerful machine. https://gitlab.enterpriselab.ch/mt-kawa/ros-kinetic-compose/ros-kinetic-robot-compose	Docker-Compose 1.25.0 Docker 19.03.5
ROS Indigo	GitLab Group containing 3 Runtimes for ROS Indigo (Deprecated). https://gitlab.enterpriselab.ch/mt-kawa/ros-indigo-compose	Docker-Compose 1.25.0 Docker 19.03.5

Tab. A.5.: Container runtime resources

DevOps

Updating the API Specification Updating the Spec and with it the Client SDKs as well as Server Stubs works as follows

1. Edit the yaml at SwaggerHub for convenience and validation and save the changes
2. Export or copy the yaml and replace the openapi.yaml in this project
3. Commit and push the changes to the repository
4. Magic
5. Find the client SDKs and server stubs over at the api-clients/mrk40-known-poses-clients project
6. Merge Branch `SwaggerHub` with the `master` branch to pull the clients built over there

The magic is actually the `swagger-codegen-cli`[1] that generates the defined client SDKs defined in the CI/CD chain of that specific project. The official image from OpenAPI doesn't allow easy calling in a CI environment like Gitlab CI. The image contributed in the GitLab project `swagger-codegen-cli` here allows exactly that. To do that it gets the `swagger-codegen-cli.jar` maven artifact from Swagger and mounts volumes from the container to the host.

A.8 Additional Evaluation Plots

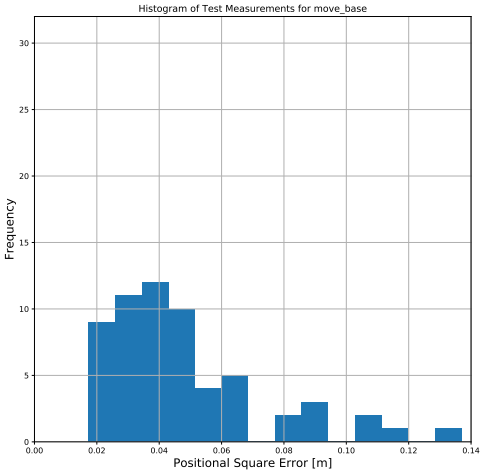


Fig. A.11.: Histogram position error move_base

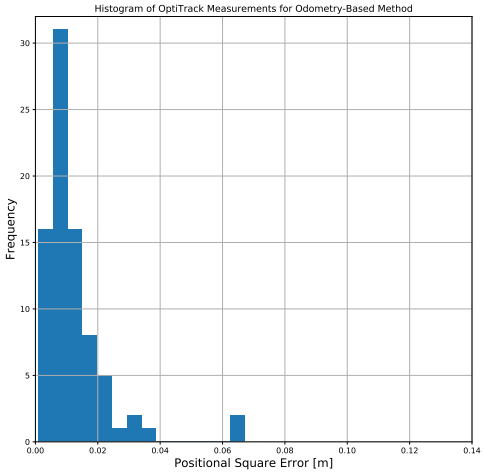


Fig. A.13.: Histogram position error odometry-based method

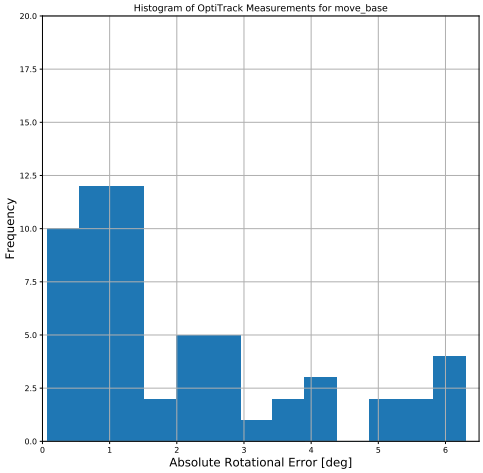


Fig. A.12.: Histogram rotation error move_base

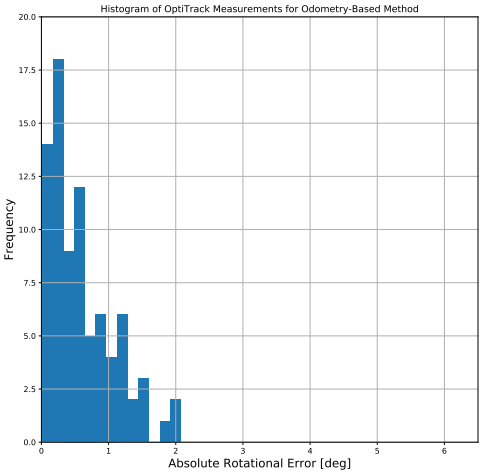


Fig. A.14.: Histogram rotation error odometry-based method

[1]Found at https://gitlab.enterpriselab.ch/mt-kawa/api-docs/swagger-codegen-cli

QQ and PP Plots

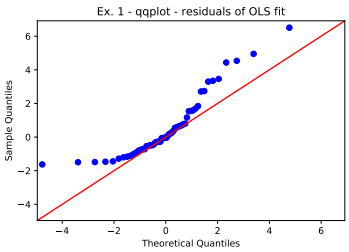


Fig. A.15.: Normal Q-Q plot comparing position errors from move_base on the vertical axis to theoretical quantiles

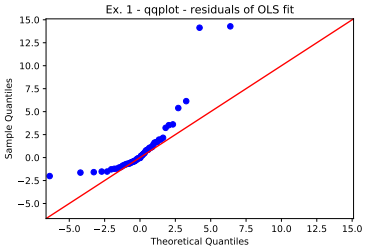


Fig. A.16.: Normal Q-Q plot comparing position errors from odometry-based method on the vertical axis to theoretical quantiles

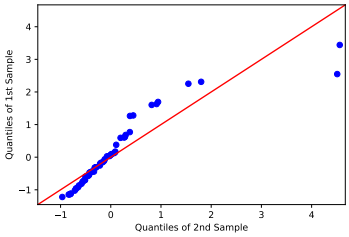


Fig. A.17.: Q-Q plot comparing position errors from move_base method on the vertical axis to odometry-based method

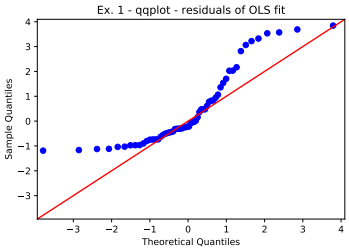


Fig. A.18.: Normal Q-Q plot comparing rotation errors from move_base on the vertical axis to theoretical quantiles

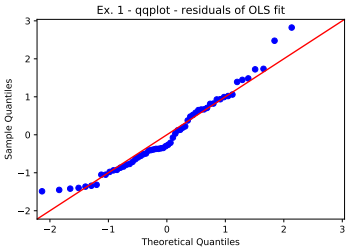


Fig. A.19.: Normal Q-Q plot comparing rotation errors from odometry-based method on the vertical axis to theoretical quantiles

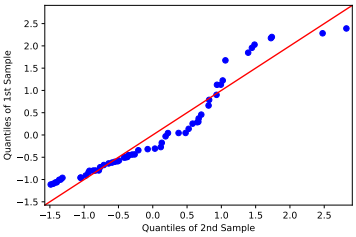


Fig. A.20.: Q-Q plot comparing rotation errors from move_base method on the vertical axis to odometry-based method

A.9 Pipelines

- Total: 58 pipelines
- Successful: 48 pipelines
- Failed: 10 pipelines
- Success ratio: 82

Navigation Stack

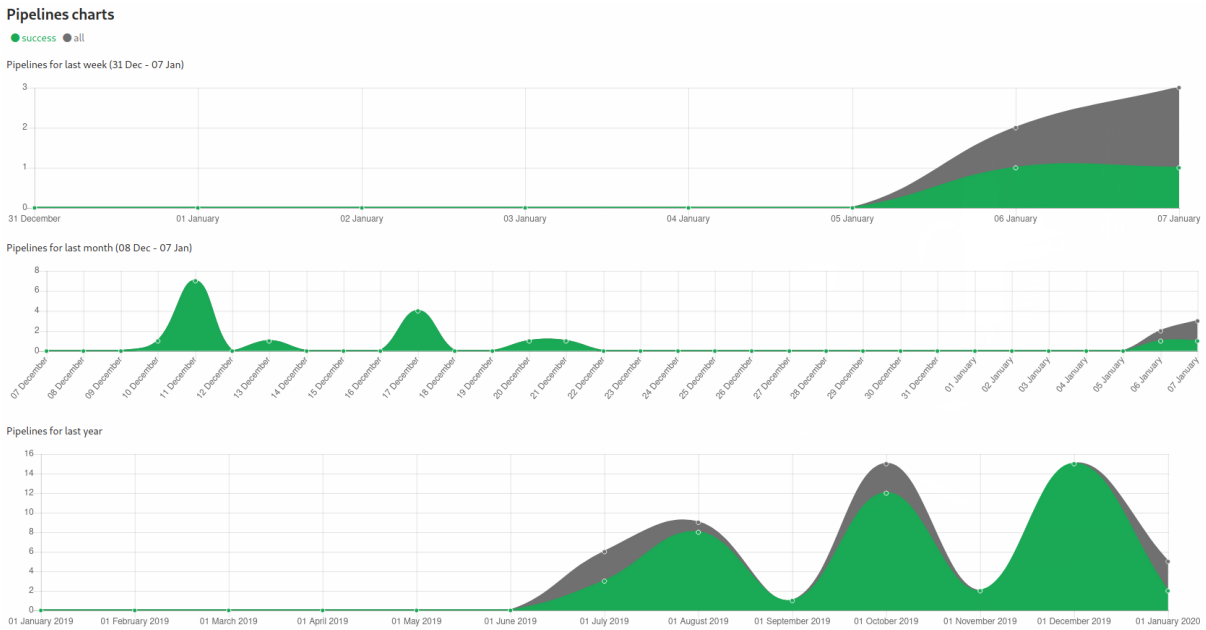


Fig. A.21.: Navigation stack pipeline

Colophon

This thesis was typeset with LaTeX 2_ε. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at http://cleanthesis.der-ric.de/.

Declaration

I hereby certify that this thesis has been composed by me and is based on my own work, unless stated otherwise. No other persons work has been used without due acknowledgement in this thesis. All references and verbatim extracts have been quoted, and all sources of information, including graphs and data sets, have been specifically acknowledged.

Rotkreuz, February 16, 2020

Kai Waelti