BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF MECHANICAL ENGINEERING
FAKULTA STROJNÍHO INŽENÝRSTVÍ

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS
ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

ROS FRAMEWORK UTILIZATION FOR AUTONOMOUS MOBILE ROBOT CONTROL SYSTEM
VYUŽITÍ NÁSTROJE ROS PRO ŘÍZENÍ AUTONOMNÍHO MOBILNÍHO ROBOTU

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR — **Bc. Patrik Vávra**
AUTOR PRÁCE

SUPERVISOR — **Ing. Martin Appel**
VEDOUCÍ PRÁCE

BRNO 2019

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF MECHANICAL ENGINEERING

Specification Master's Thesis

Department:	Institute of Solid Mechanics, Mechatronics and Biomechanics
Student:	**Bc. Patrik Vávra**
Study programme:	Applied Sciences in Engineering
Study field:	Mechatronics
Supervisor:	**Ing. Martin Appel**
Academic year:	2018/19

Pursuant to Act no. 111/1998 concerning universities and the BUT study and examination rules, you have been assigned the following topic by the institute director Master's Thesis:

ROS framework utilization for autonomous mobile robot control system

Concise characteristic of the task:

This thesis deals with four–wheel robot control system using ROS (Robot Operating System) framework, which allows robot to perform more advanced tasks such as autonomous motion, obstacle avoidance and mapping.

The aim of this thesis is to get acquainted with the ROS framework and design a topology that will include a few robots and control panels. Mobile robot will be capable of autonomous motion (automatic return to docking station) as well as remote control by user.

The outcome of this thesis will be functional ROS based robot control system, which will be implemented on real four–wheel robot. Robot design description is included in another master's thesis.

Goals Master's Thesis:

1) Perform a search study of ROS framework and familiarize yourself with ROS tools and conventions.
2) Describe ROS navigation and localization packages and their modifications used in robot's software.
3) Create simulation environment in Gazebo and test selected localization and navigation method.
4) Implement navigation into real robot in form of automatic return to the docking base.
5) Create educational or entertaining mini–games for users of robots.

Recommended bibliography:

THRUN, Sebastian, Wolfram BURGARD a Dieter FOX. Probabilistic Robotics. [1st ed.]. Cambridge: The Mit Press, 2006. ISBN 978-0-262-20162-9.

CHOSET, Howie M. Principles of robot motion: theory, algorithms, and implementation. Cambridge, Mass.: MIT Press, c2005. Intelligent robotics and autonomous agents.

Dokumentace systému ROS, dostupné online na adrese „wiki.ros.org“

Deadline for submission Master's Thesis is given by the Schedule of the Academic year 2018/19

In Brno,

L. S.

prof. Ing. Jindřich Petruška, CSc. Director of the Institute	doc. Ing. Jaroslav Katolický, Ph.D. FME dean

Abstrakt

Tato práce se zabývá vytvořením lokalizačního a navigačního systému mobilního robota pro vnitřní prostředí pomocí frameworku ROS. Stručně je zde představen projekt, v rámci kterého diplomová práce vznikla, a jeho cíle. V rešeršní části je v krátkosti popsán ROS framework, simulační prostředí Gazebo a senzory, kterými robot disponuje. Následuje vytvoření modelu robota a simulačního prostředí, v němž jsou vyzkoušeny lokalizační, navigační a další rutiny. V experimentální části je provedeno testování senzorů a popsáno využití jejich výstupů. Následně jsou upraveny a otestovány algoritmy ze simulace na reálném robotovi. V závěru jsou popsány vytvořené vzdělávací minihry. Hlavním výstupem této práce je funkční stavový automat, který umožňuje manuální ovládání, zadávání cílů pro navigaci a v případě potřeby zajistí autonomní nabití robota.

Summary

This thesis deals with the development of localization and navigation system of a mobile robot for an indoor environment based on ROS framework. The project, ROS framework, and Gazebo simulation environment are briefly described in the theoretical survey section alongside sensors that the robot is equipped with. This is followed by the creation of a model of the robot and a simulation environment in which localization, navigation, and other routines are tested. In the experimental section, the sensors are tested, and the usage of their output is described. Subsequently, algorithms from the simulation are modified and tested on the real robot. In the end section are created educational mini-games described. The main outcome of this thesis is a functional state machine that allows to manually control the robot, give the goal position for navigation and if needed, takes care of autonomous charging of robot.

Klíčová slova

ROS, Gazebo, lokalizace, navigace, SMACH, autonomní mobilní robot, ArUco, SLAM, simulační testování, mobilní robotika

Keywords

ROS, Gazebo, localization, navigation, SMACH, autonomous mobile robot, ArUco, SLAM, simulation testing, mobile robotics

Bibliographic citation

VÁVRA, P. *ROS framework utilization for autonomous mobile robot control system.* Brno: Brno University of Technology, Faculty of mechanical engineering, 2019. 82 pages, Master's thesis supervisor: Ing. Martin Appel.

Rozšířený abstrakt

Úvod

Rok 2020 bude značit přesně sto let od chvíle, kdy bylo poprvé použito slovo Robot ve hře R.U.R. (Rossumovi Univerzální Roboti). Od té doby se z fiktivních postav roboti stali běžnou součástí našich životů. V domě se automatické robotické vysavače starají o čistotu podlah, na zahradě roboti automaticky sekají trávník a každý jistě přijde na mnoho dalších příkladů využití robotů. K nejzajímavějším částem mobilní robotiky bezpochyby patří autonomní doprava. Nynější auta nabízí celou řadu asistenčních systémů a ve vývoji jsou plně nebo částečně autonomní vozidla.

Aby mohlo být dosaženo vysoké úrovně autonomního řízení, musí mít vozidla velmi dobré informace o svém okolí. K tomu používají celou řadu senzorů, například Tesla model S obsahuje dle specifikací v manuálu [1] 12 ultrazvuků, 8 kamer a radar. Malý robot pro vnitřní autonomní jízdu, který bude představen v této práci, samozřejmě patří do úplně jiné kategorie, a proto si vystačí s daleko méně senzory.

Tato práce je součástí projektu, který v roce 2020 vyústí ve Výstavu robotiky v Technickém muzeum v Brně na počest sta let od prvního použití slova robot. Projekt je v současné době tvořen dvěma diplomovými pracemi - kromě této také prací Bc. Romana Adámka [2], na kterou bude v určitých částech odkazováno. Tato diplomová práce vytváří základy ovládacího systému pro částečně autonomního robota, který bude v rámci výstavy sloužit pro ukázku různých aspektů robotiky. Předpokladem při psaní práce je, že práce na projektu bude pokračovat, a proto je u většiny balíčků uveden odkaz na dokumentaci nebo doplňkovou literaturu. Kvůli stejnému předpokladu proběhl vývoj robota i v simulačním prostředí, jelikož v něm lze rychleji otestovat nové algoritmy a postupy.

Po domluvě s vedoucím práce byl software vyvíjen, a zde je tak i popsán, pouze pro jednoho robota, ovládací a nabíjecí stanici. Software může být v budoucnu rozšířen tak, aby zahrnoval několik robotů, ovládacích a nabíjecích stanic.

Popis řešení

Práce je rozdělena do 3 hlavních částí. V první je v krátkosti představen ROS framework a program Gazebo pro simulační řešení. Následuje popis vybraných balíčků z frameworku ROS. Pro lepší pochopení problematiky jsou zde uvedeny i konvence, které komunita využívající ROS dodržuje. Některé balíčky jsou pouze krátce charakterizovány a je u nich uveden odkaz na další literaturu, u jiných je vysvětleno jejich fungování a možnosti nastavení. Tento popis je zaměřen hlavně na lokalizační a navigační možnosti ve frameworku ROS. Pro vytvoření komplexního stavového automatu je zde představen i knihovna SMACH psaná v jazyce Python. V rešeršní části je také proveden výčet senzorů a jejich parametrů. Následně je u každého senzoru popsáno jakým způsobem bude v práci využit a tam, kde je to nutné, je uvedena nezbytná teorie pro pochopení dané problematiky.

Další část se zabývá simulačním řešením. Nejprve je nutno navrhnout simulační prostředí, které by zhruba mělo odpovídat zamýšlenému exponátu použitému v Technickém muzeu. Simulační prostředí je vytvořeno pomocí GUI integrovaného v Gazebu a dostupných modelů z databáze. Do prostředí je také přidáno 20 ArUco značek, aby se robot byl schopen loka-

lizovat. Model robotu je vytvořen pomocí jazyka Xacro. Parametry, které definují jednotlivé části jsou alespoň v krátkosti zmíněny a jsou zde uvedeny ukázky z Xacro kódu. STL soubory pro vizuální zobrazení jednotlivých částí je převzato z [2]. Pomocí dostupných pluginů lze i simulovat senzory, jimiž je robot vybaven.

Po úspěšném vytvoření simulačního prostředí a robota je tato část integrována s frameworkem ROS. Následně je zde uvedeno nastavení pro lokalizační balíček *robot_localization*. Z tohoto balíčku je využit *node* EKF pro fúzi relativních i absolutních dat ze senzorů. Pro každý *node* jsou zde uvedeny jaké vstupy integruje a jeho omezení. Pro SLAM balíček *gmapping* jsou zde uvedeny pouze jeho vstupy a je ukázána mapa prostředí, která byla pomocí balíčku vytvořena. Dále je popsáno základní nastavení pro jednotlivé komponenty navigačního balíčku *move_base_flex*. Ty jsou vzájemně propojeny pomocí stavového automatu pro navigaci, vytvořeného pomocí knihovny SMACH.

Na začátku experimentální části je vypsán použitý hardware se svými specifikacemi a je schématicky znázorněno, jaké části softwaru se vykonávají na daném hardwaru. Pro implementaci softwaru na reálného robota je nutno znát parametry jeho senzorů a vytvořit strukturu, která by data ze senzoru upravila pro následnou fúzi v lokalizačním balíčku. Aby mohly být správně odhadnuty parametry robota a otestována přesnost senzorů, tzv. *ground truth data* jsou získána pomocí kamery umístěné pod stropem a ChArUco desky umístěné na vrchní části robota. Porovnáním těchto hodnot z dat enkodérů lze estimovat parametry pro kinematický model uvedený v rešeršní části. Jelikož z něj nelze jednoduchým způsobem vytvořit inverzní model, jsou parametry odhadnuty zvlášť i pro inverzní model. Dále je provedena kalibrace IMU jednotky a porovnán odhad natočení kolem svislé osy s *ground truth daty*. V rozsáhlé části týkající se estimace polohy v globálním souřadném systému pomocí zpracování obrazu, je nejprve otestována přesnost určení polohy v lokálním souřadném systému. Poté je proveden návrh vyjádření této nepřesnosti a v dalších částech jsou data z lokálního s.s. transformovaná do globálního s.s, kde jsou sloučena, a takto upravená data lze použít jako vstup do lokalizačního balíčku.

Lokalizační balíček je otestován nejprve v SLAM algoritmu, pomocí kterého je vytvořena mapa 7. patra budovy A3 Fakulty strojního inženýrství a mapa části Mechatronické laboratoře. Do laboratoře jsou umístěny ArUco značky a jejich poloha je změřena a zapsána. Po provedení těchto částí lze přistoupit na návrh navigace do nabíjecí stanice. K tomu je také potřeba vytvořit složitější stavový automat a použít 2 plánovače a kontroléry. Navigace je úspěšně otestována a vyhodnocena. Robot může být ovládán manuálně, můžou mu být v uživatelském rozhraní zadávány příkazy pro navigaci a robot také soustavně monitoruje stav své baterie. Pokud hodnota napětí klesne pod definovanou hodnotu, robot se přepne do autonomního režimu, dojede do nabíjecí stanice a po nabití z ní zase odjede. Popsané chování je implementované v rámci stavového automatu vytvořeného v knihovně SMACH.

Jedním z cílů práce je taktéž vytvoření ukázkových miniher pro návštěvníky/uživatele. První z těchto úkolů se týká navedení robota k nabíjecí stanici. Návštěvníkovi se situace podá následovně: globální navigační systém robota má poruchu a dochází mu energie, je třeba jej dovést k nabíjecí stanici. Robot se simulovanou poruchou sleduje ArUco značku umístěnou na zadní části robota, který je ovládán uživatelem, uživatel řídí vedoucího robota tak, aby dosáhli požadované zóny v okolí nabíjecí stanice.

Druhá minihra představí návštěvníkům proces tvorby mapy. Na začátku úkolu se spustí software nutný k provedení SLAM algoritmu, uživatel manuálně jezdí s robotem vybaveným lidarem takovým způsobem, aby co nejlépe vytvořil mapu prostředí. Aktuální stav mapy je zobrazován na monitoru a po dokončení je provedeno porovnání s mapou, která byla vytvořena a následně upravena takovým způsobem, aby co nejvěrněji reprezentovala skutečné prostředí. Porovnání je poté ukázáno v separátním okně.

Shrnutí a zhodnocení výsledků

V rámci práce byly vytvořeny lokalizační a navigační systémy a jejich funkčnost otestována v simulačním i reálném prostředí. Hlavním výstupem této práce je funkční stavový automat, který umožňuje manuální ovládání, zadávání cílů pro navigaci a v případě potřeby zajistí autonomní nabití robota. Úspěšnost najetí robota do nabíjecí stanice je závislá na přesnosti změření pozice a natočení ArUco značek. Pokud jsou tyto značka změřeny s dostatečnou přesností, robot ve většině případů najede do nabíjecí stanice na první pokus. Vytvořené minihry nejsou zahrnuty ve stavovém automatu z časových důvodů. Jejich implementace do automatu je závislá (minihra navádění robota) na více robotech a v době psaní práce byl k dispozici pouze jeden robot. Nicméně podstata miniher je vytvořena a otestována i na reálném robotovi. Vytyčené cíle práce tedy byly splněny.

I affirm that the presented master's thesis is my genuine work and that it was created with the support of the stated literature, under the supervision of my tutor.

Patrik Vávra

Brno . . . .

I would like to thank my supervisor, Ing. Martin Appel, for his guidance and support during the development of this thesis. Also, I would like to express my gratitude to Bc. Roman Adámek for inspiring cooperation and to all those who supported me during the work on this thesis.

Patrik Vávra

Contents

1 Introduction

The year 2020 will mark one hundred years since the word robot was first used in play R.U.R. (Rossum's Universal Robots) by Karel Čapek. Since then, robots have evolved from fictional characters to become an everyday part of our lives. To address specifically mobile robots - robotic vacuum cleaner cleans a floor without human supervision; automatic mower takes care of the lawn in the garden, and several robotic solutions exist for collecting crops from fields and trees to give a few examples.

Probably the most interesting usage of mobile robotics lies in the autonomous transport of people as well as goods. In modern vehicles, numerous Advanced driver-assistance systems are used to create a more comfortable and safer drive for drivers and passengers. These systems are the first step for truly autonomous vehicles which are in development. According to the classification of autonomous cars by the Society of Automotive Engineers (SAE), six degrees of automated driving exist [3]. As for the time of the writing, cars on a maximum level of 2 are produced, except for Audi's AI traffic jam pilot system that reached level 3. For the highest two levels, vehicle system takes all responsibility and controls without the need of any action from the driver [3].

To be able to reach that high level of autonomous behavior, the necessity is to have reliable knowledge of surroundings. For that, numerous diverse sensors are used. For example, Tesla S has according to [1] 12 ultrasonic sensors, 8 cameras and radar for perception. The Small indoor robot presented in this thesis is, of course, completely different category and manages to function properly with a fraction of necessary sensors.

This thesis forms a part of the project that will result in the Exhibition of Robotics in the Brno Technical Museum as an honor to 100 years since the invention of the word robot. The visitors will have the opportunity to see various aspects of robotics. This thesis builds the foundation for a mobile robot's software used to show visitors some of the autonomous mobile robot's features. The assumption for writing this thesis is that the work will be extended and therefore, references to tutorials and documentation are given. For the same reason is the robot modelled in a simulation environment due to the faster development and testing of new functionalities. The thesis is divided into three main sections. Firstly, the introduction and description of the used software and sensors is provided. Then follows the creation of the simulation environment, and the last main part is composed of software implementation on the real robot. This division is not strict as development and testing took place sometimes concurrently in simulation and real environment, therefore throughout both sections, number of cross-references exist and some algorithms are described in the experimental section, even though their development was first performed in the simulation environment.

After an agreement with the thesis's supervisor, the software was created and here described with the focus on only one robot, working and charging station. This can be extended in the future to incorporate several robots, working and charging stations.

2 Theoretical Survey

2.1 Basic Information about ROS Framework

The Robot Operating System (ROS) is a robotic framework distributed under the BSD license. ROS provides hardware abstraction, implementation of commonly-used functionality, and a high number of tools that facilitate the development of complex robotics software.[62]

Supported operating systems are Linux-based Ubuntu and Debian (only for some distributions). Other systems like Mac, Windows, and Raspbian are considered as experimental platforms [62]. ROS main advantage lies in a large online community that manifests itself in producing many open source packages, documenting them and providing advice on Discussion Forum. Another significant aspect of the ROS framework is its utilization in the industrial sector. ABB, Fanuc, Kuka or Universal Robots are examples of companies using ROS-Industrial in some of their robots.

ROS does not operate in real time (RT) operating system, but it is possible to integrate ROS with RT code. Several packages are also provided for communication with embedded devices - *rosserial_embeddedlinux*, *rosserial_arduino*, etc. The description of ROS architecture will be depicted in more detail in further sections.

The emphasis on RT, a cooperation of multiple robots and functioning properly in non-ideal networks led to the development of ROS 2. However, at the time of writing this thesis, ROS 2 significantly lacks in terms of documentation over the previous version.

2.2 Gazebo Simulation Environment

The Gazebo is an open-source (Apache-2.0) 3D simulator used in the development of robots and robotics algorithms as well as Hardware-in-the-loop (HIL) simulation. Supported platforms are all major Linux distributions and Mac. As stated in [4], installation on Windows is currently under development.

Gazebo consist of several parts. Gazebo Master is a topic name server that can handle multiple running simulations. Communication between parts is provided by the Communication Library through TCP/IP protocol. The Gazebo can run on 4 different physics engines included in the Physics Library, all except Dart are accessible in binary installation. From the three remainings, Open Dynamics Engine (ODE) was the original engine physics and it is still set as the default option. The visualization is provided through the Rendering Library using OGRE. The Rendering Library is responsible for rendering GUI (Graphical User Interface), sensor libraries, lighting, textures, and sky simulation. Many available sensors are simulated from the Sensor Generation. Users can interact with a simulation through GUI based on Qt. Interaction consists of modifying objects, changing simulation parameters, or logging simulated sensor data. Users can utilize plugins for direct access to the physics, sensor, and rendering libraries without using the communication.[63]

The Gazebo has been widely used in combination with ROS framework. Their integration is achieved through ROS packages *gazebo_ros_pkgs* that provide wrappers [1] around stand-alone Gazebo.[5]

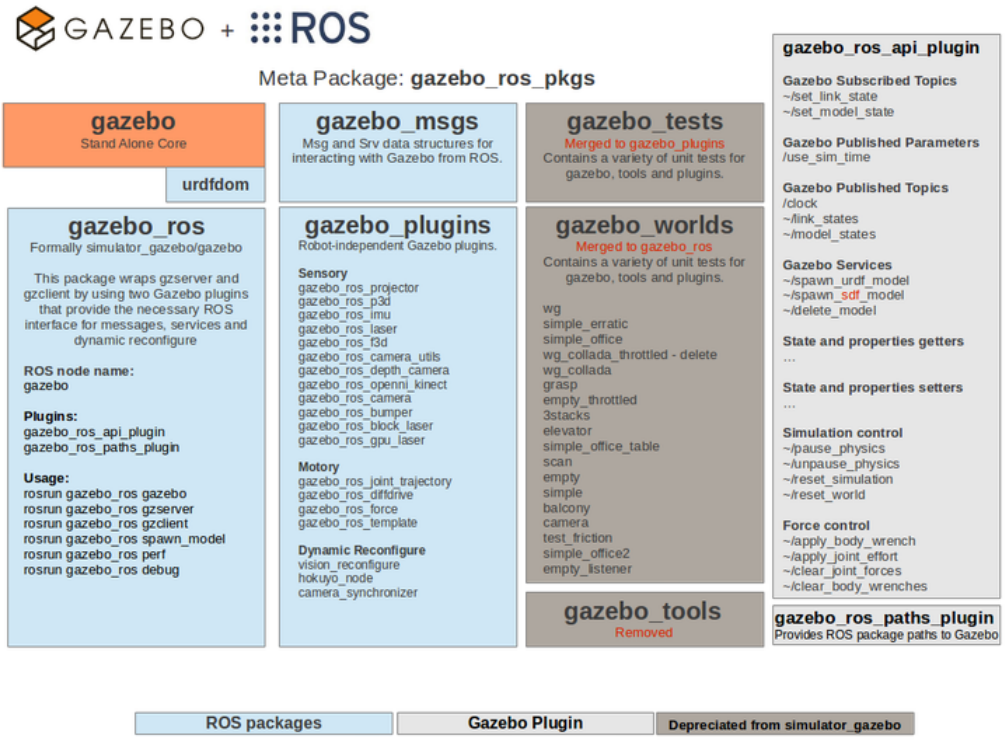


Figure 2.1: Overview of the *gazebo_ros_pkgs* interface [5]

2.3 ROS Packages Description

The software in ROS is distributed in separate packages or so-called meta-packages which contains arbitrary numbers of individual packages. A package might consist of any of the following: ROS node, ROS-independent library, a dataset, or anything else that logically forms a useful module. A node is the barest process that performs computation. Nodes communicate with each other through topics, which are named buses with anonymous publish/subscribe semantics. Packages can be created by hand or with *catkin_create_pkg*.[62]

2.3.1 Conventions in ROS

As was stated in the preceding text, ROS is an open-source framework, and therefore, some guidelines for cooperation and development must be established. ROS Enhancement Proposal (REP) serves as guidance or proposal for a new functionality. To every REP is assigned one to four digit number by which it is referred in ROS community. Here follows a brief description of the most critical REPs for a thorough understanding of indoor ROS robot localization and

[1] Wrapper function exists to call another soub-routine or system call without much additional computation

navigation packages.

REP-103

This REP, defined in [6], deals with coordinate conventions and provides a reference for units. Main principles are described below:

- Exceptions from this REP should be carefully justified and well documented
- SI units or SI-derived units should be used
- Right-handed coordinate frames are used
- Axis orientation in relation to the robot's body is:
  - ⋆ x forward
  - ⋆ y left
  - ⋆ z up
- Cartesian representation of geographic locations is East-North-Up (ENU)
  - ⋆ X east
  - ⋆ Y north
  - ⋆ Z up
- The preferred order of rotation representations:
  1. quaternion
  2. rotation matrix
  3. fixed axis roll, pitch, yaw about X, Y, Z axes respectively
  4. euler angles yaw, pitch, and roll about Z, Y, X axes respectively

REP-105

Defined in [7], REP-105 specifies conventions for naming and semantic meaning of coordinate frames for mobile robots. The following section focuses on coordinate frames for an single indoor robot:

– *base_link* is rigidly attached to the robot base, preferably in position and orientation described in 2.3.1

– *odom* is a world-fixed frame in which the robot pose evolves continuously, that is, without discrete jumps in transformation between aforementioned frames. Due to the demand for continuity, only relative data are used for computation in *odom* frame. In a typical setup that means fusing measurement from wheel encoders, IMU or visual odometry. For compensation of the accumulated drift between real and measured robot pose over time, the *map* frame must be defined.

– *map* is also a world-fixed frame in which real robot pose should not significantly drift from a measured pose. This is accomplished through discrete jumps in transformations between frames in time. Localization part of robot's software should constantly recompute position and readjust transformations to resemble real pose most accordingly.

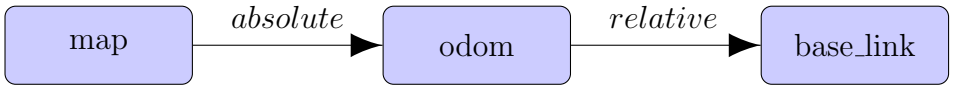


Figure 2.2: Relationship between frames for single robot

Due to the fact that the child frame can have only one parent frame in *tf* [8], the package that provides and computes all transformations in ROS ecosystem, the relationship between above mentioned frames must be established as depicted in fig. 2.2.

As was already mentioned, the transform from *odom* to *base_link* is broadcast by node fusing odometry sources. The transform from *map* to *odom* is computed by localization node. However, the same transform is not directly broadcast, while it is necessary to first receive *odom* to *base_link* transform and subtract it from the computed transform.

REP-145

REP-145, defined in [9], deals with IMU sensor drivers. Standardized frame coordination are expressed in compliance with REP-103, preferred topics name are given as:

- */imu/data_raw*: accelerometer and gyroscope data
- */imu/data*: *imu/data_raw* with orientation estimate
- */imu/mag*: magnetometer data

Also, *frame_id*[2] for IMU is defined as *imu_link*, and names for sensor standard deviation are given.

2.3.2 Localization

Two main approaches to robot localization as described by *Thrun et al.* [10] are Gaussian and Nonparametric Filters. Gaussian filters are based on the idea of representing beliefs of current, previous, and measurement state as multivariate normal distribution defined as [10]:

$$p(\boldsymbol{x}) = \det(2\pi\boldsymbol{\Sigma})^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right] \tag{2.1}$$

where $\boldsymbol{x}$ is state vector, $\boldsymbol{\mu}$ is mean vector, $\boldsymbol{\Sigma}$ is the covariance matrix which is symmetric and positive-semidefinite, and $p(\boldsymbol{x})$ denotes the density function over vector variable $\boldsymbol{x}$.

The Gaussian can be represented also in the canonical version, as opposed to the standard moments version described above. Filter utilizing a canonical Gaussian representation is called the Information Filter. The canonical representation is defined as [10]:

$$p(\boldsymbol{x}) = \eta \exp\left[-\frac{1}{2}\boldsymbol{x}^T\boldsymbol{\Omega}\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{\xi}\right] \tag{2.2}$$

where $\boldsymbol{\xi}$ is information vector, $\boldsymbol{\Omega}$ is the information matrix, and η is normalizer. $\boldsymbol{\xi}$ and $\boldsymbol{\Omega}$ can

[2]Coordinate frame in which are published data given

be obtained from their moments counterparts as follows [10]:

$$\boldsymbol{\Omega} = \boldsymbol{\Sigma}^{-1} \tag{2.3}$$

$$\boldsymbol{\xi} = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} \tag{2.4}$$

Both representations are furthered used in 5.2.3.

Nonparametric filters approximate the posterior by a finite number of values. To the most well-known nonparametric filter belongs the Particle filter. For a detailed description of the Extended Kalman Filter (EKF) and the Particle Filter, implemented in ROS as one of the options of *robot_localization*, respectively *AMCL*, refer to *Thrun et al.* [10]. The assumption for understanding the next sections is that the reader is familiar with EKF.

AMCL estimates pose based on a known map, transforms and laser scans. Due to the fact, that in the final version of this project, only one robot will be equipped with Lidar, it would take a significant amount work to modify the package to work solely on observing fiducials in the environment, as is the aim of this work. Therefore the package *robot_localization* was chosen and is further described in the next section.

robot_localization

Implementation of EKF in *robot_localization* is fairly well documented and offers many possible options, exposed as ROS parameters and services, to customize sensor fusion. Therefore it is a quite popular localization tool amongst the ROS community. To the main features belongs [15]:

- support of an arbitrary number of input sources
- possible input types of ROS *messages* are *nav_msgs/Odometry*, *sensor_msgs/Imu*, *geometry_msgs/PoseWithCovarianceStamped* or *geometry_msgs/TwistWithCovarianceStamped*.
- input customization of each sensor - node allows excluding unwanted data from a sensor
- possibility to estimate only parts of the state vector representing robot operating in planar environment
- asynchronous fusion of input measurement
- reverting filter to the last state before the reception of lagged measurement and processing all measurements until the current time
- exclusion of measurement outliers based on Mahalanobis distance

Using Mahalanobis distance in outliers rejection, instead of commonly used Euclidean distance, brings the main advantage in accounting correlation between measured variables. Basic comparison between the Mahalanobis and Eculidean distance is depicted in figure 2.3. Mahalanobis distance is defined for multivariate normal distribution as follows [11]:

$$\mathrm{MD}\left(\boldsymbol{x}, \boldsymbol{\mu}\right) = \sqrt{\left(\boldsymbol{x} - \boldsymbol{\mu}\right)\boldsymbol{\Sigma}^{-1}\left(\boldsymbol{x} - \boldsymbol{\mu}\right)^{T}} \tag{2.5}$$

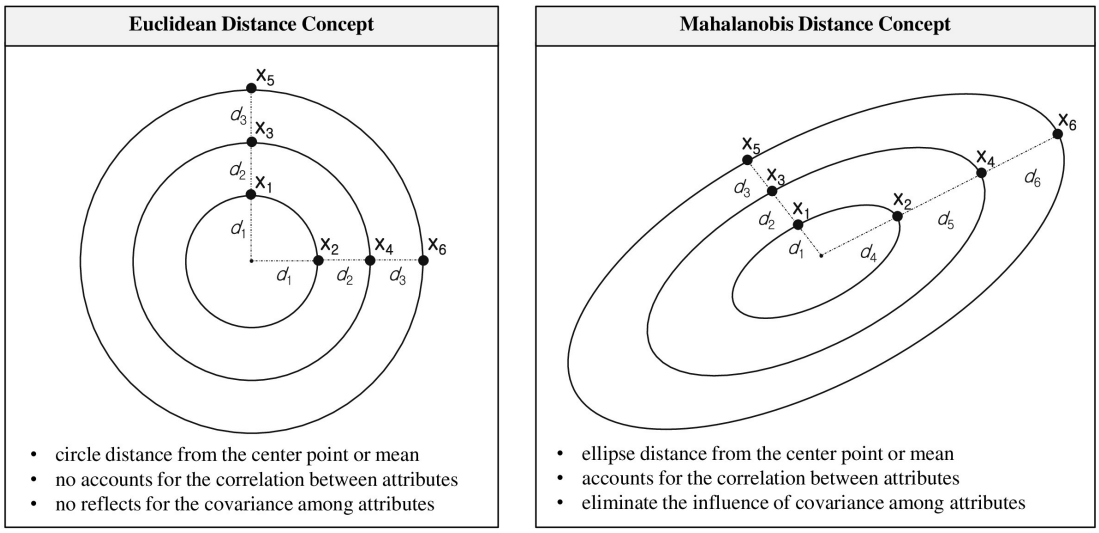


Figure 2.3: Comparison of the Euclidean and Mahalanobis distance in 2D [12]

Computed Mahalanobis distance expresses the number of variances from mean vector $\boldsymbol{\mu}$ to measurement vector $\boldsymbol{x}$. Based on this value, measurement outliers can be rejected.

As stated for example in [13], if $\boldsymbol{x}$ has p-dimensional Gaussian distribution then $\mathrm{MD}^2(\boldsymbol{x}, \boldsymbol{\mu})$ has a χ^2_p distribution.

Another part of EKF implementation in ROS, which needs to be discussed, is the prediction step of EKF algorithm. Full 12-dimensional state vector $\boldsymbol{x}$ is defined as:

$$\boldsymbol{x} = (x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}, \ddot{x}, \ddot{y}, \ddot{z}) \tag{2.6}$$

where $x, y, z, \dot{x}, \dot{y}, \dot{z}, \ddot{x}, \ddot{y}, \ddot{z}$ are position, translational velocity and acceleration respectively and $\phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}$ are rotations around fixed axis X, Y, Z, respectively angular velocities around these axis.

As described in *T. Moore, D. Stouch, 2014* [14], the calculation of predicted state vector is:

$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}) + \boldsymbol{w}_{k-1} \tag{2.7}$$

where $\boldsymbol{w}_{k-1}$ is the process noise, and f is described as a standard 3D kinematic model derived from Newtonian mechanics.

As can be seen in 2.7, there are no inputs to the prediction step. The shortcoming of this method is the inability to predict correctly $\ddot{x}, \ddot{y}, \ddot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}$. From reviewing documentation of *robot_localization* [15] and code accessible on GitHub [16], 2 solutions are implemented in *robot_localization* to overcome problems with predicting above-mentioned states:

- predicted value of the state is the same as in the previous step
- value of the state is predicted according to the difference of control term used in the movement of robot and current translational and angular velocity

Both methods are further discussed in 4.3.1.

imu_filter_madgwick

This package incorporates Madgwick algorithm [17] for estimating orientation with respect to the gravity and Earth's magnetic field from the fusion of angular velocities, accelerations, and measurement from the magnetometer. As accelerations provide a reference for attitude, inclination errors of the measured magnetic field can be compensated. Another version of the algorithm fusing only angular velocities and accelerations exists but will not be described here. The algorithm utilizes quaternion and quaternion derivative for representation of angle, respectively angular velocity. The problem of finding correct orientation is solved as an optimization task using a gradient descent algorithm. The algorithm is depicted and briefly summarized in figure 2.4.

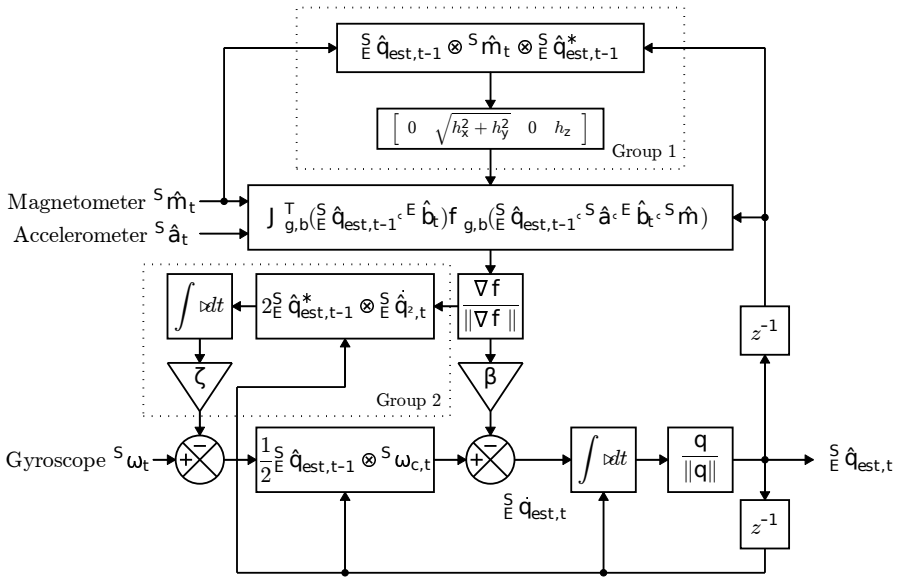


Figure 2.4: Block diagram of the complete Madgwick algorithm [17]: Measurements from magnetometer (after compensation for inclination error) and accelerometer are along with orientation estimate from the previous step the inputs in the computation of the gradient which resembles the direction of the estimated rate of change of orientation. Thereafter the resut is multiplied by filter gain (parameter β) and subtracted from the compensated measurement from the gyroscope (after rotating the gyroscope measurement from which bias was subtracted with quaternion orientation estimate from the previous step). This estimated rate of change of orientation is then numerically integrated (with orientation estimate from previous step) and then normalized serves as filter output. Gyroscope bias is computed by numerically integrating rotated estimated error in the rate of change of orientation and multiplying by parameter ζ.

Both filter's parameters values should be roughly based on equations given in article [17], β is calculated from mean zero gyroscope error and ζ is calculated from an estimated gyroscope bias drift.

The higher β gain is, the faster the filter converges, but bigger noise is introduced to the output orientation estimate [17].

Laser scan matcher

This package takes *sensor_msgs/LaserScan* or *sensor_msgs/PointCloud2* messages and by comparing consecutive messages computes transformation of the sensor in between messages. It

can be used without additional sources as stand-alone odometry estimator. However, as stated in documentation [18], providing additional odometry input improves the speed of convergence and accuracy of the estimated transformation. *Laser_scan_matcher* supports 4 kinds of inputs:

- IMU provides a change of the orientation angle
- Wheel odometry provides a change in position and orientation angle
- Constant velocity model - the assumption of constant velocity between steps, computes initial pose difference based on provided velocity
- Zero velocity model - the assumption of the same sensor pose as in the previous step

For many additional options to set refer to documentation [18].

2.3.3 SLAM

Simultaneous Localization and Mapping (SLAM) represents one of the fundamental problems of the mobile robotics in which robot does not have access to a map of the environment and concurrently does not know its position. According to *Thrun et al.* [10], there are two main forms of SLAM from a probabilistic point of view: *online* SLAM in which posterior is estimated only for the current time step. Examples are Extended Kalman Filter SLAM and Extended Information Filter SLAM. The *full* SLAM, on the other hand, estimates the full path over the entire time. For example, GraphSLAM belongs to the full SLAM. Some algorithms, for example, FastSLAM, performs online and full SLAM.

There are many packages in ROS that implements different approaches to SLAM using different algorithms or map representations. According to findings in the article [19] where several different SLAM packages were compared, *KartoSLAM*, *HectorSLAM*, and *Gmapping* produce the most accurate maps. From the three, *Gmapping* has the best documentation and therefore was chosen to implement.

Gmapping

Gmapping belongs to particle filter algorithms, more specifically to FastSLAM 2.0 algorithm with adaptive particle resampling.

Because of the complex problem that SLAM represents and the fact that it does not belong to the aims of this thesis, no detailed description is discussed here. The reader is encouraged to read the excellent book [10] for SLAM basics and refer to article [20] for the working principle of the *gmapping* package and documentation [21] of *gmapping*. As the guide for setting parameters of *Gmapping* article [22] was used.

2.3.4 Navigation

Navigation of robot is usually composed of several parts - planning and controlling or sometimes called global, and local planners, environment representation, recovery behaviors and a higher decision layer that manages these sub-processes. Typical setup of ROS navigation stack using meta-package *move_base* is displayed in fig. 2.5

Global planner (further referred to as planner) takes into consideration the whole part of the environment needed to move from start to goal position. In a standard setup in ROS

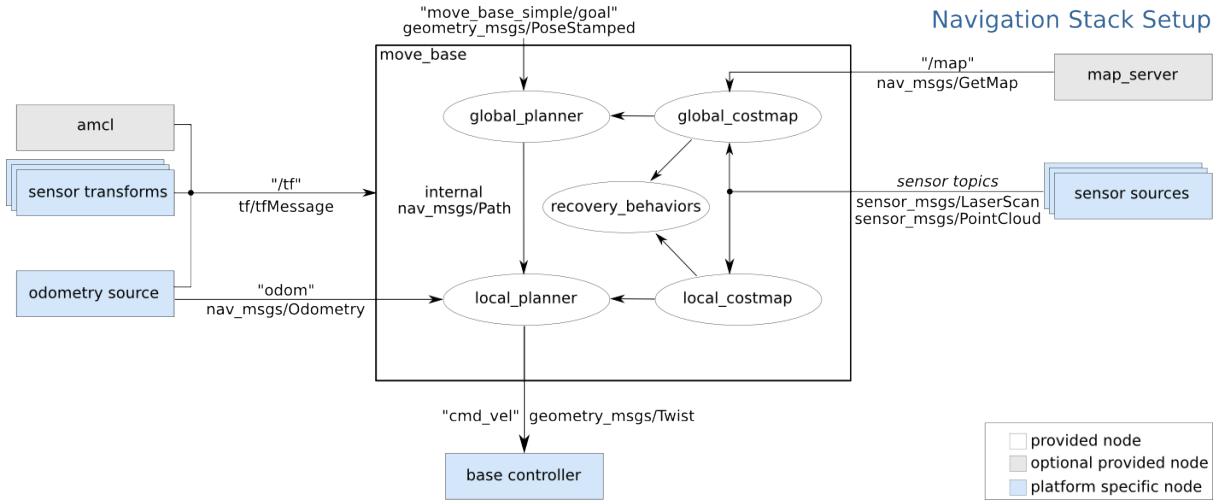


Figure 2.5: An overview of *move_base* setup [23]

navigation, the planner will simply plan a path to the goal without consideration of kinematic and dynamic constraints of robot[3].

Local planners (further referred only as controllers) will receive path from a planner and will compute desired velocity commands that will result in a robot moving to the goal position, assuming that some node equivalent to the *base_controller*, as depicted in fig. 2.5, will subscribe to published velocity and convert it to physical signals controlling motors of the robot. Controllers subscribe to *odom* frame in *tf*, defined in 2.3.1, to determine the robot position and to */odometry* message that contains current velocity of the robot in *base_link*.

When the controller does not compute any desired velocity due to possible obstacles collision, recovery behavior will take control of the robot. If recovery action is successful, a planner will try to find a new path to the goal position.

The specific implementation of a planner, controller and recovery behavior must adhere to the interface specified in package *nav_core* [23]. This is especially useful in the possibility of trying different implementations of above-mentioned sub-processes without too difficult modification of the navigation stack.

Another part of the navigation system constitutes of environment representation. *Move_base* supports package *costmap_2d* that builds an occupancy grid containing values defining the probability of an obstacle being in each cell. Package subscribes to sensor sources and can add or remove obstacle from the map. Two maps are used in navigation setup, one called global costmap for the planner, and local costmap for the controller.

As far as the choice of the navigation meta-package for the final implementation is concerned, *move_base_flex* is backward compatible with *move_base* and offers many other options for creating a complex navigation system. Therefore, taking in consideration possible enlargement of the project in the future, *move_base_flex* was chosen and is further described.

move_base_flex

Move_base_flex (MBF) provides separation between abstract navigation and specific implementation. Navigation is divided into three separate actions or one complex action as depicted in fig. 2.6. Each action produces comprehensive result and feedback information, which can be utilized in customized navigation for complex behaviors. By separation of abstract navigation,

[3]There are some planners (for example *sbpl_lattice_planner*) that take into consideration kinematic and dynamic constraint, but, for brevity of this text, they will not be further discussed

MBF is not bound to any particular map representation. MBF also allows to load and use multiple planners, controllers, and recovery plugins. As commercial use of MBF is concerned, Magazino Gmbh, one of the authors of MBF, successfully operates robots using MBF in six different commercial settings [24].

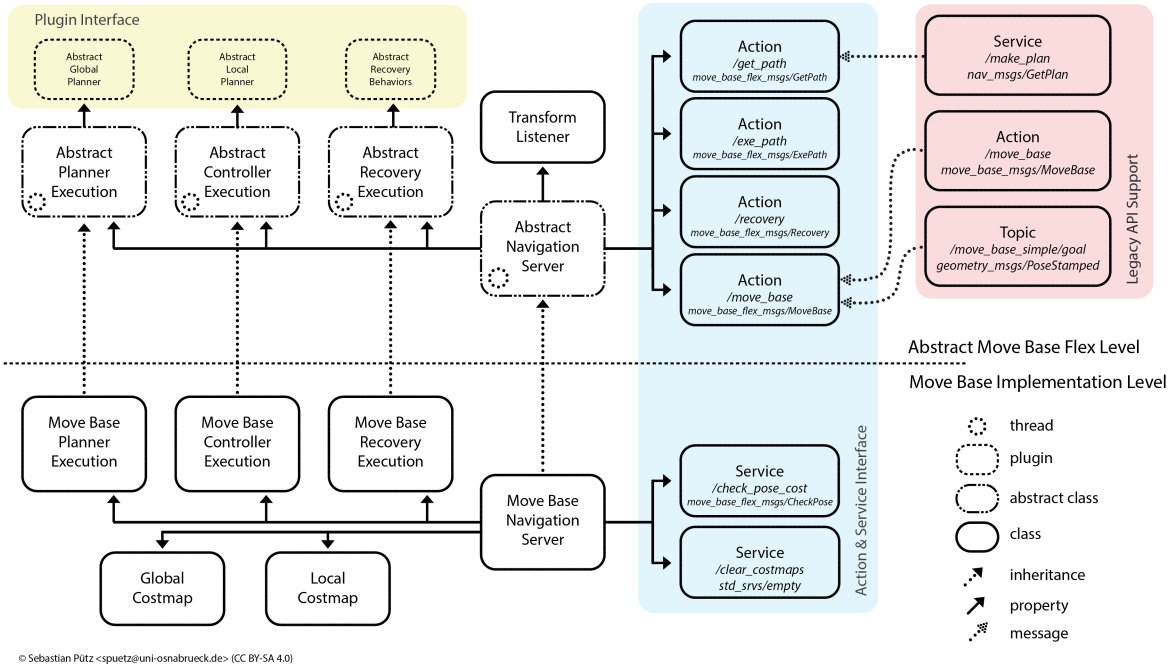


Figure 2.6: The *move_base_flex* architecture: Shown is the relationship between Abstract navigation level and Costmap implementation level [24]

In the following subsection, the packages, and plugins that are picked to create the desired navigation functionality in *move_base_flex* are shortly described.

costmap_2d This package serves as a customizable structure that exposes its functionality in layers. The Static map layer contains information about the portion of costmap that is not changing, whereas the Obstacle map layer either mark (add) or clear (remove) obstacle from the costmap for each observation of supported sensor. To add new values around obstacles, the Inflation layer exists. Calculation of new values for cells further than the inscribed radius and closer than the inflation radius is described in layer documentation [25], here modified for better readability:

$$cost = 253e^{-scale(distance-radius)} \tag{2.8}$$

where *radius* is the inscribed radius, *distance* is the distance from obstacle and *scale* is the cost scaling factor. All geometric representations are depicted in fig. 2.7. Inflation settings have a significant effect on the planner and controller behavior.

global_planner This package uses either A* or Djikstra algorithm to expand cells from start to goal cell. To every expanded cell, value is assigned according to distance from start and value of a given cell in the global costmap. Values from costmap are transformed according to

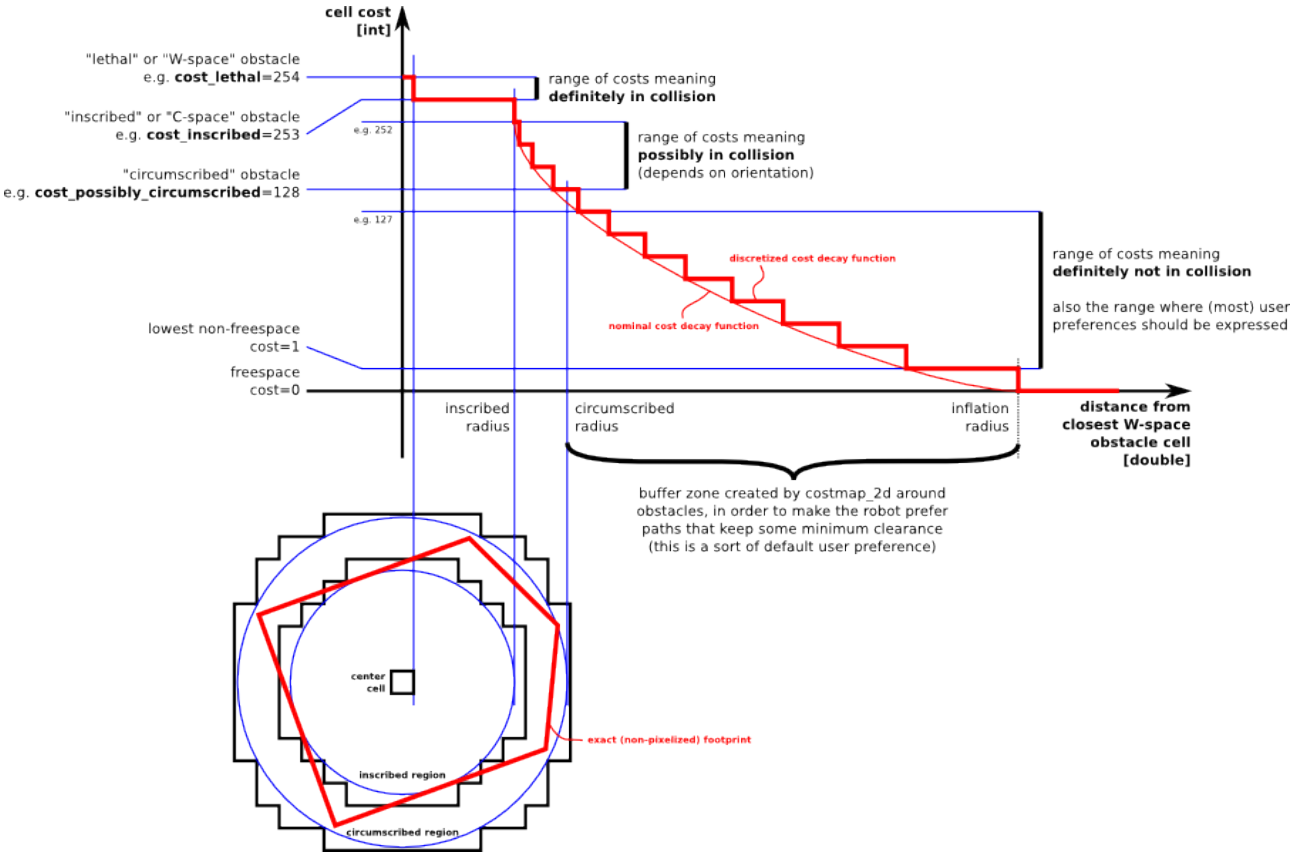


Figure 2.7: The Inflation layer: Propagation of cost values from occupied cells to neighbouring cells [26]

the ROS Navigation Tuning Guide [27] as follows:

$$cost = COST\_NEUTRAL + COST\_FACTOR \cdot costmap\_cost\_value \tag{2.9}$$

where $COST\_NEUTRAL$ and $COST\_FACTOR$ are parameters for customization of the setting of the *global_planner* and *costmap_cost_value* is the value of the cell in global costmap. The Final path is computed using a gradient descent algorithm. The path is then given as a list of poses to which orientation can be additionally added by orientation filter.

dwa_local_planner Dynamic Window Approach (DWA) works with the assumption of modelling robot velocity as piecewise constant function in time. Therefore, robot trajectories are formed from finitely many segments of circles segments [28]. Algorithm of DWA can be described as follows:

1. Restrict the search space velocity only to those velocities that are admissible in terms of collision with obstacles and dynamic constraints of the robot
2. Discretely sample in the robot's velocity space
3. Perform forward simulation for each sampled velocity
4. Evaluate each trajectory using the cost function
5. Pick the highest-scoring trajectory

The cost function in package is described in package documentation [29], here modified for better readability:

$$cost = bias_p \cdot distance_p + bias_g \cdot distance_g + scale \cdot cost_o \tag{2.10}$$

where $distance_p$ is the distance to path from the endpoint of the trajectory, $distance_g$ is the distance to local goal from the endpoint of the trajectory and $cost_o$ is the maximum obstacle cost along the trajectory. Further, $bias_p$, $bias_g$, and $scale$ are parameters for setting the weight for each criterion.

rotate_recovery* and *moveback_recovery As was already described; recovery action takes over control when the controller can't compute any feasible trajectory. As the name of *moveback_recovery* implies, while executing this behavior, the robot attempts to move backward with the desired velocity to set distance. While performing *rotate_recovery*, the robot rotates for 360 degrees, if it would not result in a collision with obstacles.

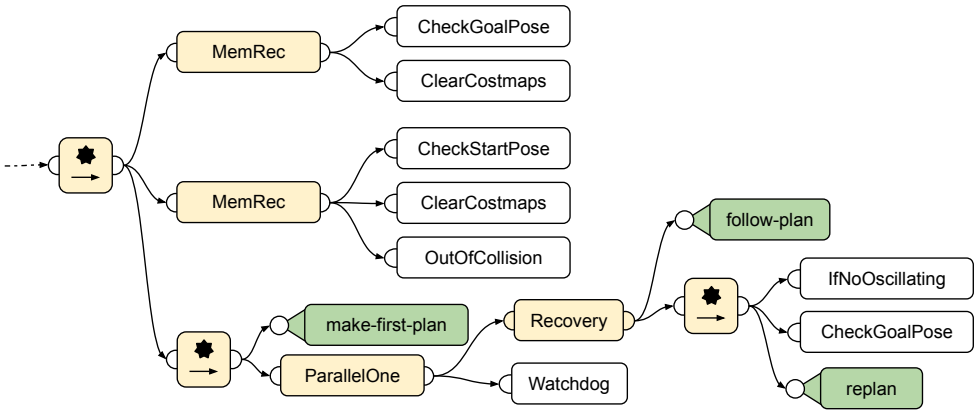


Figure 2.8: Navigation Sub-Tree [24]

hierarchical finite state machine To create a complex robotic system, many different components must be integrated into a hybrid architecture that creates global system behavior. In the introductory paper for MBF [24], two approaches for designing flexible strategy are presented - Behavior Tree (BT) and state machine (SMACH) package. Example of BT in navigation using MBF can be seen in fig. 2.8. For its excellent integration with ROS and introspection possibilities, SMACH was chosen and is described in separate section 2.3.5.

2.3.5 SMACH

SMACH is a stand-alone Python library for creating complex robot behavior. It functions separately from ROS, but the package *smach_ros* exists to integrate SMACH tightly with ROS. From SMACH state, actions and services can be called or act as servers. Listening or subscribing to topics or using ROS debugging possibilities can also be utilized in SMACH state. SMACH structures compose of individual states that can be grouped in one of the containers: StateMachine, Concurrence, Sequence, Iterator or user-defined container. The container can be added into another container and act as a single state.

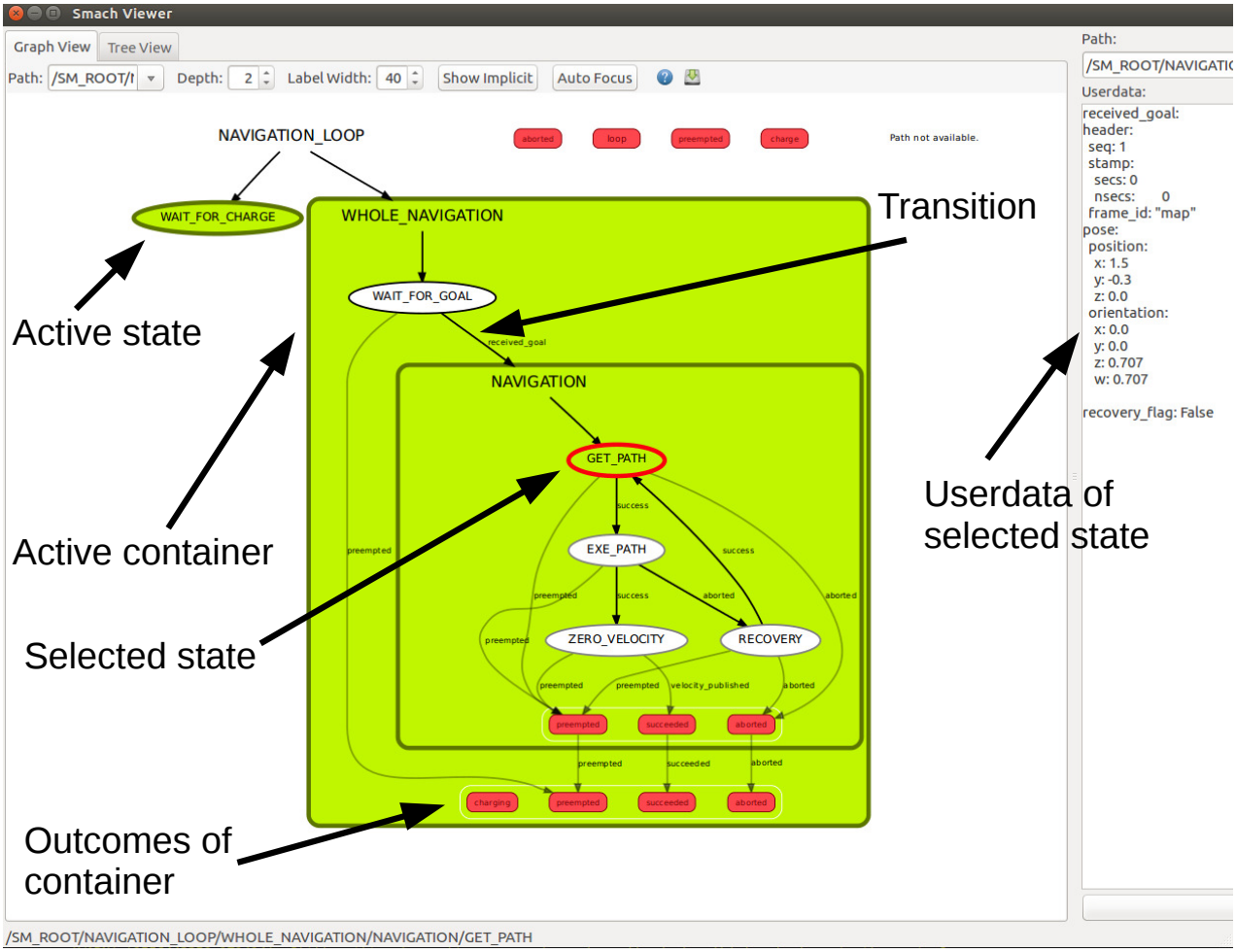


Figure 2.9: SMACH introspection: Shown is the content of Concurrence container *NAVIGATION_LOOP* in which several states or another whole container, considered as one state, can run simultaneously. There runs *WAIT_FOR_CHARGE* with *WHOLE_NAVIGATION* at the same time - colored in green. Userdata of selected state *GET_PATH*, which is also the running state of *NAVIGATION* container, is displayed on the right side.

For introspection the state of SMACH, the package *smach_viewer* provides GUI. It can visualize currently active state, userdata that is exchanged between states are shown and the transitions between containers or states are easy to understand through graphical edges.

Since its creation, SMACH has been successfully used in projects involving autonomous recharging, fetching a drink from a refrigerator, playing billiard and many more, as stated in its introductory article in 2010 [30].

2.4 Sensors and Their Usage in Robotics

The further described sensors were chosen for the lowest possible price but enough quality to ensure autonomous navigation of robots. For the absolute localization, a camera was chosen as Lidar is much more expensive and the camera can also be used for other purposes. One Lidar was available in the Mechatronics laboratory and was used for SLAM. For the relative localization, low-price IMU and wheel encoders were used.

2.4.1 Camera

As is further described in 5.1, Raspberry Pi 3B+ is used, therefore as the first option for the camera, the Raspberry Pi Camera Module v2 with 8 Mpx emerged. As written in specification [31], video can be captured at 1080p with 30 *fps*, 720p with *fps*, 640×480 with 90 *fps* or another chosen resolution.

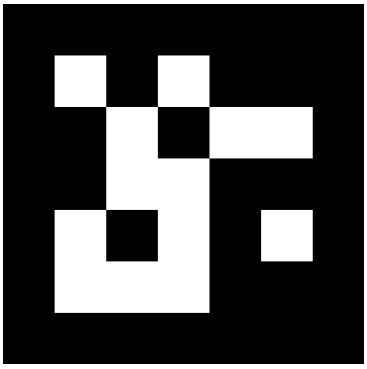


Figure 2.10: ArUco marker

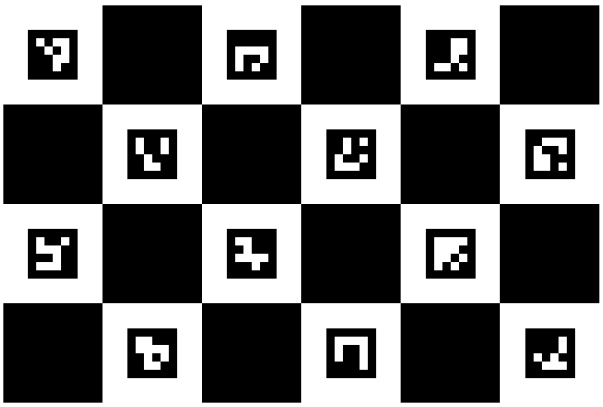


Figure 2.11: ChArUco board

OpenCV OpenCV is a computer vision library developed originally by Intel, currently is open source, released under BSD 3-Clause [32]. To supported languages belongs C++, Python, Java and MATLAB. It provides many functions with extensive documentation and tutorials. When installing the full version of ROS, OpenCV is also installed.

ArUco Introduced in 2014 [33], ArUco library composes of tools to generate [34] fiducial[4] markers and detecting them in the image. ArUco marker, visible in fig. 2.10, is composed of an outer black border and inner binary matrix that defines its unique identifier (id). The inner square matrix can be built from 4×4 to 6×6 bits. Dictionaries are lists of markers, differentiating in the number of marker's bits and the total number of markers in the dictionary. The biggest utilization of ArUco markers is in estimating the pose of the marker with respect to the camera. Tutorial for marker creation and pose estimation can be found on the web [35]. ChArUco board composes of the chessboard and ArUco markers. It combines the fast detection and versatility of ArUco markers with a possibility to accurately refine chessboard corners. Therefore, ChArUco boards are used when high precision is essential, e.g. in camera calibration, accurate pose estimation, and others. As stated in OpenCV documentation [36], ChArUco boards are better for high precision than ArUco boards.

[4]term used in computer vision for a reference object

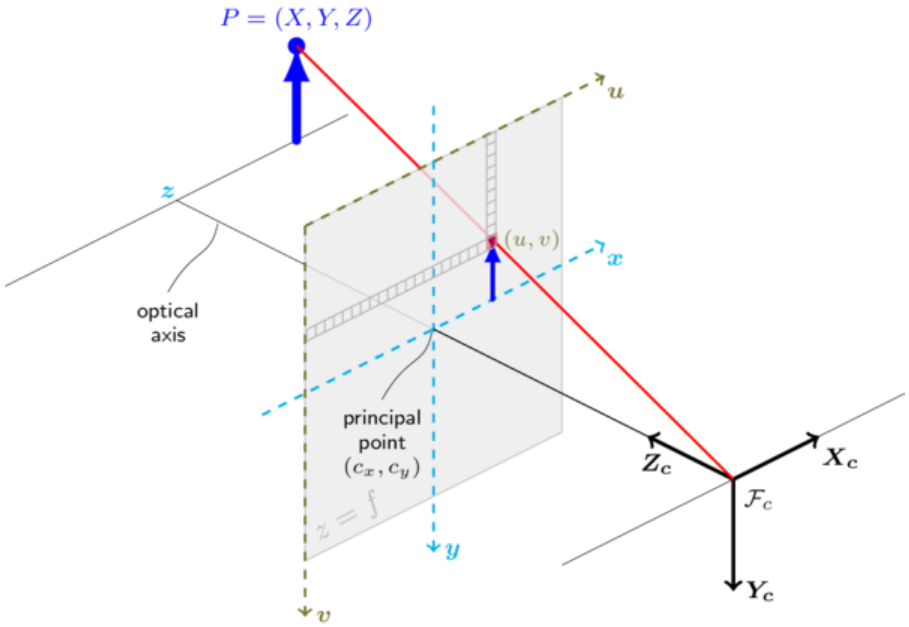


Figure 2.12: Pin hole camera model [37]

Camera calibration

For obtaining a real-world unit from the camera image, the camera must be calibrated. To do that, projection model for projecting 3D points into the image plane must be established. Using perspective transformation in so-called pinhole camera model (figure 2.12), transformation is described using equation [37]:

$$s\boldsymbol{m}' = \boldsymbol{A}\,[\boldsymbol{R}|\boldsymbol{t}]\,\boldsymbol{M}' \tag{2.11}$$

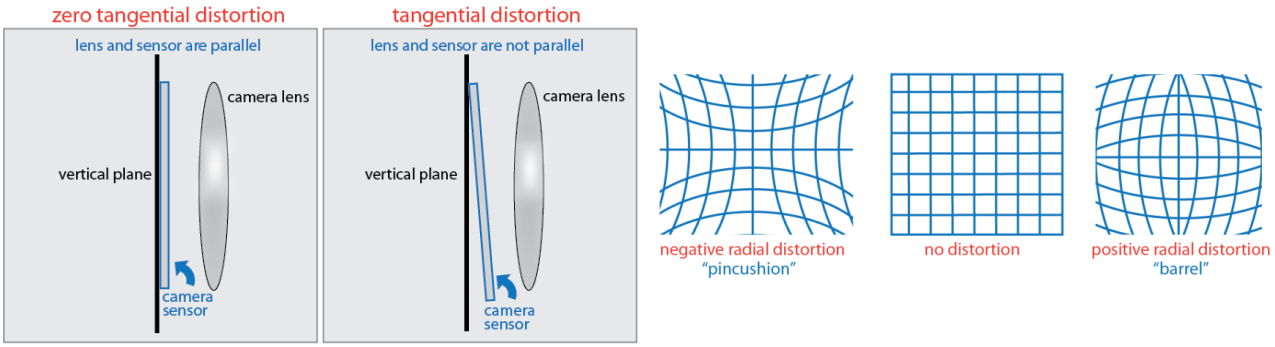
where s is scale factor, $\boldsymbol{m}'$ is the array of coordinates of the projection point in pixels, $\boldsymbol{A}$ is the camera matrix, $[\boldsymbol{R}|\boldsymbol{t}]$ is the joint rotation-translation matrix or sometimes called extrinsic matrix and $\boldsymbol{M}'$ is the array of coordinates of point in the world coordinate system. Rewritten in the detailed form:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.12}$$

where (X, Y, Z) are the coordinates of a point in the world coordinate space, (u, v) are the coordinates of the projected point in pixels, (c_x, c_y) are the coordinates of principal point, f_x, f_y are the focal lengths in pixels and r_{xx} with t_x are components of the transformation matrix $[\boldsymbol{R}|\boldsymbol{t}]$.

As can be seen in equations 2.11 and 2.12, if the scale of the image is changed, the camera matrix (sometimes called matrix of intrinsic parameters) must be scaled as well. Therefore the calibration is necessary to do only for one resolution and can be rescaled for another resolution as long as the focal length of the camera is fixed.

However, cameras used in the real world are not perfect, and their lenses have radial and tangential distortions (showed in figure 2.13). Due to this distortions, the existing pinhole



(a) Tangential distortion (b) Radial distortion

Figure 2.13: Distortions [39]: Subfigure a) shows how tangential distortion originates. Subfigure b) shows how different radial distortions are manifested in picture.

camera model, assuming $z \neq 0$, must be extended to [37]:

$$\begin{aligned} x' &= x/z \\ y' &= y/z \\ r^2 &= x'^2 + y'^2 \\ x'' &= x' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_1 x' y' + p_2 \left(r^2 + 2x'^2 \right) \\ y'' &= y' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + p_1 \left(r^2 + 2y'^2 \right) + 2p_2 x' y' \\ u &= f_x x'' + c_x \\ v &= f_y y'' + c_y \end{aligned} \tag{2.13}$$

where $k_1, k_2, k_3, k_4, k_5, k_6$ are radial and p_1, p_2 are tangential distortion coefficients.

The calibration process can be described as computing camera matrix and distortion coefficients from at least several images when some object size in image is known. In OpenCV, calibration can be done using ChArUco board fairly straightforward following documentation [38]. If using another image resolution, the camera matrix must be rescaled, while distortion coefficients remain the same [37].

2.4.2 IMU

IMUs can be bought for a reasonable price and therefore belong to the most used sensor in mobile robotics. In this project, *LSM9S1* is used. *LSM9S1* incorporates three-axis accelerometer, gyroscope, and magnetometer and therefore can be called 9 DOF IMU. Specified ranges in datasheet [40] are ± 2 – 16 g for the accelerometer, $\pm$ 245 – 2000 for the gyroscope $°/s$ and $\pm$ 4 – 16 *gauss* for the magnetometer. The module can communicate via SPI or I^2C

IMU needs to be calibrated to give better results. Accelerometer and gyroscope calibration can be done using basic operations of averaging and scaling or in a more advanced manner as described in the article [41] where axis non-orthogonality, scale, and bias error are estimated. The same errors are present in the magnetometer. However, magnetic deviations resemble the biggest problem of magnetometers.

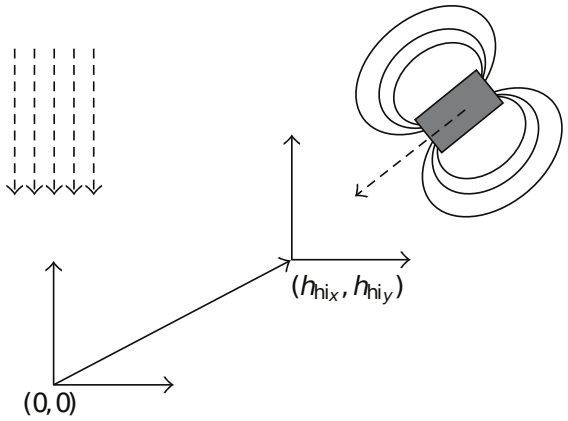
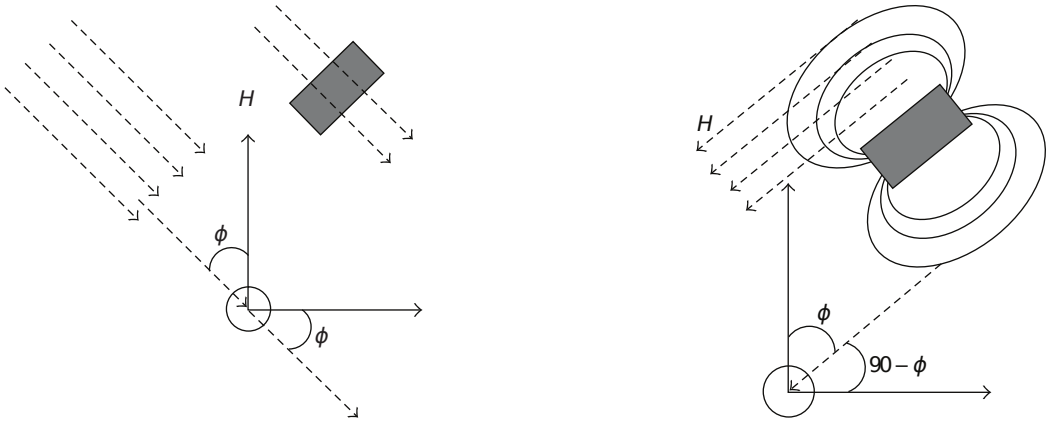


Figure 2.14: Hard iron error [42]: Presence of hard iron errors causes bias in the sensed magnetic field. A permanent field induces hard iron errors in all orientations.



(a) Negligible effects of soft iron errors (b) Maximal effect of soft iron errors

Figure 2.15: Soft iron errors [42]: Soft iron errors are induced by a complex magnetic field generated by ferro-magnetic materials that have a dependent magnitude of magnetic field on the mutual angle of material's and Earth's magnetic field. In subfigure a) ferrous material generates a magnetic field in a parallel direction to Earth's. Thus a small error is introduced. In subfigure b), Earth's magnetic field is aligned with magnetic poles of error source; thus soft iron errors have significant effect on the measurement.

Magnetometer calibration

According to the article [42], a complete error model for three-axis magnetometer is defined as:

$$\hat{\boldsymbol{h}} = \boldsymbol{A}\boldsymbol{h} + \boldsymbol{b} + \boldsymbol{\varepsilon} \tag{2.14}$$

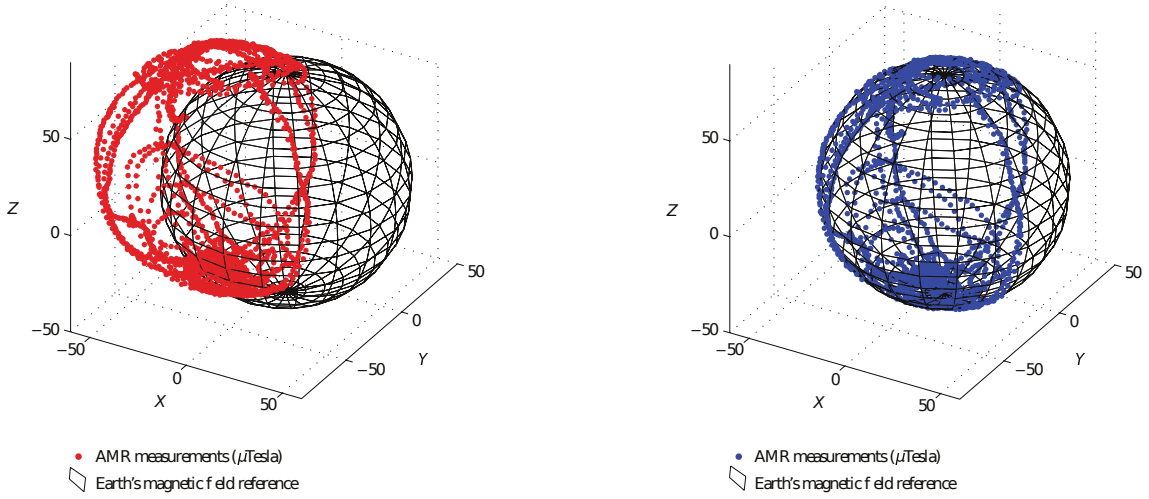
where $\boldsymbol{A}$ is matrix combining scale factors, misalignments, and soft iron (figure 2.15) disturbances, $\boldsymbol{b}$ represents the combined bias of the sensor offset and hard iron (figure 2.14) disturbances, $\boldsymbol{\varepsilon}$ is noise with normal distribution and zero mean value, $\boldsymbol{h}$ is the error-free magnetic field in the sensor frame and $\hat{\boldsymbol{h}}$ is the reading from magnetometer in the sensor frame.

In the article [42], mathematical derivation of obtaining calibration values from the measurement to compensate for errors introduced in equation 2.14 can be found. The derivation is not presented here for the conciseness of this text, instead only the main concept is introduced here.

If the error-free magnetometer in the magnetic deviation-free environment is rotated about

three world fixed axis by many different angles and the endpoint of the measured vector is plotted, an ideal sphere with a radius equal to the magnitude of the Earth's magnetic field in the measuring place appears. On the other hand, if the magnetometer has measuring error and magnetic deviations are present in the environment, the resulting plane is ellipsoid with the center of symmetry in offset. Transforming ellipsoid to sphere through adaptive least square method, described in [43], is the key concept behind calibration. After obtaining the calibration coefficients, error-free measurement is achieved through removing offset by the computed array and then by multiplying result after subtraction with the computed transformation matrix.

In summary, the calibration process can be defined as a computing offset array and transformation matrix.



(a) Uncalibrated magnetic field measurement (b) Calibrated magnetic field measurement

Figure 2.16: Comparison of uncalibrated and calibrated measurement of the magnetic field [42]: The ellipsoid is transformed into the sphere

2.4.3 Lidar

To another used sensor belongs Light Detection and Ranging (Lidar) sensor Hokuyo URG-04LX-UG01 (further referred to as URG) which was available in the Mechatronics laboratory. URG distance measurement works on the principle of the phase difference calculation, described for example, in the book [44]. The most important characteristics from the specifications [45] are:

- detection in one plane
- Scan Angle: 240°
- Angular resolution: $\approx 0.36°$
- scan frequency: 10 *Hz*
- Guaranteed accuracy for white paper: 3 %
- Detectable range: 20 – 5600 *mm*
- Resolution: 1 *mm*

Figure 2.17: Hokuyo URG-04LX-UG01 [46]

URG was used in several theses, and therefore its characteristics were already measured. In thesis [47], extensive measurement of accuracy dependent on object material, color, distance, and the angle was carried out and confirmed manufacturer's specifications for white paper detection but showed inferior accuracy for other objects.

2.4.4 Wheel Encoders

On each motor's shaft encoders are mounted. Encoders based on Hall effect from *Pololu* company are used. From reading their specification on the web [48], maximal resolution 20 counts per revolution can be obtained in the quadrature mode. For determining robot speed from wheel encoders, skid steer kinematics model must be established.

Kinematics model

Model is derived according to these assumptions :

1. the geometric center and the center of the mass lies in the same location
2. angular velocities of the same-side wheels are equal
3. four wheels of robot are always in contact with the firm ground surface

Kinematics relation between robot velocity and wheel velocities can be represented as follows [49]:

$$\begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = \boldsymbol{J}_\omega \begin{bmatrix} \omega_l r \\ \omega_r r \end{bmatrix} \tag{2.15}$$

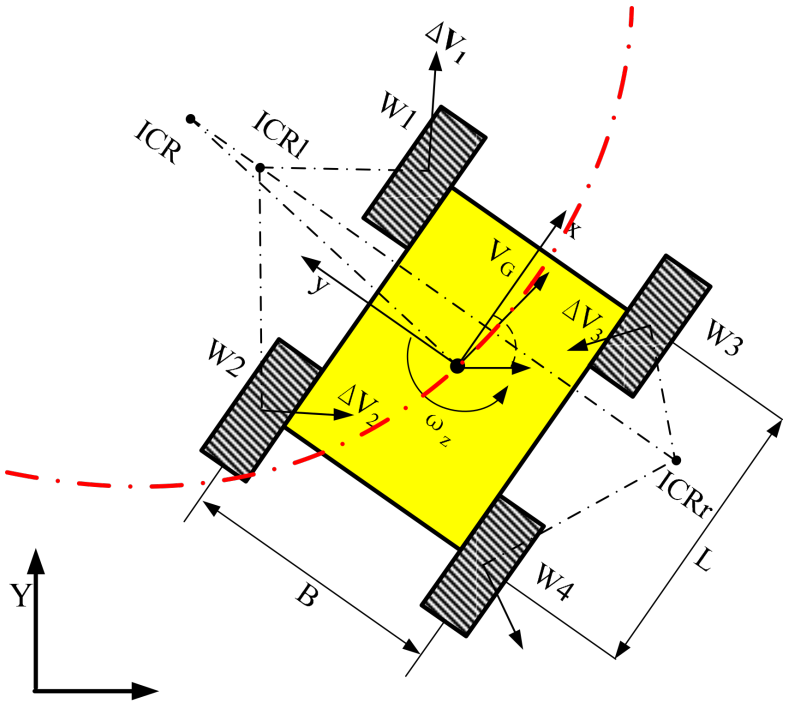


Figure 2.18: Kinematics model of skid-steer robot [49]

where v_x, v_y and ω_z are robot velocity, r is wheel radius and ω_r, ω_l are angular velocities of wheels on respective side. $\boldsymbol{J}_\omega$ is defined as:

$$\boldsymbol{J}_\omega = \frac{1}{y_l - y_r} \begin{bmatrix} -y_r & y_l \\ x_G & -x_G \\ -1 & 1 \end{bmatrix} \tag{2.16}$$

where (x_l, y_l), (x_r, y_r) and (x_G, y_G) are coordinates of instantaneous centers of rotation (ICR) of the left-side tread (ICR_l), right-side tread (ICR_r) and the robot body (ICR_G) respectively.

For the symmetrical kinematics model ($x_G = 0$) can be deduced the new form of $\boldsymbol{J}_\omega$:

$$\boldsymbol{J}_\omega = \frac{1}{y_l - y_r} \begin{bmatrix} -y_0 & y_0 \\ 0 & 0 \\ -1 & 1 \end{bmatrix} \tag{2.17}$$

where $y_0 = -y_r = y_l$

These equations represent non-holonomic restriction in the motion plane because $\boldsymbol{J}_\omega$ has no inverse [49].

Based on these equations, the instantaneous radius of the path curvature can be obtained as:

$$R = \frac{v_G}{\omega_z} = \frac{-v_l + v_r}{-v_l + v_r} y_0 = \lambda y_0 \tag{2.18}$$

where λ is non-dimensional path curvature, which definition is obvious from the above equation.

Then ICR coefficient χ can be introduced as:

$$\chi = \frac{2y_0}{B}, \qquad \chi \geq 1 \tag{2.19}$$

where B is track width, and χ represents occurring slippage. If χ equals 1, the kinematics model is the same as for the ideal differential drive. The main difference between the models is that ICR values for ideal wheels are constants and corresponds with ground contact points [49].

From simulations and experiments in the article [49], the following dependence was deduced:

$$\chi(\lambda) = 1 + \frac{a}{1 + b\sqrt{|\lambda|}}, \qquad \lambda \in [0, 10] \tag{2.20}$$

where a and b are experimentally estimated constants.

Another important finding from simulation and experiments in the article [49] is that χ is not dependent on magnitude of velocity and in comparison with χ as a static number, variable χ gives better accuracy in dead-reckoning[5].

[5]localization based on only positionally relative information

3 Problem Analysis

This thesis is part of the project that will result in the Exhibition of Robotics in the Brno Technical Museum. Visitors of the museum, presumably mostly children, will have the opportunity to manually control the robot, fulfill entertaining or educational mini-games and in other ways dive into the robotics world.

Robots used in the exhibition will be cheap, their components mostly easy-to-replace and equipped only with wheel encoders, IMU and camera. One of them will be equipped with Lidar. Fiducials in the environment will be used for localization and Augmented Reality (AR). Control station will include, apart from hardware interface for controlling - e.g., joystick, buttons; also computer, therefore software can run on this PC without too much computational limitations.

This thesis aim is to provide the groundwork to entire control structure using ROS framework. Work described here focuses on the software system for one robot with a possibility to extend the software to incorporate several robots. The most interesting tasks arising from the project objectives that will be described in this thesis are:

- localization system based on detection of fiducials in the envirionment
- autonomous navigation to the charging station, docking, and undocking sequence
- following of objects equipped with ArUco marker
- creation of the map of the environment and automatic comparison of the created map with ground truth[1] map

3.1 Division of the Work on the Project

The project includes two theses. Tasks performed by the author are documented in this thesis, tasks completed by Bc. Roman Adámek are described in his thesis [2]. Summary of Adámek's work:

- design of mechanical and electrical parts of the robot
- design of charging station
- programming GUI for manual robot control
- implementation of AR elements into GUI
- programming robot's microcontroller - PI motor controller, reading data from sensors
- programming microcontroller in the charging station
- creation of some of the entertaining or educational mini-games for users
- and other minor tasks

[1] Can be defined as the most accurate measurement that is available

4 Simulation in Gazebo

4.1 Creation of Simulation Environment

As the first step in the simulation solution, the environment that will resemble the intended setting in the museum had to be modelled. The environment is modelled by walls, moveable and static obstacles, ArUco markers, and few other parts for visualization purposes. Walls, doors, and windows were created in *Building Editor* in Gazebo (fig. 4.1) which is quite intuitive and therefore no description of the process is given here.

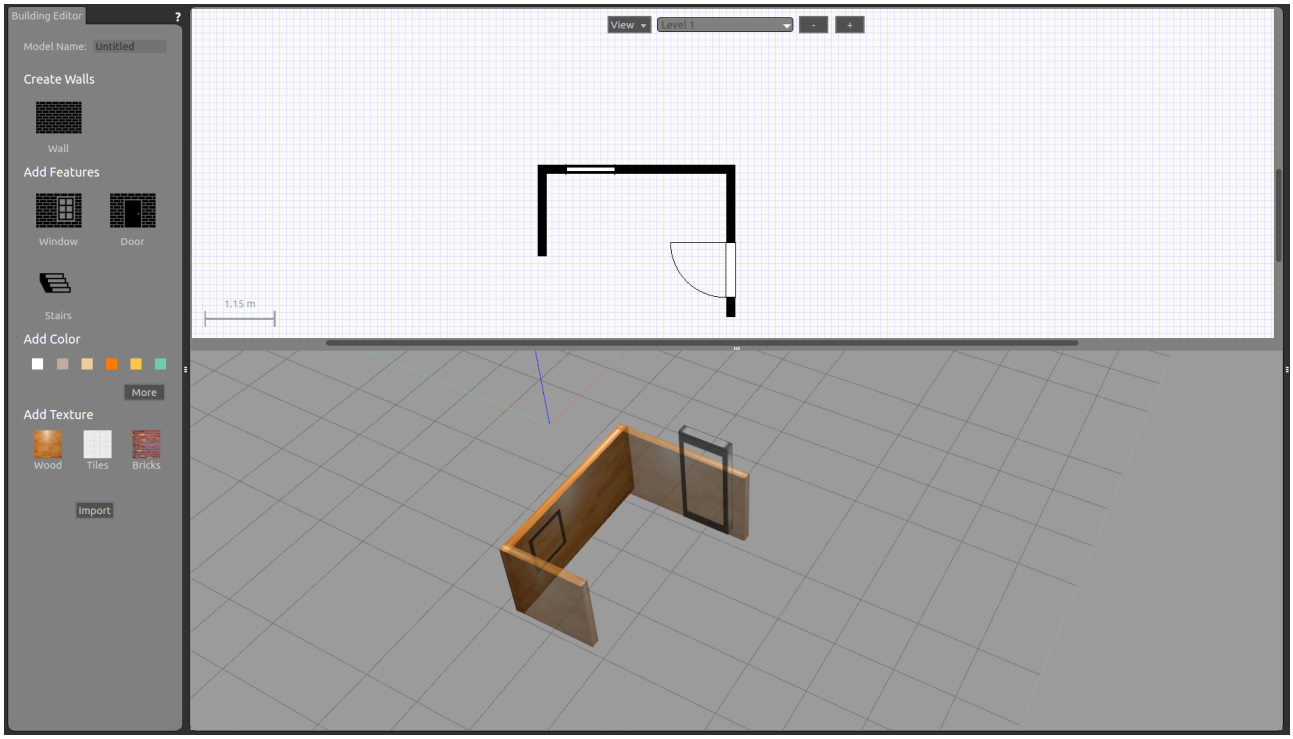


Figure 4.1: Gazebo *Building Editor*

Moveable and stationary objects were added from The Model Database Repository through GUI. Models are downloaded and then stored locally. Each object has to follow a specific structure to behave correctly in Gazebo. Meta information about the model are stored in *model*.config, Simulator Description Format (more about SDF in [50]) of the model is stored in *model*.sdf. Optional files are in Meshes Directory that contains COLLADA and/or STL model files, in Material Directory are textures and in Plugins Directory where plugins for the model are stored. In fig. 4.2 complete simulation environment with the robot is depicted.

File template presented above was used when adding ArUco markers in the simulation. Due to the time complexity of creating more different markers, the script from GitHub repository [51] under the Apache 2.0 License was used to create 20 markers from given images. Then markers were placed on the wall to cover the entire environment approximately evenly.

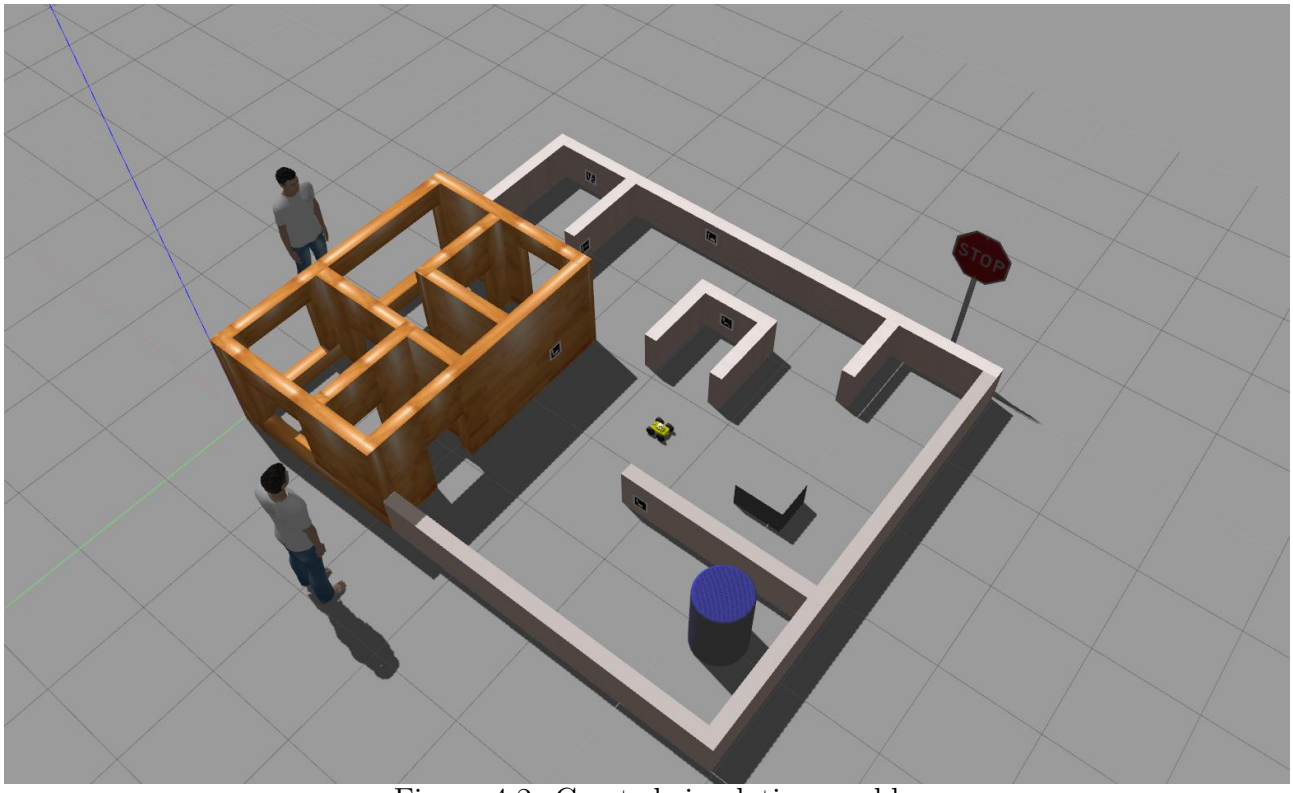


Figure 4.2: Created simulation world

As concerning the settings of Gazebo simulation, the default physics engine ODE was used with *quickstep* physics constraint solver with 70 iterations to converge. *Real time update rate* alongside with *max_step_size* was set to 1000, respectively 0.001 to resemble real-world time. The friction model was set to *cone_model* because of the faulty implementation of *pyramid_model* in Gazebo version 7. For the explanation of presented parameters and many others refer to documentation [52].

4.2 Robot Modelling

Robots in ROS are modelled in The Universal Robotic Description Format (URDF) which is XML file format. URDF model is composed of two elements: links and joints that connect links. Brief description of modelling robot parts is in 4.2.1. For speeding up of tedious writing in URDF language serves XML macro (Xacro) language. Xacro enables to define repeating properties and create own macros. Example of using macros to add a wheel to base link and to add IMU sensor to the robot:

```
<!-- Add wheel  -->
<xacro:rubber_wheel prefix="front_right" parent="$(arg prefix)base_link" reflect="false">
    <origin xyz="${wheel_offset_x} -${wheel_offset_y} ${wheel_offset_z}" rpy="0 0 0" />
</xacro:rubber_wheel>
<!-- Import IMU -->
<xacro:sensor_imu_hector prefix="imu" parent="$(arg prefix)base_link">
    <origin xyz="0 0.0 0.055" rpy="0 0 0" />
</xacro:sensor_imu_hector>
```

Figure 4.3: ArUco markers in detail

For converting Xacro to URDF serves ROS package *xacro*. For additional information on URDF and Xacro usage refer to tutorials and documentation [53], [54]. Complete information on modelling robot in ROS could also be found in the thesis [55].

4.2.1 Robot Parts Modelling

Links

Links in URDF represents a rigid body. Each link consists of several parts in terms of properties that are divided into visual, collision, and inertial sections.

Here follows a brief description of link elements with the example from one of the links used in modelling robot for this thesis.

Visual aspect of bodies can be represented using geometric primitives or mesh in *.stl* or *.dae* format. Origin defines the transformation of coordinate system from joint. STL files for parts were taken from [2].

```
<visual>
    <origin xyz="0.011 -0.060276 0" rpy="${PI/2} 0 ${PI/2}" />
    <geometry>
        <mesh filename="package://mech_ros/robot_description/meshes/
        Chassis.stl" scale="${Scale}" />
    </geometry>
    <material name="gray_color" />
</visual>
```

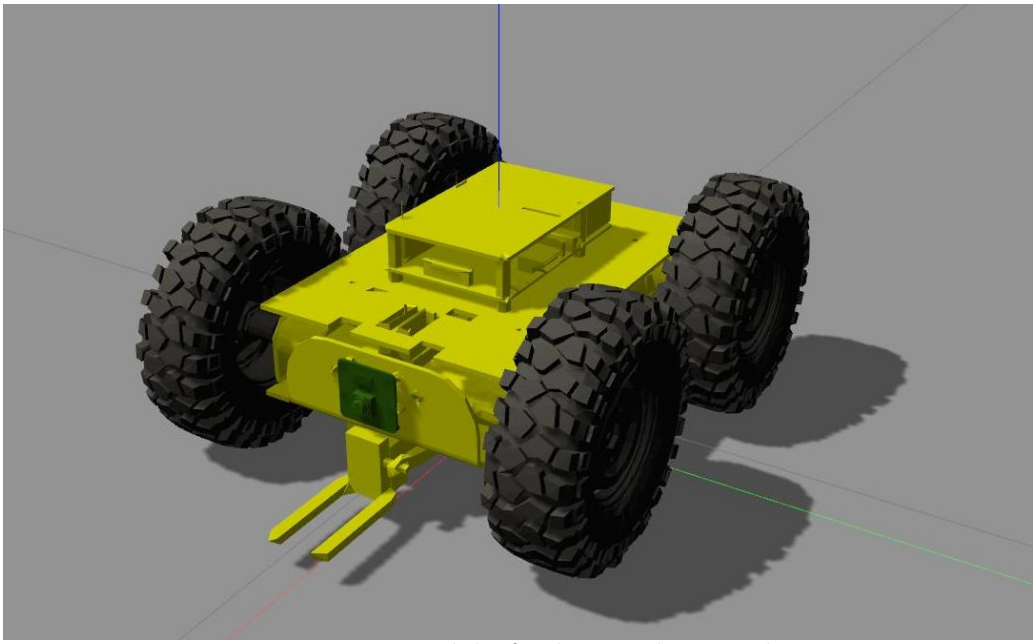


Figure 4.4: Model of robot without Lidar

Inertial element composes from the origin of the center of the mass, mass, and Inertia tensor. For root link (the link that has no parent) inertia element cannot be modelled. Instead, additional dummy link named *Inertia* with root link as a parent is placed to the mass center of root link. Note that in this case for *Inertia* link that resembles inertia element of *base_link*, non-diagonal elements of tensor were computed as having a significantly lower value than diagonal elements, and therefore, were set to zero.

```
<inertial>
    <origin xyz="0 0 0.02" />
    <mass value="1.172" />
    <inertia ixx="0.001568" ixy="0.0" ixz="0.0" iyy="0.003775" iyz="0.0" izz="0.00483" />
</inertial>
```

Collision element defines the geometric shape that is considered when creating contacts in Gazebo. Due to the computational cost of collision detection, it is recommended to model collision geometry with geometric primitives. Here the mesh of *base_link* is replaced with the block.

```
<collision>
    <origin xyz="0.008 0 0.023" rpy="0 0 0" />
    <geometry>
        <box size="0.242 0.11 0.085" />
    </geometry>
</collision>
```

Gazebo element provides additional information for gazebo simulation. Here only the surface from gazebo element is needed for the wheel model. The coefficients values that define frictions were obtained from trial and error method:

```
<surface>
    <friction>
        <ode>
            <mu>1.0</mu>
            <mu2>1.0</mu2>
            <slip1>0.5</slip1>
            <slip2>0.0</slip2>
            <fdir1 value="1 0 0"/>
        </ode>
    </friction>
</surface>
```

Joints

Joints bind parental and child links. Each link can have only one parent. The origin of the joint is the transformation from the origin of the parental link. There exists 6 kinds of joint: revolute, continuous, prismatic, fixed, floating, and planar. Only continuous for rotating without angle limits, and fixed for tight binding of links were used.

For, as will be described below, wheel continuous joints are controlled with the velocity controller, joints can be modelled as follows:

```
<joint name="${prefix}_wheel_joint" type="continuous">
      <parent link="${parent}"/>
      <child link="${prefix}_wheel"/>
      <xacro:insert_block name="origin" />
      <axis xyz="0 1 0" rpy="0 0 0" />
      <limit effort="1000" velocity="1000" />
      <joint_properties damping="0.0" friction="0.0"/>
</joint>
```

Transmission element must be added to URDF file to control joints from ROS:

```
<transmission name="${prefix}_wheel_joint_trans">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="${prefix}_wheel_joint">
        <hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
    </joint>
    <actuator name="${prefix}_wheel_joint_motor">
        <hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>
    </actuator>
</transmission>
```

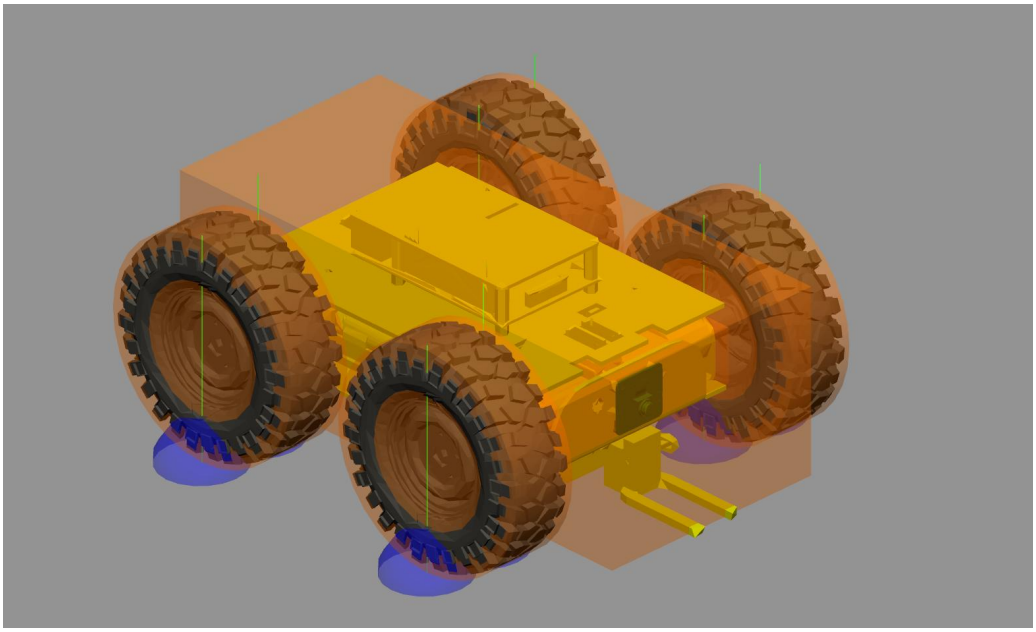


Figure 4.5: Collision geometry and contacts

Final model

Robot model consists of four wheels, base link, IMU link, Camera link, and Lidar link. Transformations between parts can be seen in fig. 4.6. As an extension, other links and joint could be added for modelling the lifting mechanism.

4.2.2 Sensors and Control

As was stated in 2.2, Gazebo can simulate sensors through plugins. Gazebo element must be added to URDF file and either bind with specified link as Gazebo reference or in some plugins specified separately. Here follows gazebo element for IMU (magnetometer is simulated separately) where parameter's meaning should be self-explanatory:

```
<gazebo>
    <plugin name="${prefix}_controller" filename="libhector_gazebo_ros_imu.so">
        <alwaysOn>true</alwaysOn>
        <topicName>${prefix}/data_raw</topicName>
        <bodyName>${prefix}_link</bodyName>
        <frameId>${prefix}_link</frameId>
        <updateRate>50.0</updateRate>
        <gaussianNoise>0.01</gaussianNoise>
        <rateOffset>0.0 0.0 0.0</rateOffset>
        <rateDrift>0.005 0.005 0.005</rateDrift>
        <rateGaussianNoise>0.005 0.005 0.005</rateGaussianNoise>
        <accelOffset>0.0 0.0 0.0</accelOffset>
        <accelDrift>0.005 0.005 0.005</accelDrift>
        <accelGaussianNoise>0.05 0.05 0.05</accelGaussianNoise>
    </plugin>
</gazebo>
```

As can be seen above, plugin allows specifying sensor frame, topic name, frequency, individual noise components, and others. Plugins for IMU, magnetometer, camera, and Lidar are used. Wheel encoders are not simulated instead velocity measurement is obtained from ROS package *diff_drive_controller* that controls wheel velocities through *hardwareInterface/VelocityJointInterface* defined in transmission element of wheels and supports skid steer robots by controlling the same side wheel to equal velocity.

4.3 Integration with ROS

After the previous section, everything is set up for simulating the robot using ROS in Gazebo. The procedure for complete simulation is as follows: Gazebo is started with created world as an argument, then the robot is converted from Xacro to URDF using package *robot_description* and then inserted in the simulation through node *spawn_model* in package *gazebo_ros*. For broadcasting transformations of individual robot parts, the node *robot_state_publisher* in the same named package is used.

Most algorithms were at first tested in Gazebo and then implemented on real hardware. Launch files, files which can start many different nodes, were kept separately for simulation and real target even though they could be written in a way that changing parameter would result in launching either simulation or either real target.

4.3.1 Localization using Extended Kalman Filters

As was noted in 2.3.1, two localization sources have to broadcast transformations to assure that *base_link* pose is correctly defined with respect to the *map* frame. Two nodes of EKF from *robot_localization* package were used for that purpose. One that broadcast transformation from *map* to *odom* was named *ekf_abs* and the one broadcasting transformation from *odom* to *base_link* frame was named *ekf_rel*. Transformation tree for all frames is depicted in fig. 4.6

ekf_rel

As inputs to *ekf_rel* serves the following:

- linear and angular velocity of the robot from *diff_drive_controller* (that is the difference from real robot implementation, details will be given further) in the form of *nav_msgs/Odometry*
- angular velocity and linear acceleration from IMU in the form of *sensor_msgs/Imu*
- orientation of the robot with respect to *map* frame in the form of *sensor_msgs/Imu* - strictly speaking that should belong only to absolute localization, but Madgwick algorithm is iterative (for Madgwick one iteration equals one measurement sample) and therefore there should not be big jumps in estimated orientation

As the robot moves only in one plane, *two_d_mode* was set to true. Because on start, uncertainty in relative localization is zero, elements in *initial_estimate_covariance* were set to a minimal number. Parameter *dynamic_process_noise_covariance* dynamically scales the *process_noise_covariance* based on the magnitude of the robot's velocity. This setting should ensure that the covariance of the estimated pose does not grow when the robot is stationary.

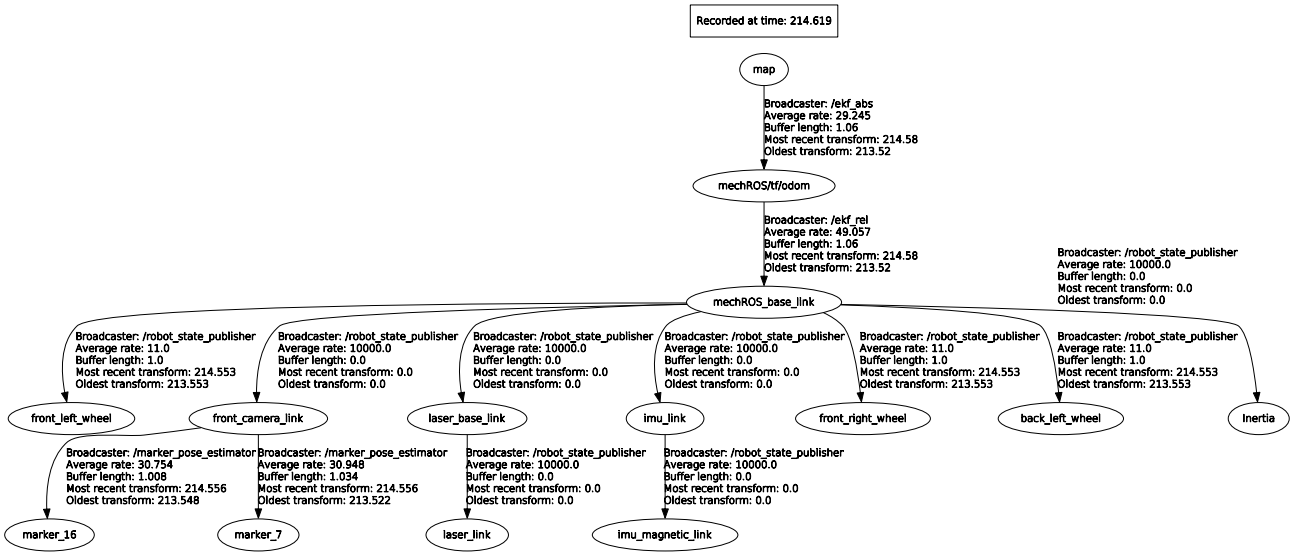


Figure 4.6: Transformations tree: *ekf_abs* broadcast transformation from *map* to *odom*, *ekf_rel* publish *odom* to *base_link* transformation, *marker_pose_estimator* (is furthered discussed in 5.2.3) publish transform from camera to the marker and transformations between robot parts are broadcasted by *robot_state_publisher*. Note that *back_right_wheel* is not shown here for sizing reason.

Another setting that could improve pose estimation is the *use_control* parameter. As was noted in 2.3.2, predicted linear acceleration and angular velocity of the robot in the prediction step of EKF have the same value as in the previous step. That inherently introduces lag in state vector variables. Using control command for predicting acceleration could overcome this problem. While in navigation mode, the controller sends velocity command at a maximal rate of 10 *Hz* and sometimes even slower, this settings does not have big impact on performance. Its significance could be increased in manual control. Settings for using control term:

```
<!-- Use control to predict acceleration -->
<param name="use_control" value="true"/>
<param name="stamped_control" value="false"/>
<param name="control_timeout" value="0.03"/>
<rosparam param="control_config">[true,  false, false,
                                  false, false, true]</rosparam>
<rosparam param="acceleration_limits">[0.4, 0.0, 0.0,
                                       0.0, 0.0, 5.0]</rosparam>
<rosparam param="decceleration_limits">[1.0, 0.0, 0.0,
                                        0.0, 0.0, 7.5]</rosparam>
```

Acceleration and deceleration of real robot were obtained from measuring translational and angular velocity from wheel encoders and IMU as a response to step in desired robot velocity. Computed parameters were used for both simulated and real robot.

Another solution for the reduction of lagging of filtered output is to increase corresponding elements in *process_noise_covariance* at the cost of inflating noise from sensors presented in filtered output.

ekf_abs

The *ekf_abs* subscribes to inputs:

- same inputs as *ekf_rel*
- estimation of a pose in *map* frame from processing image from the camera - How to get an estimation from ArUco markers detection will be described in detail in 5.2.3.

The same settings as for *ekf_rel* does apply to *ekf_abs* except for the *initial_estimate_covariance*. Diagonal elements of the mentioned matrix must be big enough that when starting robot in arbitrary position in the map, upon seeing the first ArUco marker, the robot is properly localized.

With the current value of the covariance matrix of pose estimation is parameter *xxxx_rejection_threshold* related. For every part of the sensor measurement (replace *xxxx* in the previous parameter), rejection value based on Mahalanobis distance, defined in 2.3.2, can be set. Here it is utilized for rejection of pose estimation from ArUco marker measurement because occasionally, the id of marker could be wrongly determined, and therefore, without rejecting it, large error would be inserted in the filter. The parameter value is based on χ_p^2 distribution for 3 DOF. For the selected p-value 0.05, value from any table describing χ_p^2 distribution can be obtained as 7.815.

As processing image can take some time, pose estimation from ArUco markers may be delayed. Therefore parameter *smooth_lagged_data* was set to true. Reverting filter time was described in 2.3.2. Parameter *history_length* specifies how long are the data stored for reverting state. Based on the observation of processing the image on Raspberry Pi, the parameter was set to 0.2 *s*. For simulation, it is sufficient to set the parameter to lower value.

This settings for localization are true provided the map of the environment alongside the map of markers are already created.

Data from IMU follow REP-145, described in 2.3.1, naming convention: *imu/data_raw* from IMU are together with magnetometer measurement *imu/mag* published to *imu_filter_madgwick* where orientation is estimated and alongside linear acceleration and angular velocity published as *imu/data*.

Parameters (namely gain β and ζ) for node *imu_filter_madgwick* were set according to recommendations in 2.3.2 and then modified by trial and error method.

Differences in positional outcomes from *ekf_abs*, *ekf_rel*, and pose estimated from image processing, are shown in Appendix in fig. 6.1.

4.3.2 SLAM using Gmapping

To successful set-up of the navigation stack, map needs to be created. Then it is used in *global_costmap* and could also be utilized as a static layer in *local_costmap*.

Gmapping needs to subscribe to laser scan measurement and transformation from *odom* to *base_link*. For that purpose *ekf_rel* node from 4.3.1 was used. As an additional input to *ekf_rel* served pose estimation from *laser_scan_matcher*. For speeding up convergence, IMU readings were set as inputs to *laser_scan_matcher*. Most of the *Gmapping* and *laser_scan_matcher* parameters were left at their default values. Some modifications were made to them while performing SLAM on the real robot (maps in 5.3) but not here nor there any explanation of parameters is given because the theory behind these packages would have to be described, which is out of the scope of this thesis.

With these settings, *Gmapping* was launched alongside other nodes and the robot was controlled through *rqt_gui* to drive around the environment. The resulting map can be seen in fig. 4.7.

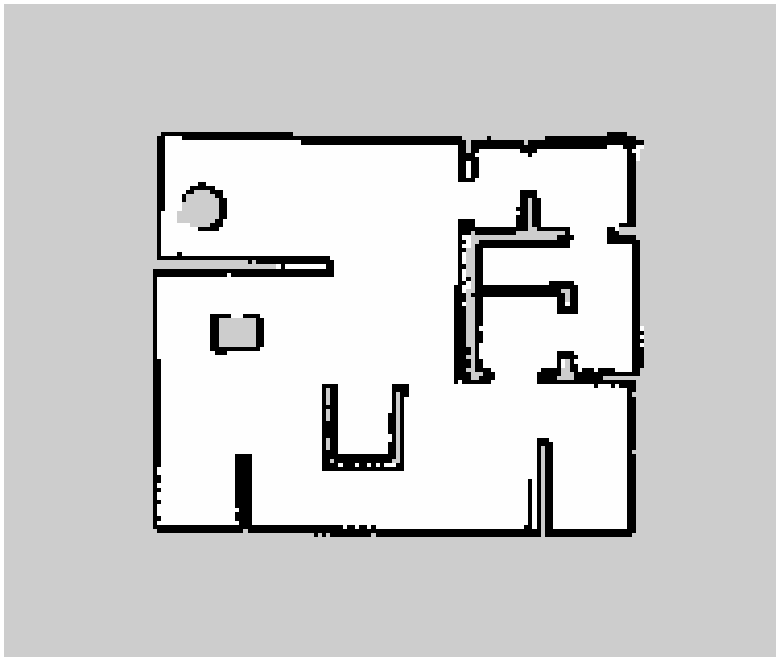


Figure 4.7: Created map of simulation environment: White color represents free cells, black occupied cells, and gray unknown status of cells.

4.3.3 Navigation using Move Base Flex

As a first introduction with navigation stack in ROS, *move_base* package was used. Subsequently for better customization and extensibility of navigation was then proceeded to *move_base_flex*. As reported in 2.3.4, numerous things must be correctly set for proper navigation. As default values for most parameters for setting navigation stack were used recommended values from *Tuning Guide* [27], and afterward, the parameters were adjusted for better behavior.

Planner Both A* and Djikstra algorithms, available in package *global_planner* for expanding cells to find a feasible path, were tried. Surprisingly, Djikstra algorithm provided better results, and therefore, was selected even at the cost of being slower (as used maps are relatively small, the difference in speed is not so significant) due to the expansion of higher amount of cells.

Controller DWA local planner has to know the kinematics and dynamics constraints of the robot in order to control the movement of the robot correctly. The maximal and minimal velocity of the real robot were measured and also used for simulations. The problem arising from the used skid steer platform is that rotation is achieved through different angular velocities of the same side wheels. Therefore maximal translational and rotational velocities cannot be reached simultaneously. As a solution, both maximal values of translational and rotational velocities, were lowered from their actual measured values. Another solution is implemented on the real robot – when the angular velocity of the wheel calculated from the inverse kinematics model, described in 5.2.1, could not be reached, both rotational and translational velocity of the robot is lowered by scale to preserve radius of the desired path arc.

```
dwa:
# Selected robot kinematics parameters
    max_vel_x: 0.15 # measured 0.24
    min_vel_x: -0.15 # measured -0.24
    max_trans_vel: 0.15 # measured 0.24
    min_trans_vel: 0.05 # measured 0.05
    max_rot_vel: 1.3 # measured 1.8
    min_rot_vel: 0.3 # measured 0.3
```

Measurement of the robot's acceleration and deceleration was already described in 4.3.1. Parameters $bias_p$, $bias_g$ and *scale* (here slightly renamed for better readability) were tuned empirically.

Global and Local map Since the robot does only have the monocular camera for sensing outer world and reliable detection of obstacles from this type of sensor is a task beyond the time possibilities of this thesis, both *global_costmap*, and *local_costmap* were set to static map. If the robot would be equipped with Lidar (as one robot is intended to have) then in obstacle layer would be Lidar set as the source for marking and clearing obstacles. Inflation layer has the same settings for both maps. Parameters *radius* and *scale* for inflation layer were tuned empirically. Local map is set to rolling window. Its center remains in *base_link* and moves together with robot.

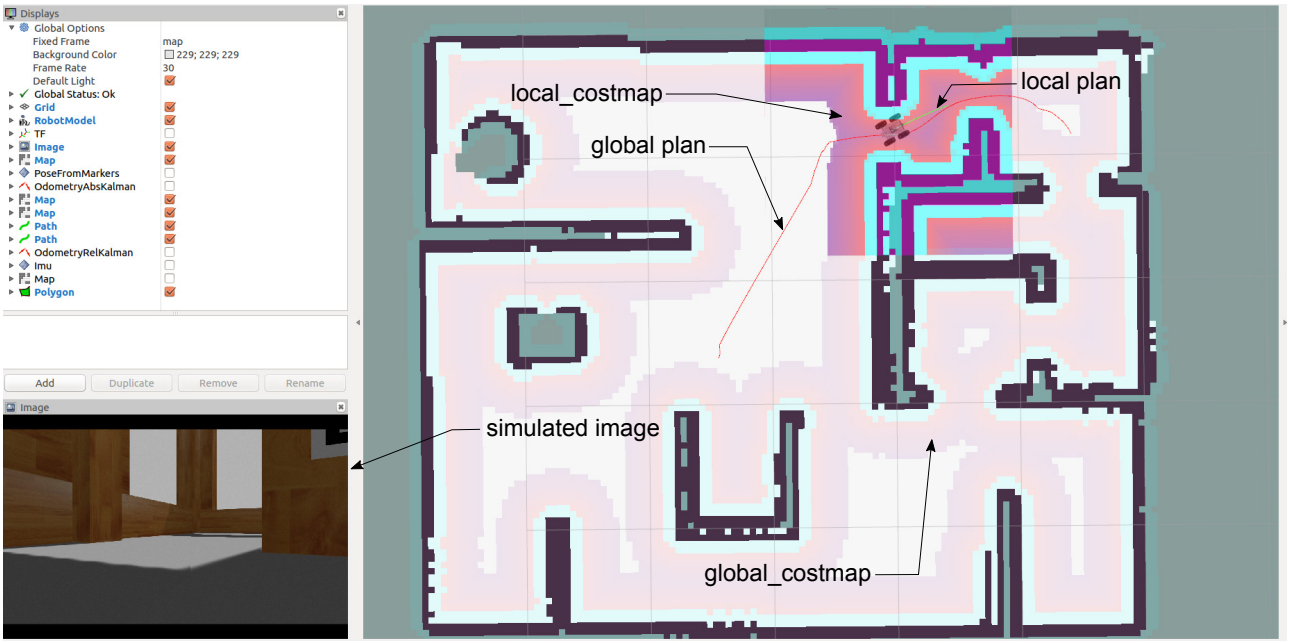


Figure 4.8: Navigation in the simulation environment: Image from ROS visualization tool - Rviz where some parts of the navigation system can be seen.

SMACH To create a simple navigation SMACH, the example from tutorials was used. Its functionality resembles the behavior of *move_base* with the following behavior:

1. *WAIT_FOR_GOAL*: waiting for message *geometry_msgs/PoseStamped* with coordinates of goal position – goal coordinates are passed to *GET_PATH* through userdata

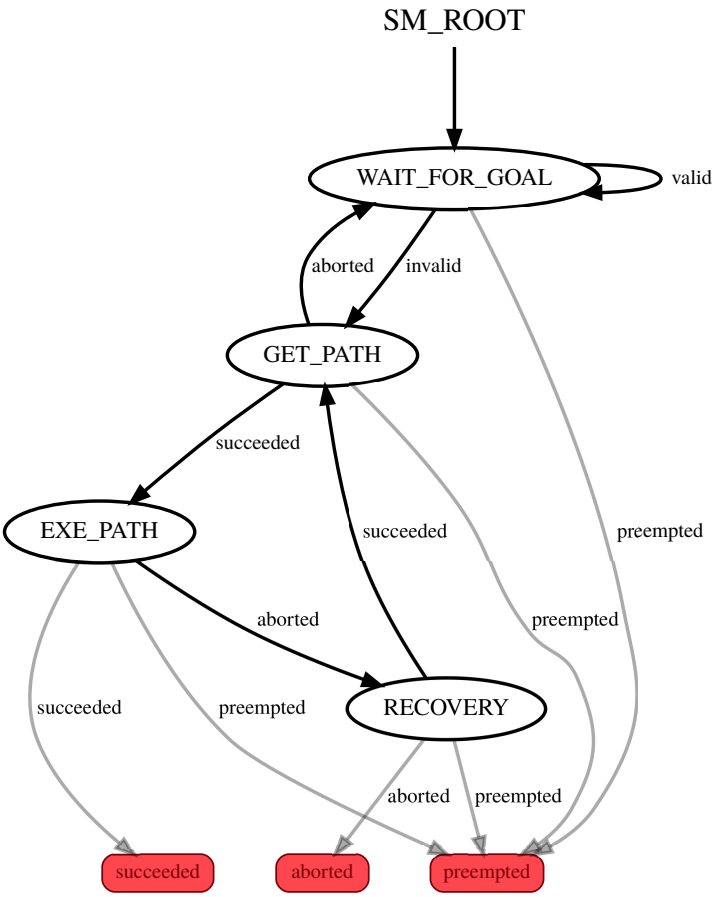


Figure 4.9: Basic SMACH for navigation

2. *GET_PATH*: planning path to goal point from current position – planned path is passed through userdata to the controller

3. *EXE_PATH*: executing the movement to goal point - controller sends velocity commands that robot subscribes to. If the robot gets stuck, *RECOVERY* state becomes active with the outcome from the controller that specifies current status. If successful, state machine exits with outcome *succeeded*

4. *RECOVERY*: Based on userdata, desired recovery behavior is executed. If successful, *GET_PATH* state become active.

When continuous navigation loop is desired, the successful outcome from *EXE_PATH* is connected to *WAIT_FOR_GOAL* and this loop runs until state machine is preempted by command or *RECOVERY* is aborted.

The most significant advantage of this approach to navigation showed in fig. 4.9 is that multiple different planners, controllers and recovery behaviors can be loaded at the start and then used as desired, according to the given situation.

Recovery behaviors One of the userdata in SMACH *Navigation* is *recovery_flag*. This flag is used for choosing recovery behavior. As the first attempt to get the robot unstuck, *moveback_recovery* is used. If this attempt is not successful, the robot slowly rotates to relocalize itself by using *rotate_recovery*.

5 Implementation on Real Robot

Implementation on the real robot was performed concurrently with the simulation. New algorithm or procedure was first tested in the simulation, then on the real robot and robot's parameters in the simulation were then modified to resemble the real robot more closely.

5.1 Brief Description of Used Hardware

As was characterized in 3.1, the design of mechanical and electrical parts of robot alongside the charging station was realized in the thesis by *Adámek* [2]. Therefore only necessary information about hardware devices and their connections is illustrated here.
Working station:

- desktop computer with 4-core 3.6 *GHz* Intel i3-8100, 8 GB RAM and Radeon RX 550 with Ubuntu 16.04 as the operating system.

Robot:

- Raspberry Pi 3B+ (further referred to as RPi): single-board computer with 1.4 *GHz* quad-core processor and 1GB SDRAM. Available interfaces are Wi-Fi, Camera Serial Interface, USB, I^2C, SPI, UART, and others. Image based on Ubuntu 16.04 with pre-installed ROS was downloaded from [56] and installed.
- Hardware Attached on Top (HAT) board for RPi designed in [2] with ATSAMD21J18A (further referred to as SAMD21) from Atmel supported in Arduino platform. Available interfaces are USB, I^2C, SPI and UART.

Charging station:

- same RPi as on the robot
- Arduino ProMicro with 8bit ATmega328P. I^2C interface was used for communication.

Schematic in fig. 5.1 shows the communication protocol between used hardware and describes in bullet points what is running on each hardware.

Processing image is performed either on the RPi on the robot or either on the Working station. For autonomous movement, it is necessary to not introduced additional lag from transmitting video over Wi-Fi. Delay of image over Wi-Fi ranged mostly between $0.25 - 0.3$ s but sometimes even longer delay was measured. Various options for transmitting video were tried but neither showed consistent lag beneath 0.2 s or consistent image quality.

When the robot is controlled manually, the video must be transmitted anyway, and therefore, the image processing takes place on Working station.

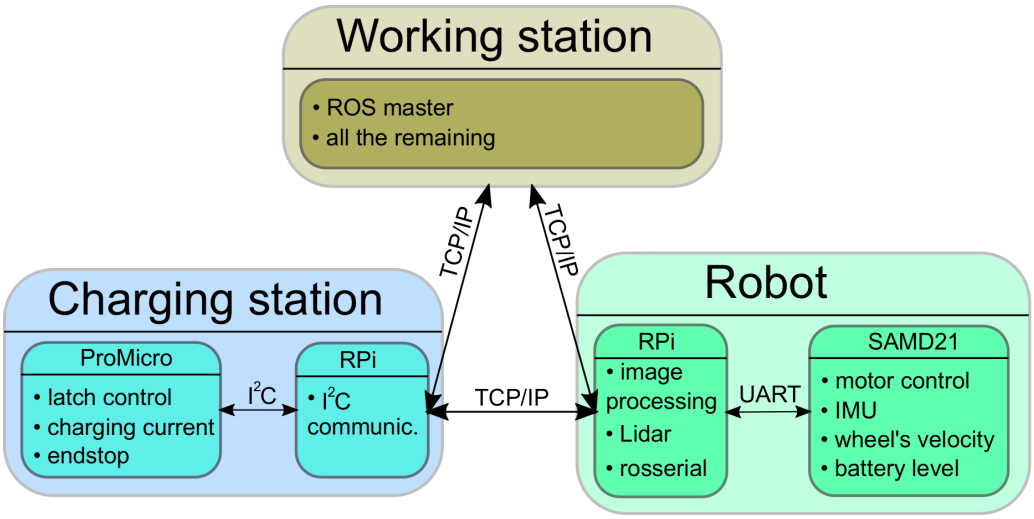


Figure 5.1: Schematic of used hardware: Communication between hardware is shown. If name a of sensor is given, that means reading data from the sensor and publishing them either directly to ROS network or indirectly through UART or I^2C. For example, Lidar means reading data from Lidar and publishing them to ROS network

5.2 Experiments with Sensors

In order to evaluate settings of parameters of specific packages or to estimate unknown coefficients, it is necessary to have ground truth data for comparison. As a method to obtain ground truth data was selected pose estimation of ChArUco board from the camera image. Logitech C922 camera was placed under the ceiling and aimed in such way that it was focusing the floor. There was a trade-off between accurately determining ArUco markers on ChArUco board and between capturing as much area as possible in the camera image. The camera was firstly calibrated using another ChArUco board with more ArUco markers. Then ChArUco board was placed on the robot (fig. 5.2), and data (2D pose) were obtained for each experiment described below.

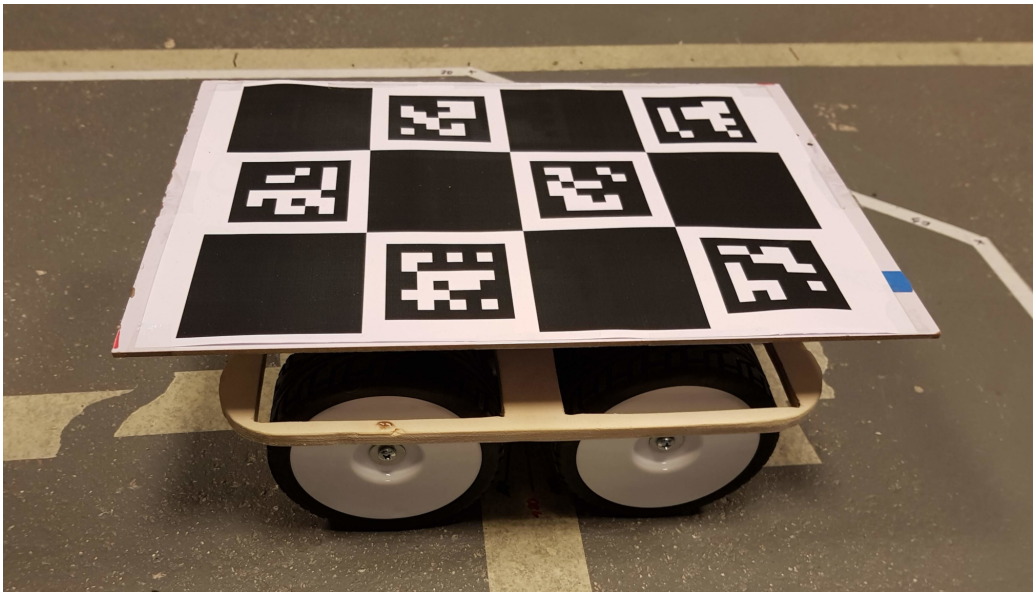


Figure 5.2: ChArUco board mounted on robot

5.2.1 Kinematics Parameter Estimation

Kinematics model was established in 2.4.4. Resulting equations are:

$$\begin{aligned} v_x &= \frac{\omega_r r + \omega_l r}{2} \\ v_y &= 0 \\ \omega_z &= \frac{r\left(\omega_r - \omega_l\right)}{B\chi} \end{aligned} \tag{5.1}$$

Four parameters had to be estimated: $r,\ B,\ a,\ b$ (χ is dependent on a, b). For estimating the parameters, 19 experiments were conducted for estimation and 4 for validation. In each, the robot was manually controlled by a joystick and drove with a different trajectory. Data from wheel encoders and board pose estimation from camera beneath ceiling were measured. After that, the kinematics model based on 5.2 with numerical integration of velocities to obtain 2D pose was created in Simulink. Following was to estimate parameters $r,\ B,\ a,\ b$ based on wheel's angular velocities as inputs to the kinematics model and pose from image processing as the measurement.

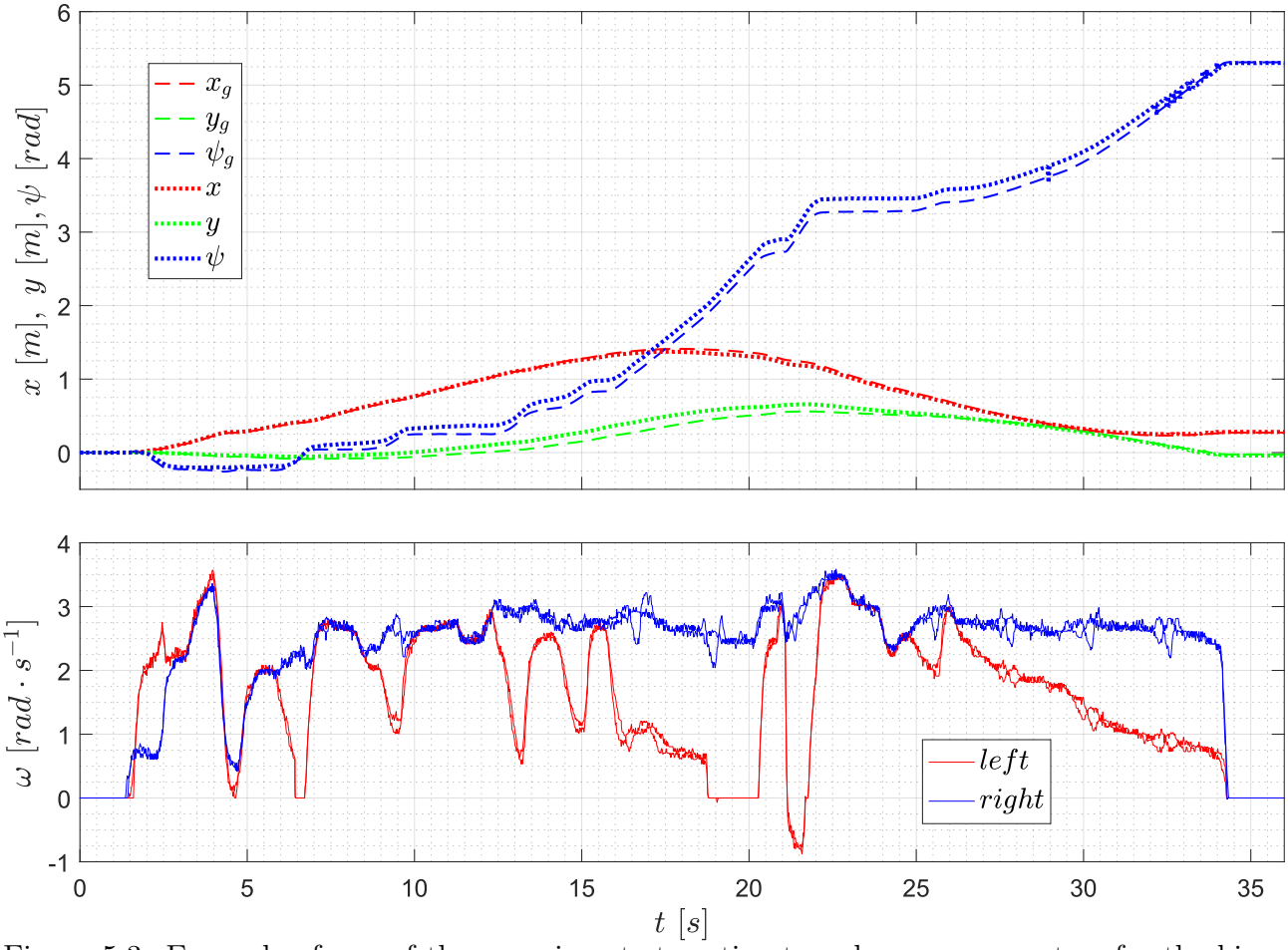


Figure 5.3: Example of one of the experiments to estimate unknown parameters for the kinematics model. Inputs to the simulation model are angular velocities of four wheels, two on each side.

As can be seen in fig. 5.3 and 6.2, pose from the kinematics model resembles measurement well enough. Figure in Appendix is from validation.

However, no easy way was found to obtain inverse kinematics model from forward model. Only an extremely complicated model with the help of symbolic solver that would be hard to implement. Therefore additional estimation with fixed $\chi = 1$ was performed and from the simplified model (it is the same model as for an ideal differential drive) was inverse kinematics model obtained as follows:

$$\begin{aligned} \omega_r &= \frac{2v_x + \omega_z B}{2r} \\ \omega_l &= \frac{2v_x - \omega_z B}{2r} \end{aligned} \tag{5.2}$$

Forward kinematics is implemented in script *Kinematics.py* that subscribes to created messages *mech_ros_msgs/WheelsVelocities* and converts them to *nav_msgs/Odometry*. Inverse kinematics is implemented in program *HAT_ROS_ALL.ino* running on SAMD21.

Parameter	Forward	Inverse
r	0.04561 m	0.04539 m
B	0.19007 m	0.23860 m
a	0.26484	–
b	0.10107	–

Table 5.1: Estimated parameters for forward and inverse kinematics

5.2.2 IMU Calibration and Usage

As the first step for IMU calibration, IMU was left still, and data from accelerometer and gyroscope were obtained. Elementary statistics analysis was carried out and from this were retrieved biases, variances, and gyroscope drift that served as first approximation for using IMU.

Magnetometer calibration was performed by rotating the robot in many different angles, measuring values from every axis and storing them. From these data transformation matrix and bias array was computed.

With the same settings as in the previous experiments, the robot was controlled to drive and camera under the ceiling was capturing the scene. The data from magnetometer were transmitted to *imu_filter_madgwick* where orientation was estimated. Parameters of the node were tuned so that the resulting orientation from Madgwick algorithm was as close as possible to ground truth orientation.

When moving, the robot is shaking quite a lot, and therefore, the measurement from the accelerometer is not usable for planar localization as the measurement was extremely noisy.

Data flow is as follows: Data from the accelerometer, gyroscope and magnetometer, are read in SAMD21 and transmitted through RPi to ROS network in the form of created message *mech_ros_msgs/RawImu*. Then they are processed in node from script *IMU_from_raw.py*. Readings from accelerometer and gyroscope are scaled, and the bias is subtracted from them. From the readings from the magnetometer, the bias is subtracted, and the result is multiplied by the transformation matrix. Processed data are then published in topic */imu/data_raw* and */imu/mag*. Calibration data for IMU are stored in *IMU_calibration.yaml* file and are loaded on

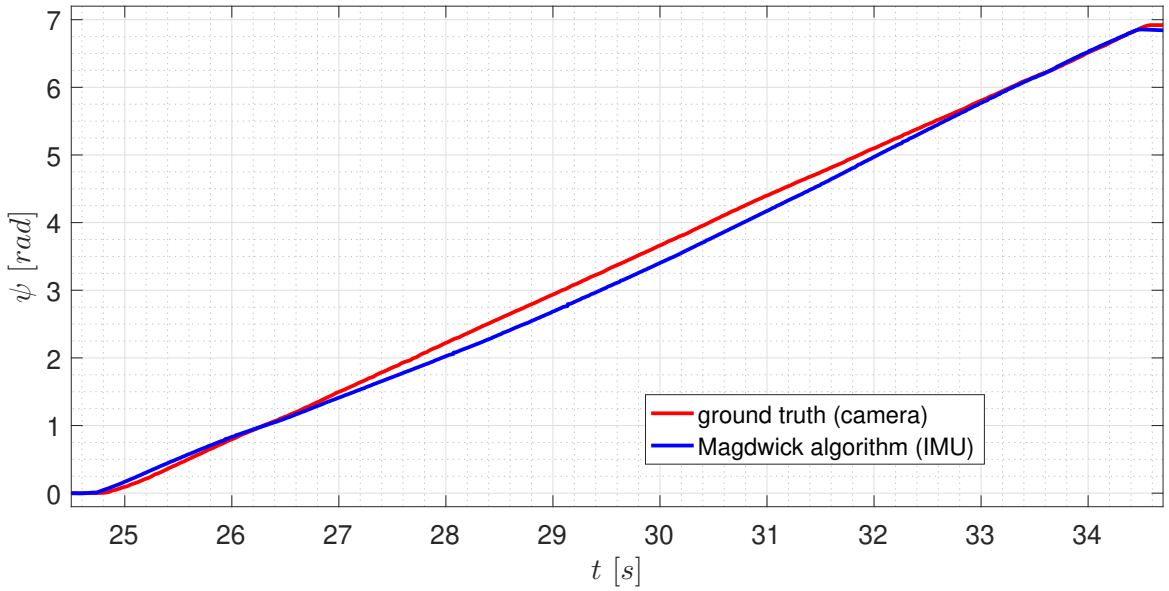


Figure 5.4: Experimental verification of Madgwick filter orientation estimate: From both measurements were their first measurement subtracted for better comparison

the start of the node. Accelerometer and gyroscope bias can be recalibrated optionally on start or/and while the program is running through created services *mech_ros_msgs/Start_recalibration* and *mech_ros_msgs/End_recalibration*.

5.2.3 Pose Estimation from Image Processing

In order to use ArUco marker pose estimation as input to *ekf_abs*, a model that converts readings to global frame (*map* frame) and assigns appropriate covariance matrix, must be established.

Accuracy testing

The procedure was as follow. RPi camera was calibrated using ChArUco board as a first step. Calibration data are stored in the *camera.yaml* file. To measure the accuracy of ArUco marker's pose estimation, the marker was attached alongside the ChArUco board to a panel as shown in fig. 5.6. Firstly only the estimation of the ChArUco board pose was tested, and the results showed that its accuracy is far better than the estimation of a single marker. Therefore pose of ChArUco board is taken as ground truth measurement even though some error is introduced to the analysis because of this assumption. Three types of measurement were performed. First was to move the panel 5 *cm* further from the camera in a longitudinal direction. 50 samples were taken for each distance and covariance of $\sigma^2_{a_x}$, $\sigma^2_{a_y}$, $\sigma^2_{a_\varphi}$ were computed. Meaning of a_x, a_y, a_φ is obvious from fig. 5.5. OpenCV function that estimates pose of markers returns translation and Rodrigues vector from the camera to marker. Rodrigues vector is converted to Euler angles and alongside translations is inverted to represent transformations in fig. 5.5.

Figure 5.7 shows that the pose estimation accuracy is greatly dependent on the marker's surface. Through regression, a dependence of variances $\sigma^2_{a_x}$, $\sigma^2_{a_y}$, $\sigma^2_{a_\varphi}$ on the surface was obtained as power function shown in eq. 5.3. The coefficients acquired by regression served only as a first approximation and had to be modified in a such way that variances of variables were increased. Main reasons for this inflation are the dependence of image quality on the ambient light condition and shaking of the camera when moving. Covariances were computed as

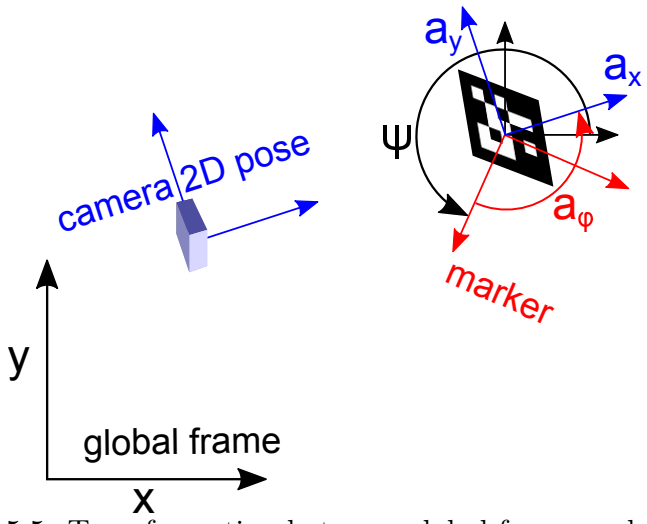


Figure 5.5: Transformation between global frame and camera

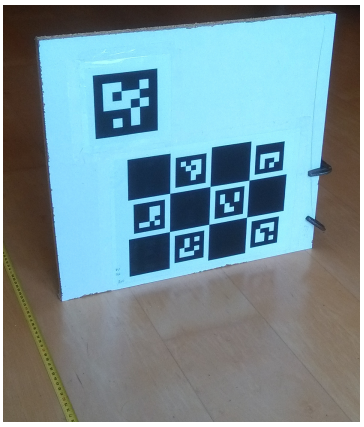


Figure 5.6: Measurement setup for ArUco marker pose estimation accuracy analysis

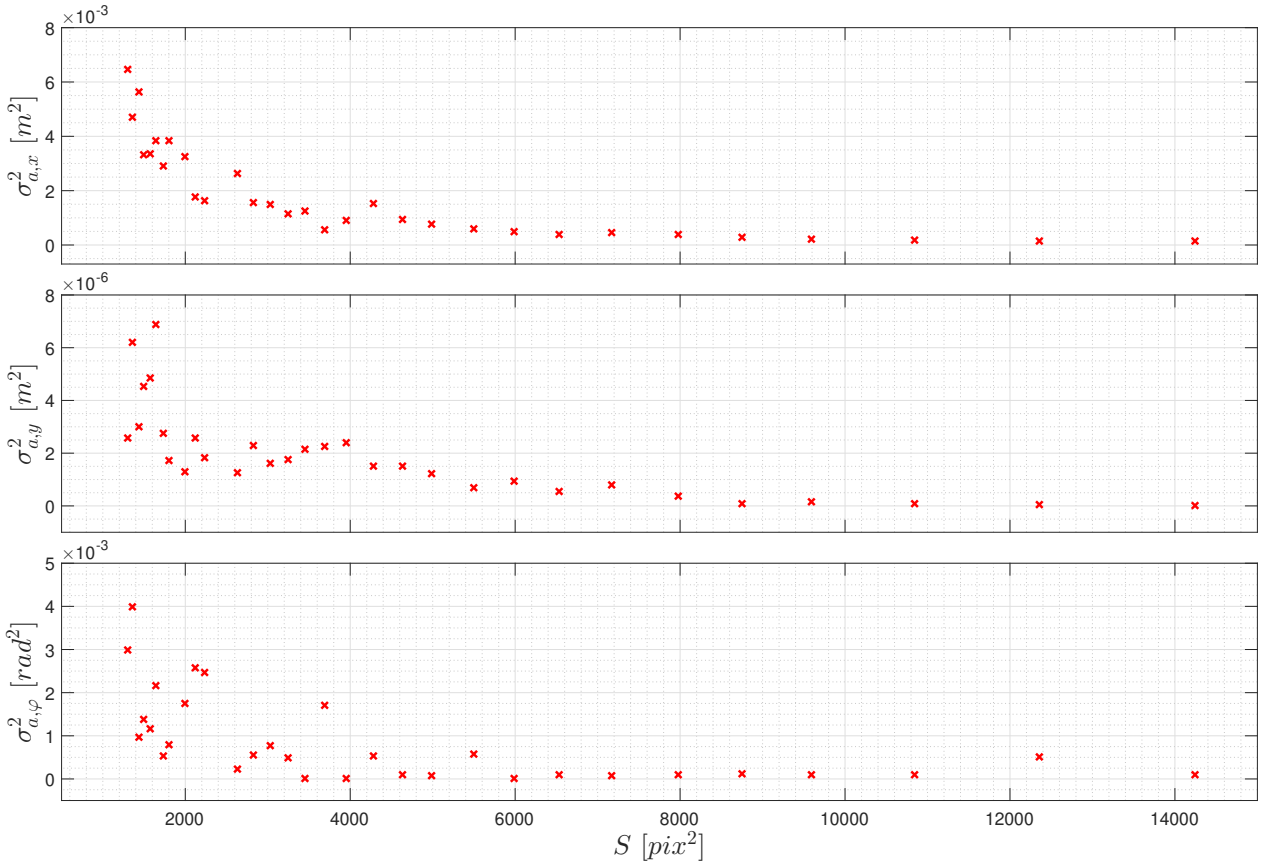


Figure 5.7: Dependence of measurement error variance on marker's surface: Here S stands for surface of marker in image bounded by its corner, and given in pixels squared

significantly smaller than variances and thus are neglected in further analysis.

$$\sigma_{ii}^2 = \frac{K_i}{S_{norm}^{A_i}} \tag{5.3}$$

where S_{norm} is normalized surface, K_i and A_i are coefficients, and σ_{ii}^2 is the variance of either $a_x,\ a_y,\ a_\varphi$.

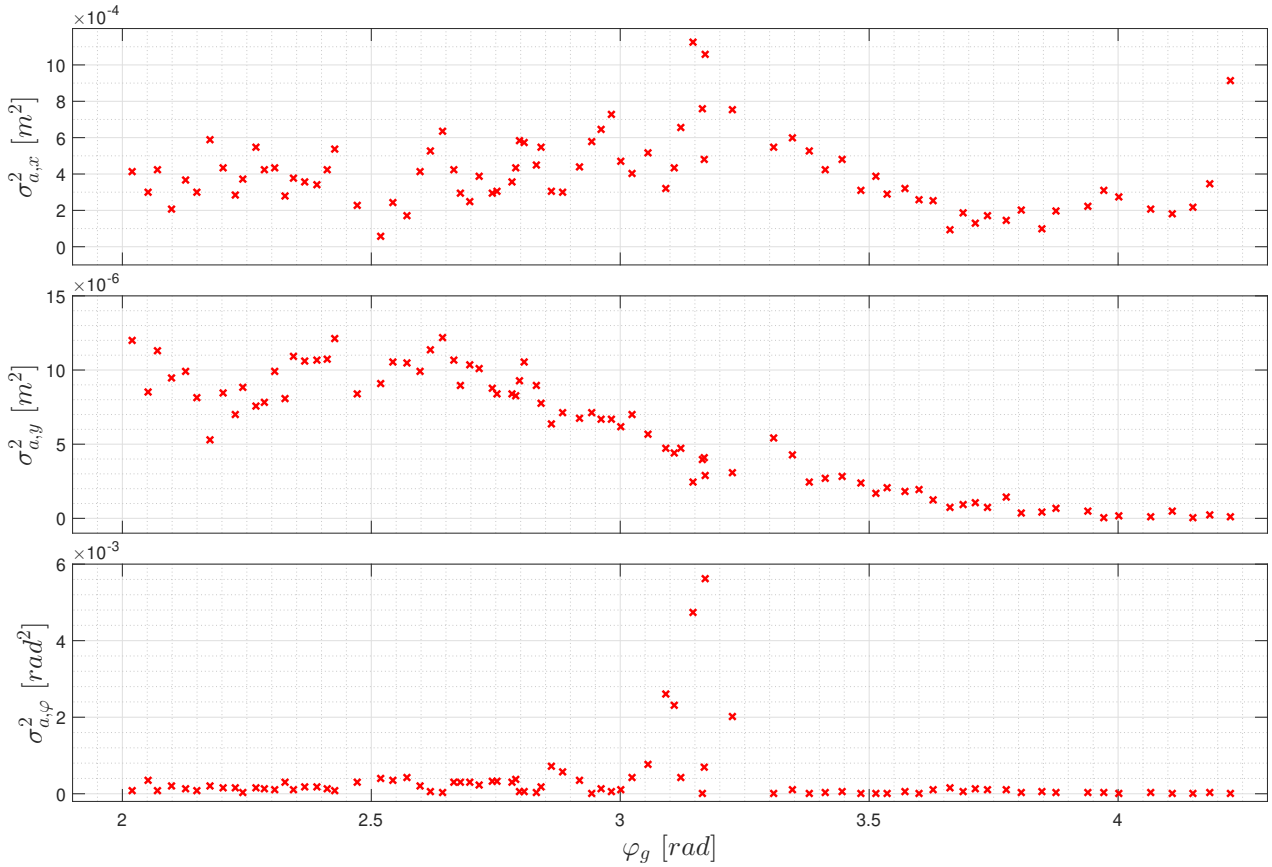


Figure 5.8: Dependence of measurement error variance on angle: Here φ_g stands for ground truth measurement of angle a_φ

Setting the panel at a constant distance and changing yaw (rotation angle about the global Z axis) by small angles was performed as the second experiment. Covariances were computed as in the previous experiment. In figure 5.8 is the dependence of variances on yaw shown. Dependence of $\sigma^2_{a_y}$ is excluded from further analysis as $\sigma^2_{a_y}$ is two orders of magnitude smaller than the other two and is probably influenced mainly by different light reflection on opposite sides. Variances of the remaining two variables are approximately the same for all angles except around π where the marker is parallel to the camera.

If variances are dependent on the surface and only little dependent on the yaw angle, the surface must be normalized with respect to yaw to give correct values. To account for higher variances around π, member $B\left(a_\varphi\right)$ was added to eq. 5.3. Function $B\left(a_\varphi\right)$ can be seen in fig. 5.9.

$$S_{\text{norm}} = B\left(a_\varphi\right) \frac{S}{\left|\cos\left(a_\varphi\right)\right|} \tag{5.4}$$

where S is the marker's surface in pixels^2 and S_{norm} is the normalized surface.

In the third type of experiment, the variance dependence on a lateral position in the image was tested. The panel was positioned at a constant longitudinal distance and shifted by 5 cm in a lateral direction. To make it more general, lateral distance is expressed as the angle from the optical axis in horizontal plane. No significant dependence on this angle was found. Figure can be seen in the Appendix.

This measurements and findings are quite consistent with experiments performed on April-

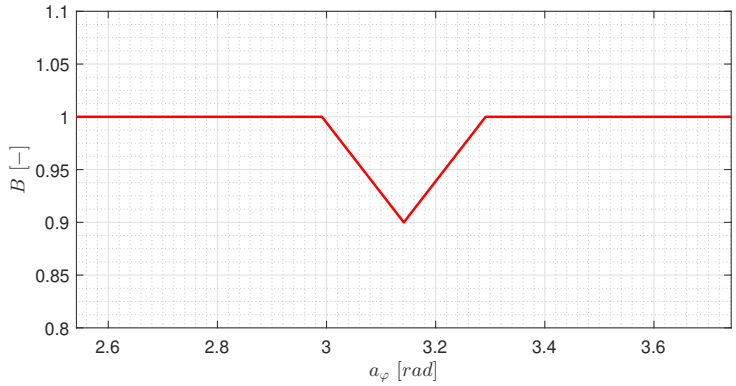


Figure 5.9: Function $B\left(a_\varphi\right)$ compensates for increase measurement error for a_φ angles near π value.

Tags markers (similar to ArUco marker) in Gazebo simulator by *Cerón*, 2017 [57].

Transformation to the global frame

Obtaining 2D pose of camera in the global frame is described by equation 5.5 and depicted in fig. 5.5.

$$\boldsymbol{a}' = f\left(\boldsymbol{a}, \boldsymbol{p_3}\right) \tag{5.5}$$

where $\boldsymbol{a}'$ is 2D pose in the global frame, $\boldsymbol{a}$ is pose in marker's frame, and $\boldsymbol{p_3}$ is transformation from global to marker's frame. This equation is rewritten in detailed form in 5.7.

$$\boldsymbol{a} = \begin{bmatrix} a_x & a_x & a_\varphi \end{bmatrix}^T; \quad \boldsymbol{p_3} = \begin{bmatrix} x & y & \psi \end{bmatrix}^T \tag{5.6}$$

$$\begin{aligned} a'_x &= a_x \cos\left(a_\varphi + \psi\right) - a_y \sin\left(a_\varphi + \psi\right) + x \\ a'_y &= a_x \sin\left(a_\varphi + \psi\right) + a_y \cos\left(a_\varphi + \psi\right) + y \\ a'_\varphi &= a_\varphi + \psi \end{aligned} \tag{5.7}$$

In order to compute the pose, $\boldsymbol{p_3}$ must be known. Therefore the position of the marker has to be measured and is stored in file *MechLAB_markers.yaml*. Due to the manual measurement of the marker's pose, uncertainty is introduced, and based on how well each marker's pose could be measured, diagonal elements σ^2_{xx}, σ^2_{yy}, $\sigma^2_{\psi\psi}$ of covariance matrix $\boldsymbol{\Sigma}_{p_3}$ are added to the file shown here:

```
frame_id : "map"
landmarks:
    9 :
        Translation: [4.995, -2.105, 0.0]
        RPY: [0.0, 0.0, 1.570796327]
        Covariances: [0.008, 0.008, 0.008, 0.005, 0.005, 0.005]
```

Measurement of each marker can be taken as a normal multivariate distribution $\boldsymbol{a} \sim$

$N(\boldsymbol{\mu}_a,\ \boldsymbol{\Sigma}_a)$ where $\boldsymbol{\mu}_a$ is the mean vector and $\boldsymbol{\Sigma}_a$ is the covariance matrix whose diagonal elements are computed according to equations 5.3 and 5.4.

For transforming a multivariate Gaussian distribution, first-order linearization of the transformation can be used. New distribution $\boldsymbol{y} \sim N(\boldsymbol{\mu}_y,\ \boldsymbol{\Sigma}_y)$ can be approximated as follows [58], here rewritten with use of such symbols to remain consistent in the whole thesis:

$$\boldsymbol{\mu}_y = f(\boldsymbol{\mu}_x) \tag{5.8}$$

$$\boldsymbol{\Sigma}_y = \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \boldsymbol{\Sigma}_x \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}^T \tag{5.9}$$

where $\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}$ is a matrix of first-order partial derivates, also called Jacobian.

Assuming that both distributions of $\boldsymbol{a}$ and $\boldsymbol{p_3}$ are independent, covariance matrix $\boldsymbol{\Sigma}_{a'}$ of 2D pose of camera in the global coordinate system can be computed as follows [58] (here also some symbols are rewritten):

$$\boldsymbol{\Sigma}_{a'} = \frac{\partial f(\boldsymbol{a}, \boldsymbol{p_3})}{\partial \boldsymbol{a}} \boldsymbol{\Sigma}_a \frac{\partial f(\boldsymbol{a}, \boldsymbol{p_3})}{\partial \boldsymbol{a}}^T + \frac{\partial f(\boldsymbol{a}, \boldsymbol{p_3})}{\partial \boldsymbol{p_3}} \boldsymbol{\Sigma}_{p_3} \frac{\partial f(\boldsymbol{a}, \boldsymbol{p_3})}{\partial \boldsymbol{p_3}}^T \tag{5.10}$$

In this case, the Jacobians are as follows:

$$\frac{\partial f(\boldsymbol{a}, \boldsymbol{p_3})}{\partial \boldsymbol{a}} = \begin{bmatrix} \cos(\varphi+\psi) & -\sin(\varphi+\psi) & -y\cos(\varphi+\psi) - x\sin(\varphi+\psi) \\ \sin(\varphi+\psi) & \cos(\varphi+\psi) & x\cos(\varphi+\psi) - y\sin(\varphi+\psi) \\ 0 & 0 & 1 \end{bmatrix} \tag{5.11}$$

$$\frac{\partial f(\boldsymbol{a}, \boldsymbol{p_3})}{\partial \boldsymbol{a}} = \begin{bmatrix} 1 & 0 & -y\cos(\varphi+\psi) - x\sin(\varphi+\psi) \\ 0 & 1 & x\cos(\varphi+\psi) - y\sin(\varphi+\psi) \\ 0 & 0 & 1 \end{bmatrix} \tag{5.12}$$

Fusion of multiple measurement

The final part is a fusion of the measurements. With the assumption that the normal multivariate distributions are independent, the resulting distribution can be computed as a product of the two multivariate distributions. As measurements come from the same source, the independence assumption is violated. If that remains neglected, then in standard (moments) Gaussian multivariate representation the resulting mean vector $\boldsymbol{\mu}_c$ and covariance matrix $\boldsymbol{\Sigma}_c$ of product of the two multivariate distributions are computed according to [59]:

$$\boldsymbol{\mu}_c = \left(\boldsymbol{\Sigma}_1^{-1} + \boldsymbol{\Sigma}_2^{-1}\right)^{-1} \left(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2\right) \tag{5.13}$$

$$\boldsymbol{\Sigma}_c = \left(\boldsymbol{\Sigma}_1^{-1} + \boldsymbol{\Sigma}_2^{-1}\right)^{-1} \tag{5.14}$$

These equations can be rewritten in a canonical form in which the resulting information vector $\boldsymbol{\xi}_n$ and information matrix $\boldsymbol{\Omega}_n$ of n Gaussian multivariate distributions product is taken from [60] (symbols are rewritten to be consistent in the thesis):

$$\mathbf{\Omega}_n = \sum_{i=1}^{n} \mathbf{\Omega}_i \tag{5.15}$$

$$\boldsymbol{\xi}_n = \sum_{i=1}^{n} \boldsymbol{\xi}_i \tag{5.16}$$

Both representations were tried and the latter implementation in Python showed as faster, and therefore, is used. Resulting pose estimation of robot then serves as input to *ekf_abs*. The computation of all above is implemented in *Aruco_detect.py* and *Estimate_pose_markers.py*

Due to not knowing, how severe is the violation of observations independence assumption, a second method where the resulting mean vector is calculated as a weighted average based on corresponding inverted covariances is implemented for fusion each measurement. The resulting covariance matrix is computed as average from individual covariances divided by the number of measurements. The method can be selected by adjusting the appropriate ROS parameter.

5.3 Experimental SLAM

With the same initial setting as in 4.3.2 for packages *gmapping*, *laser_scan_matcher*, and *ekf_rel* named node of *robot_localization* package, SLAM was performed. The package *urg_node* was used to read data from Lidar and publish them as ROS messages. As already mentioned in 5.2.2, due to the skid steer design of robot with no damping elements, except for the rubber wheels, the robot suffers a lot of vibration when moving, and therefore, the position estimation was better without integrating the acceleration from IMU as input to *ekf_rel*.

Further exclusion of input concerned orientation estimate from *imu_filter_madgwick*. As the IMU is located very near to motors and other electronics, but mostly due to the indoor operational environment, the magnetometer's readings were inaccurate. In the place where calibration was made, the orientation estimate was acceptable, but in other rooms, when moving in a straight line, the readings from magnetometer were changing significantly along the path.

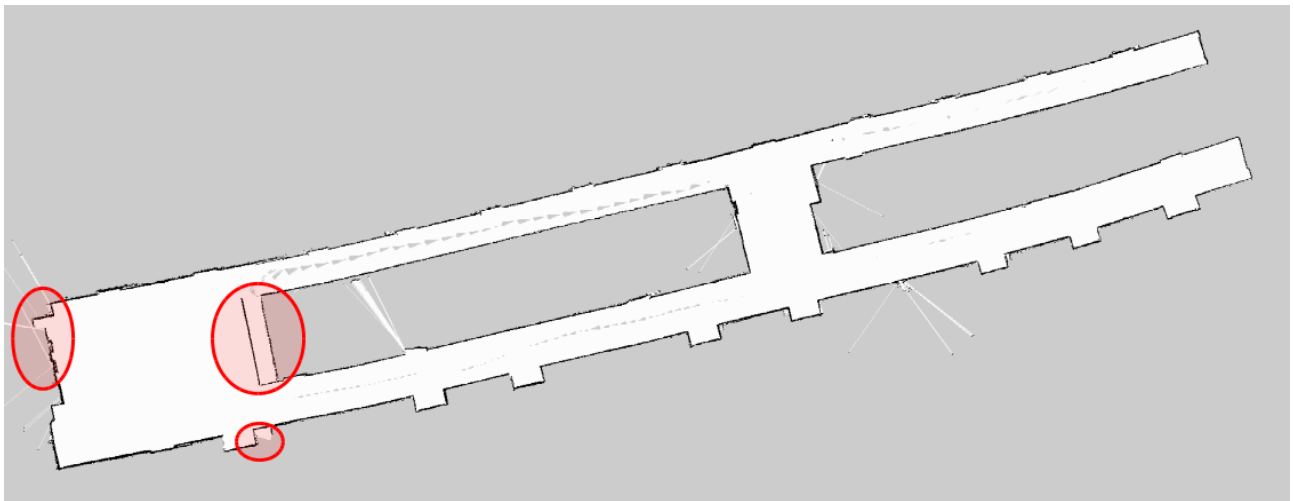


Figure 5.10: Created map of the seventh floor of the A3 building: In red are errors in experimental SLAM. The biggest ellipse marks mistake in loop closure, the left-most ellipse shows the place where chairs and furniture were placed and the last mark contains error added after the loop closure.

Translational and angular velocity estimation from wheel encoders were sufficiently accurate (angular velocity matched readings from gyroscope) as long as the floor remained hard. When the robot was moving over a carpet, the velocity estimation was significantly worse, but that kind of behavior should be expected.

To test SLAM in a bigger environment, the robot was controlled to drive in the seventh floor of the A3 building in Faculty Campus. Some parameters of *gmapping* had to be changed to give better results, and the created map can be seen in fig. 5.10.

Further testing took place in the Mechatronics laboratory and therefore SLAM was performed in the part of the laboratory. The created map of the part of the Mechatronics Laboratory can be seen in fig. 5.20.

5.4 Navigation to Charging Station

For successful navigation, ArUco markers had to be placed in the environment, their position measured and file with their pose and covariance had to be created as described in 5.2.3. The measurement of distance was performed manually by tape measure. Markers were placed on surfaces in that way that their yaw angle is approximately either 0°, 90°, 180° or 270°. The further the positions of ArUco markers from the coordinate system (and also the measurement) origin, the higher the uncertainty of the marker's position is. This was taken into account when inserting each marker to file *MechLAB_markers.yaml*.

Figure 5.11: Photo of placement of ArUco markers in the Mechatronics Laboratory

As discussed in 5.1, images from camera were processed either on RPi or working station. When processing the image with resolution 1280×720 on RPi, the frequency was about 10 Hz. With the same resolution, frequency higher than 20 Hz could be achieved on working station. However, the delay introduced when transmitting images from RPi to working station was inconsistent, and therefore, image processing is performed on RPi when navigating autonomously.

Navigation to the plug in the charging station is divided into two navigations subroutines. The first subroutine is navigating robot to the pose in front of the charging station and the

second one is navigating to plug in the charging station from the pose before the station. This must be done using backward motion as the connector for charging is located in rear part due to design requirements.

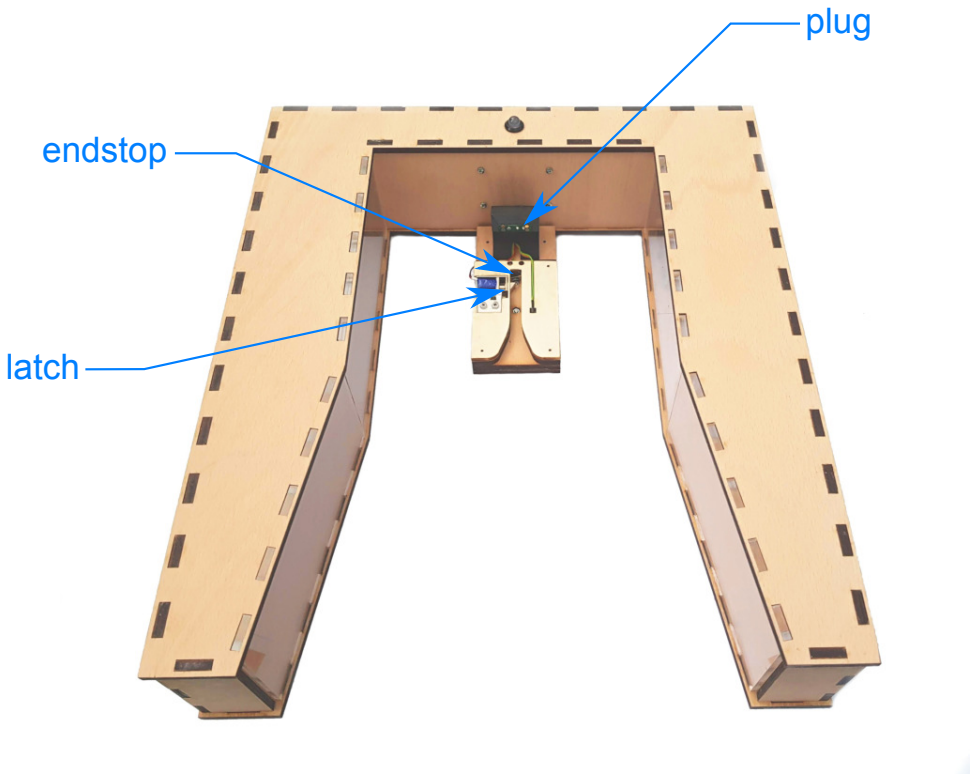


Figure 5.12: Photo of charging station: Taken from [2] and modified

The state machine that controls complete navigation to plug in charging station is described and depicted in figures 5.13 and 5.14.

Because of the requirements described above, two separate controllers and planners are loaded on start into *move_base_flex* with different names. Both planners are of the type of the *global_planner*. Standard planning is made by *Normal_planner*, the path from the pose in front of station to plug is planned by the *Charging_station_planner*. Setting of the second one is different in orientation mode, which adds orientation to each position point in the planned path. Here is mode set to backward orientation along all the path.

Controllers are also of the same type of *dwa* plugin. Standard controller is named simply *dwa* and controller for to/from navigation station is named *dwa_station*. Here the differences lie among other things in restriction of maximal velocities, setting the value of parameter *forward_point_distance* to zero to ensure backward motion, and also modifying values of *path_distance_bias, goal_distance_bias*, and *occdist_scale*. Also, the lower frequency of the latter controller gave better results.

The same behavior of the *Charging_station_planner* and *dwa_station* could be achieved by dynamically reconfiguring parameters of *Normal_planner* and *dwa*, but as completely different plugins for controller and planner were also tried and can be used in the future, loading them separately is a better option.

Dynamical reconfiguration must be performed for both *global_costmap* and *local_costmap* for parameters *footprint* that must be scaled down and *inflation_layer* that is completely shut down. These settings are true for moving to and from the station; afterward, the parameters are dynamically set to normal values.

The problem arising from moving back to the station is that it means moving away from the

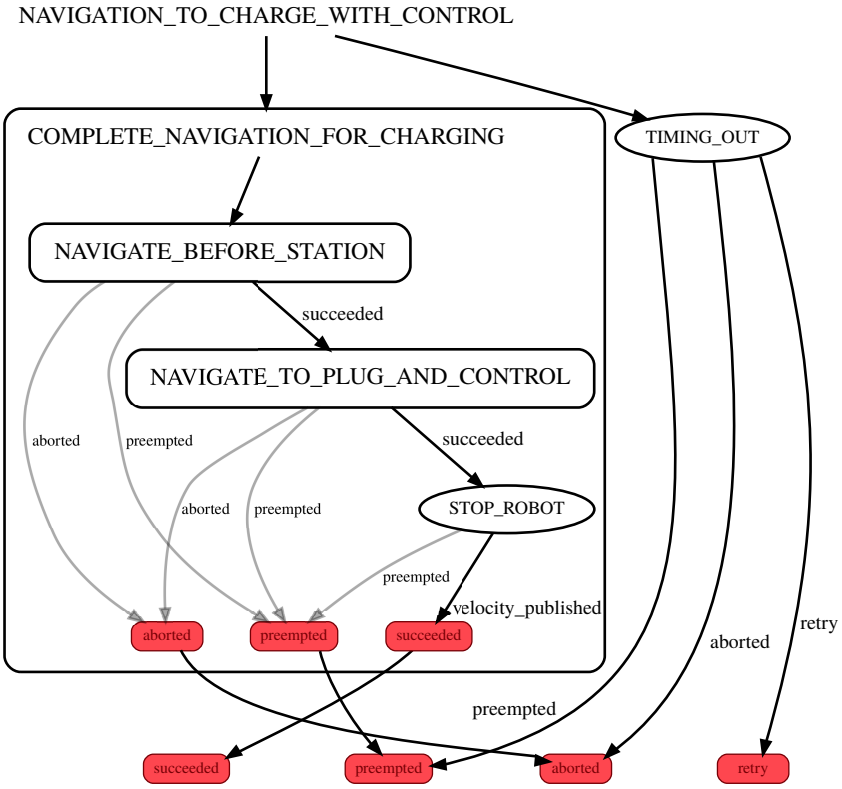


Figure 5.13: State machine for navigation to plug in charging station: Robot navigates before station (State machine *NAVIGATE_BEFORE_STATION* is normal navigation as shown in fig. 4.9) and then navigates moving backward (State machine *NAVIGATE_TO_PLUG_AND_CONTROL* is shown and described in fig. 5.14) to plug in charging station. The latch in the charging station does not place any resistance on the robot on the way to the station. When docking is successful, robot stops and state machines outcome is successful. Concurrently with this runs time countdown. If whole navigation takes a longer time then set time, *TIMING_OUT* state returns either retry (in which the whole process is repeated) or aborted, based on the set number of maximal repetitions and already unsuccessful attempts.

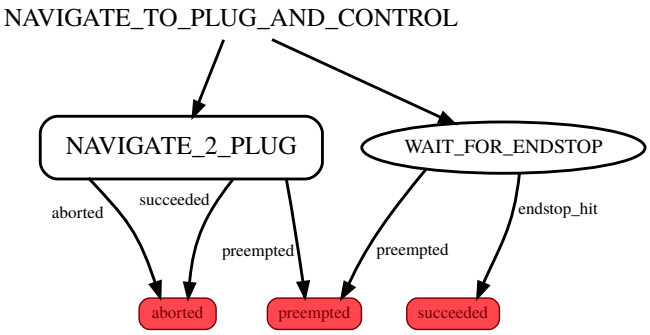


Figure 5.14: *NAVIGATE_TO_PLUG_AND_CONTROL*: Goal pose for navigation (state *NAVIGATE_TO_PLUG*) is deliberately set farther than the actual pose of plug due to not ending navigation as successful before endstop in charging station (status of endstop is published by RPi in charging station) is hit. State machine outcome is successful only if endstop is hit.

markers. Therefore an error of estimation pose from marker is increasing, and as for successful docking accuracy of around 5 *cm* is needed, the visible markers when moving backward should be placed as precisely as possible. The biggest problem was arising when the yaw angle of the real marker did not correspond sufficiently accurate with the yaw angle of the marker in

MechLAB_markers.yaml.

When marker's pose is measured accurately enough, the robot performs successful navigation to plug in most cases at first attempt. Data from one successful navigation are shown in fig. 5.15. In view of these findings, two conclusions can be drawn: Charging station should be placed in such a place, that when moving backwards, markers are near and visible preferably on both sides of the robot. The second one is that markers should be measured in such a way that ensures, that markers near the station are measured most accurately. For example, map origin could be in charging station.

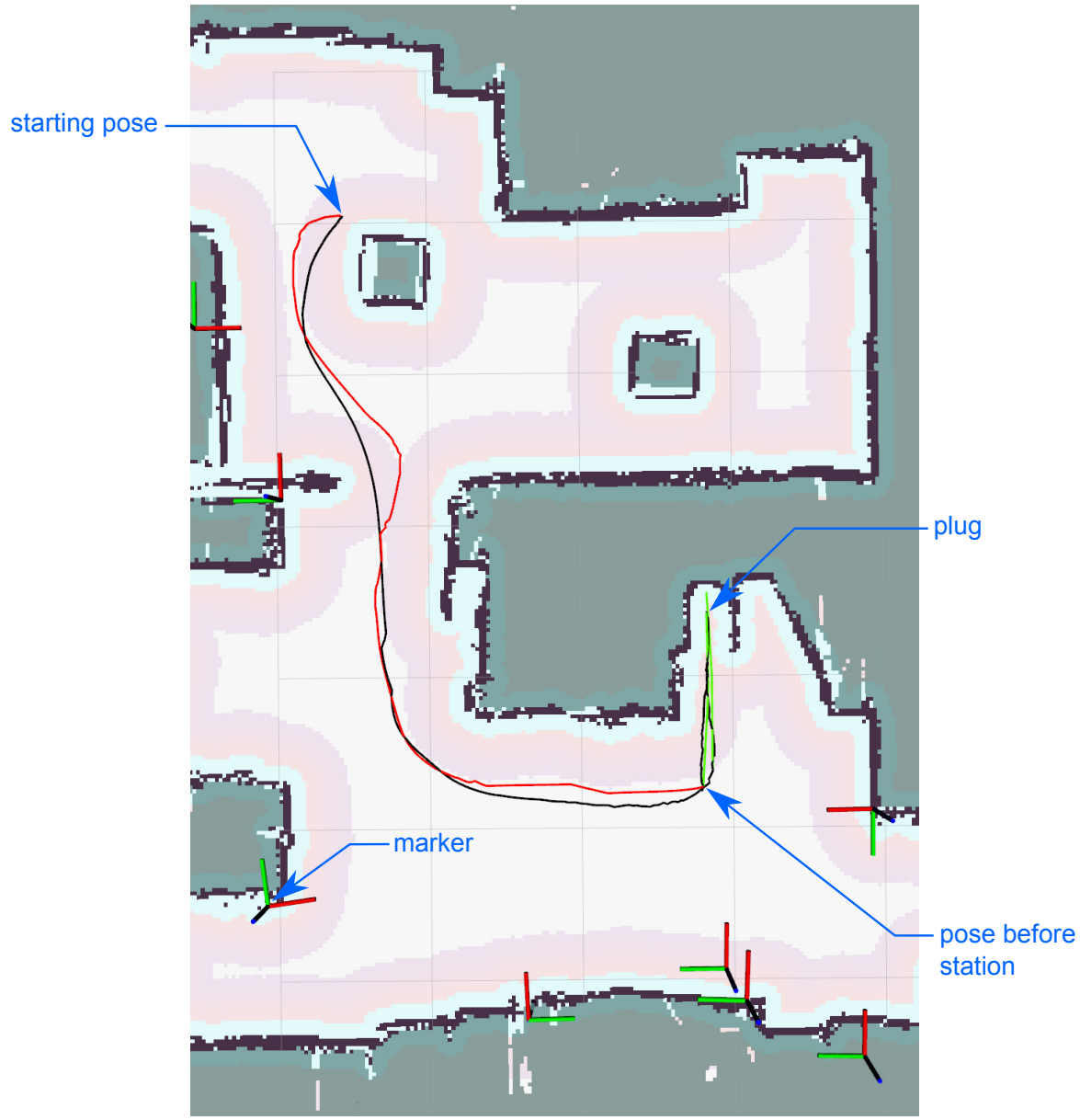


Figure 5.15: Navigation to and from plug in charging station: Black line represents the actual path of the robot, the red line shows the global plan to the pose before the station computed from *Normal_planner*, and the green represents planned path from pose before the station to the plug and back created by *Charging_station_planner*. Also, the markers that the robot saw along the path are shown. Their position is based on the last transformation before they were no longer seen.

All settings as pose before charging station, the pose of the plug in charging station, maximal

time for one attempt for navigation for charging, and number of repetitions before aborting navigation for charging are accessible as ROS parameters, and therefore their value can be easily changed in *navig_flex_real.launch* file that launches all needed nodes on Working station for complete navigation and charging cycle described here and in the following section.

5.5 Complex Behavior SMACH

As the robot is intended to operate partially autonomously, the state machine *NAVIGATION_TO_CHARGE_WITH_CONTROL* introduced in the previous section builds only part (serves as one state) of the complex state machine that controls the robot. The top layer of the created state machine is visible in fig. 5.16.

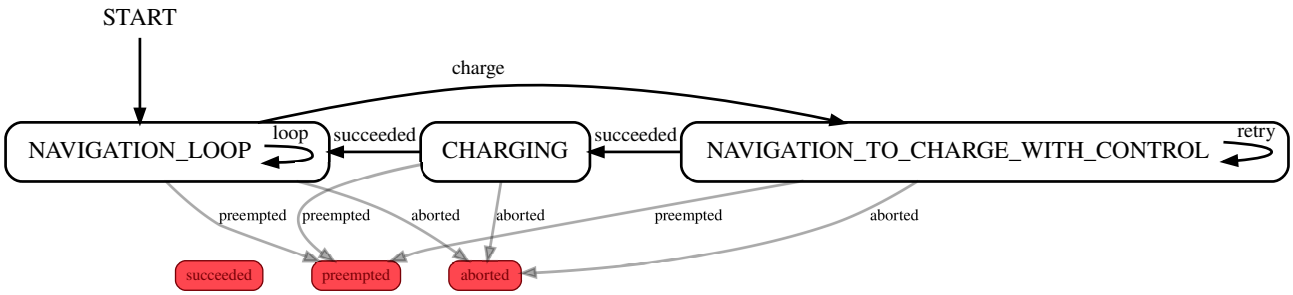


Figure 5.16: The top layer of state machine that controls robot

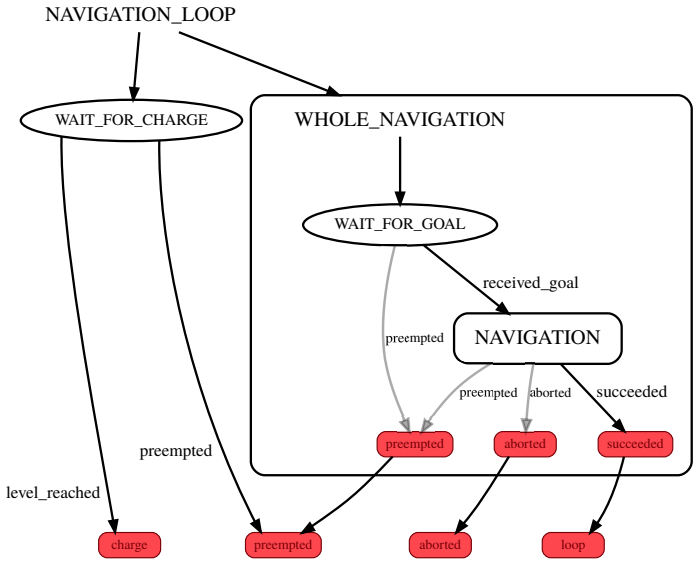


Figure 5.17: The concurrent state machine that monitors robot's battery level and navigates to the user-defined goals or allows to manually control robot.

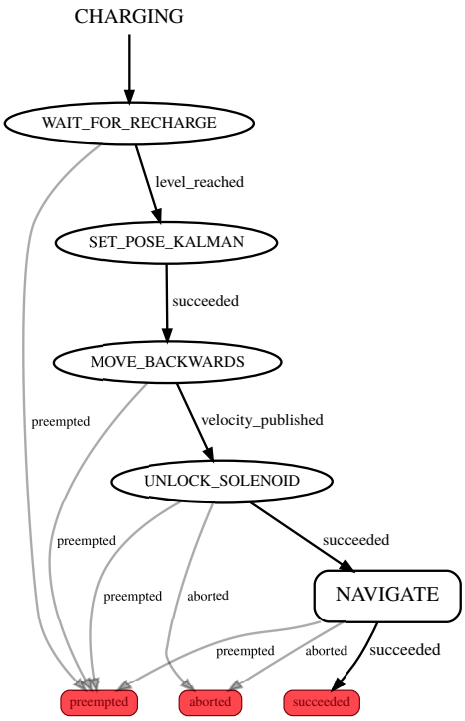


Figure 5.18: Sequence of states after docking

The state machine *NAVIGATION_LOOP* depicted in fig. 5.17 is a concurrent state machine. It simultaneously subscribes to topic */volt_battery* containing the value of robot battery voltage and either navigates the robot to the user defined-goal or allows to manually

control the robot to drive. If the battery voltage level reaches the set value, it pre-empts the *WHOLE_NAVIGATION* state and the *NAVIGATION_TO_CHARGE_WITH_CONTROL* takes control.

After successful execution of the above-mentioned state, robot is plugged in charging station and is being charged. Then in state machine *CHARGING*, the battery voltage level is monitored and when it reaches the defined threshold, pose of the plug is set as robot's current pose to *ekf_abs* through *robot_localization/SetPose* service to correct an error in pose estimation acquired when navigating to charging station. Then robot is commanded to move backward to alleviate the load on the latch actuated by a solenoid. More on charging station design in [2]. After that, the signal is sent to charging station RPi to release the latch, and then robot navigates to the pose in front of the station and *NAVIGATION_LOOP* becomes the active state.

This whole state machine runs on the working station but subscribes and publishes to topics or services also hosted on the robot or the charging station.

5.6 Mini-games for Users

In accordance with the project aim described in section 3, some entertaining tasks for the visitors of the Exhibition must be developed. Tasks for users concerning the use of AR are created and described in [2]. Here are further characterized two tasks that are focused more on the robotic aspect of the project. These tasks are not included in the state machines described above.

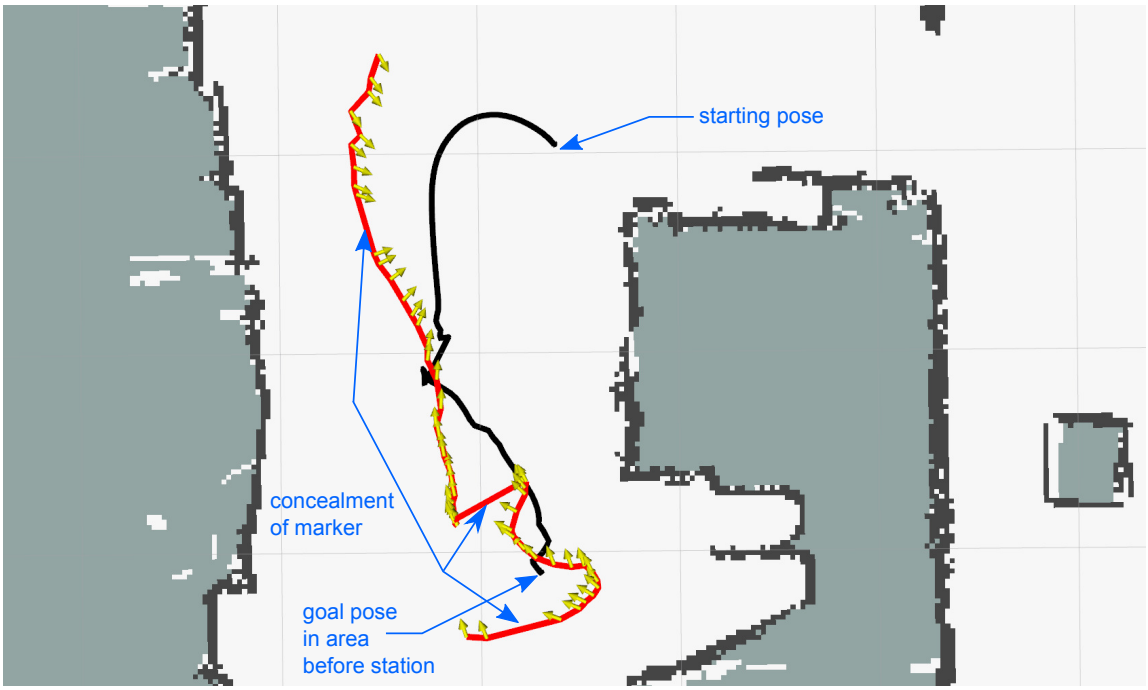


Figure 5.19: Marker following: ArUco marker was held by hand and moved in such a way that the robot followed the marker in the position before the charging station. On that way, the marker was deliberately hidden 3 times, moved and then again shown to test if the robot can handle it. The aim of this experiment was to guide the robot to the desired pose, not to let the robot precisely follow the same path as the marker's path. The red line with orientation arrows represent the marker's time-varying pose and the black line shows the path of the robot.

5.6.1 Navigation of Malfunctioned Robot

This task will be presented to visitors as follows: Due to the simulated malfunction of the robot's global navigation system, the robot must be guided to charging station. The guide is performed by another robot, which has the ArUco marker attached to the rear part of the robot, controlled by the user. The manually controlled robot is slowly driving from an initial position (where its marker was visible to malfunctioned robot's camera) followed by the malfunctioned robot. If malfunctioned robot reaches a defined area before the station, the task is fulfilled.

As at the time of the writing this thesis, only one robot was available; this task is not completely developed and tested. However the most significant part - following an ArUco marker, is implemented and tested. For this functionality was the script from *Ubiquity Robotics* [61] under BSD license used and modified to work alongside other custom nodes. Modified node subscribes to the topic containing pose estimation of ArUco markers and then compute necessary velocity to reach the desired pose in front of the selected marker. Main loop of the script is shown in simplified form in algorithm 1.

Algorithm 1 Main loop for velocity computation

```
1: while ROS is running do
2:     if markerSpotted() then
3:         lostCount ⇐ 0                    ▷ set number of times the marker was not spotted to 0
4:         x; y ⇐ getCurrentMarkerPosition()          ▷ x is forward, y is lateral distance
5:         if x > maxDistance then
6:             linSpeed; angSpeed ⇐ 0; 0
7:         else
8:             yaw ⇐ atan2(y, x)
9:             angSpeed ⇐ yaw·angularRate − angSpeed/2   ▷ Damping for smoother motion
10:            angSpeed ⇐ adjustToLimits(angSpeed)
11:            linSpeed ⇐ (x−minDistance)·linearRate
12:            linSpeed ⇐ adjustToLimits(linSpeed)
13:        end if
14:    else if 0 < lostCount ≤ maxHysteresis then
15:        linSpeed ⇐ linSpeed·linearDecay
16:        lostCount ⇐ lostCount+1
17:    else if maxHysteresis < lostCount < maxRotations then
18:        angSpeed ⇐ sign(angSpeed)·lostAngularSpeed
19:        lostCount ⇐ lostCount+1
20:    else
21:        linSpeed; angSpeed ⇐ 0; 0
22:    end if
23:    publishVelocity(linSpeed, angSpeed)
24:    sleep(20 Hz)
25: end while
```

where maxDistance, minDistance, maxRotations, maxHysteresis, lostAngularSpeed, angularRate, linearRate are adjustable parameters, here renamed for better readability.

To sum up algorithm: based on distance and angle of the marker with respect to the robot, velocity is computed; when the marker is lost, linear velocity decreases. If set marker to follow is not visible for an even bigger amount of time, the robot rotates for a set time to find the

marker. If marker is still not found, the robot stops moving and waits for the marker to reappear in the image.

As was written above, another robot is not available, thus, the functionality of this task was tested manually. Selected marker was held by hand and moved from arbitrary position to position in front of the charging station. Data from this experiment are shown in fig. 5.19.

Script implementing this behavior is called *Follow_target_marker.py*.

5.6.2 Creating Map of Environment

Next task is designed to show the participants how the real world sensors are not exact, and therefore, some error is always introduced when working with robots. With settings coarsely described in 2.3.3 and 5.3, nodes necessary to perform SLAM will be launched, and the person performing this task will manually control the robot equipped with the Lidar by a joystick in such a way that the whole environment is mapped. The current state of the map will be shown live in Rviz. When the user is satisfied with the created map, the map is saved and its location path sent through *mech_ros_msgs/Map_comparsion* service. Automatically saving the map and sending its path is not implemented yet, but the automatic comparison of the created map with the ground truth map through service is already programmed and is further described.

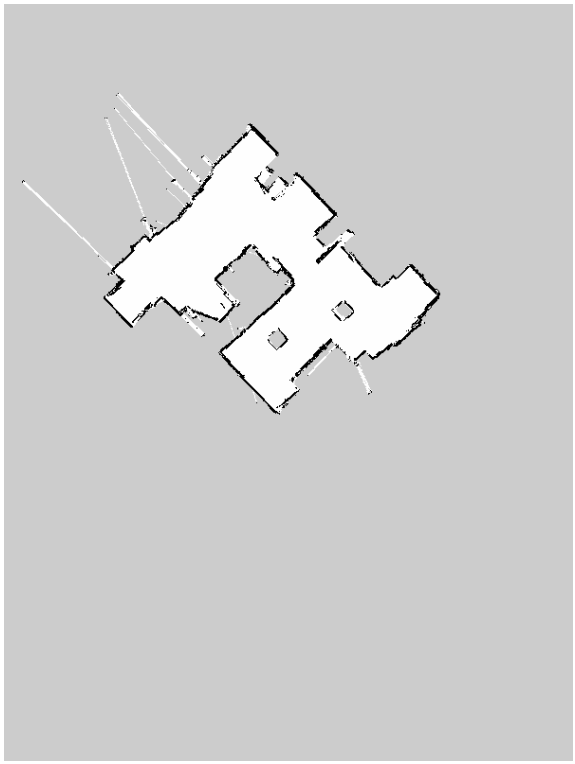


Figure 5.20: The created map of the part of the Mechatronics Laboratory

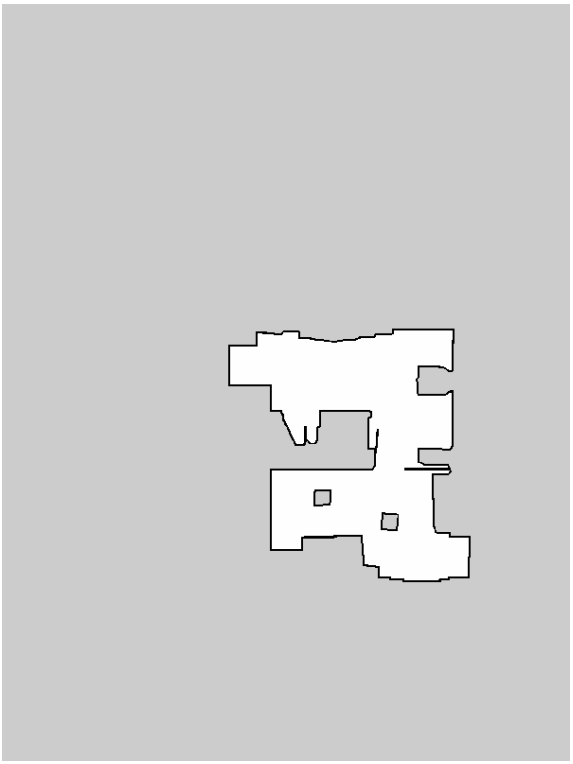


Figure 5.21: The ground truth map of the part of the Mechatronics Laboratory

As inputs to map comparison serves two separate maps. As is shown in figures 5.20 and 5.21, these maps are saved in different rotations and different positions with respect to the image. These attributes are dependent on the initial pose of the robot when starting SLAM. Therefore the maps have to be aligned to be compared. The procedure to compute transformations is shown as pseudo code in algorithm 2. Most functions' purpose should be apparent from the name of the function. Function *matchTemplate* is an OpenCV function that slides smaller image

over bigger through every possible translation and based on selected type of metric returns best transformation and value that represents how similar the images are.

To account for rotation, template image (cropped ground truth map) is rotated and then matched against the bigger image. To speed up the algorithm; the best two angles using coarse resolution are found, and then is the angle between these two refined.

Algorithm 2 Find best matching angle and coordinates between maps

```
1: function compareMaps(createdMap, groundTruthMap, coarseAngle, refAngle)
2:     bValue; bAngle; sBestAngle ⇐ 1e10; 0; 0
3:     template ⇐ cropImage(groundTruthMap)
4:     croppedMap ⇐ cropImage(createdMap)
5:     for i ⇐ 0 to 360/coarseAngle do
6:         rotatedTemplate ⇐ rotateImage(template, i·coarseAngle)
7:         value; coord ⇐ matchTemplate(croppedMap, rotatedTemplate)
8:         if value < bValue then
9:             bValue; bAngle; sBestAngle; bCoord ⇐ value; i·coarseAngle; bAngle; coord
10:        end if
11:    end for
12:    signum ⇐ sign(sBestAngle−bAngle)
13:    for i ⇐ 0 to int(coarseAngle/refAngle) do
14:        rotatedTemplate ⇐ rotateImage(template, bAngle+signum·i·refAngle)
15:        value; coord ⇐ matchTemplate(croppedMap, rotatedTemplate)
16:        if value < bValue then
17:            bValue; bAngleRef; bCoord ⇐ value; bAngle+signum·i·refAngle; coord
18:        end if
19:    end for
20:    return bAngleRef; bCoord
21: end function
```

The assumption for this algorithm to work correctly is that both map have a same grid resolution. The resulting angle and coordination for template transformation are used for visualization. To visualize the resulting map, as shown in fig. 5.22, several bitwise, threshold, and other operations are used on both images. Then the map with marked pixels differences is visualized in a separate window with a percentage and written evaluation of the created map. The percentage is counted as a ratio of differentiating pixels to pixels of template subtracted from 1.

The above-described functionalities are implemented in script *Compare_maps.py* with brief comments.

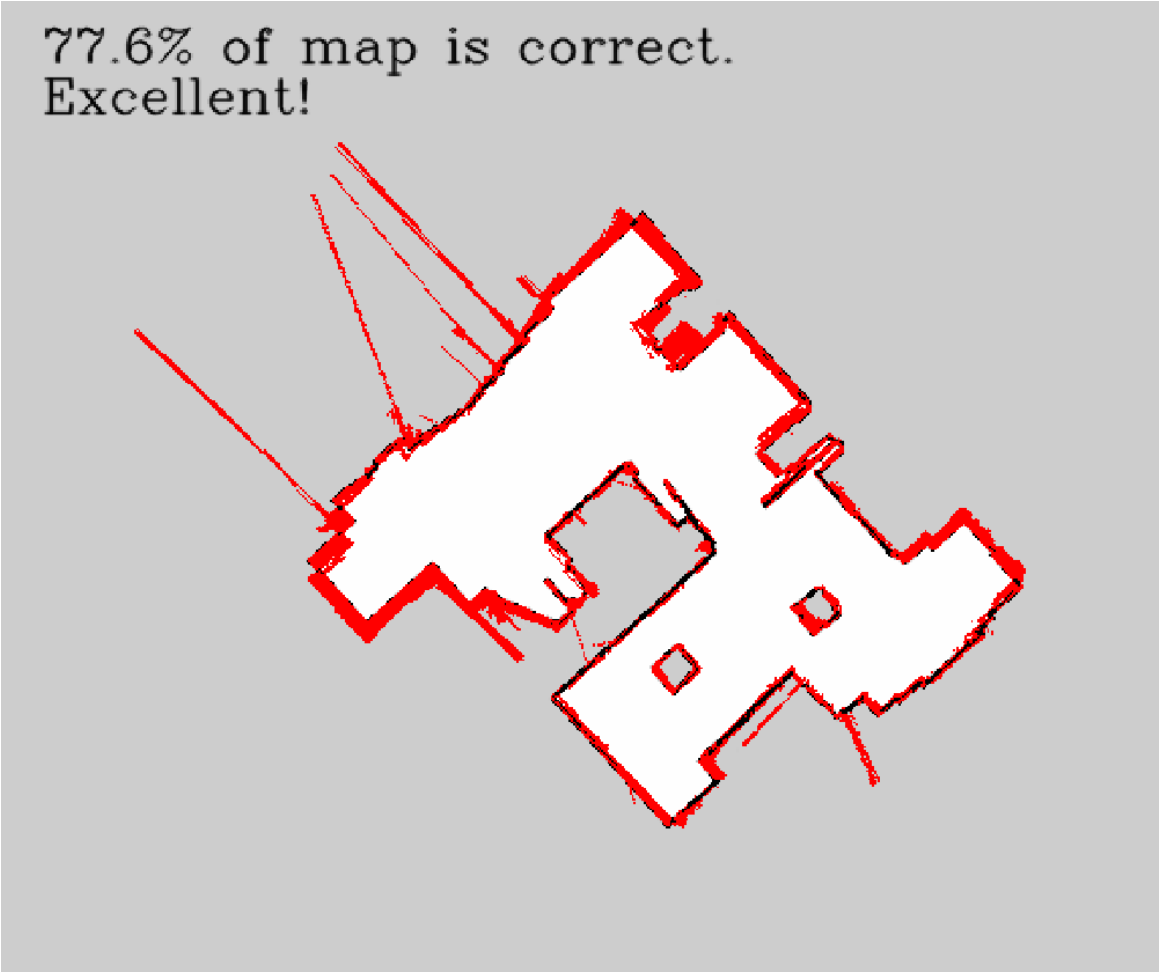


Figure 5.22: The resulting comparison of created map with ground truth map: Red pixels show the differentiating pixels.

6 Conclusion

The outcome of this thesis is a functional state machine that allows to manually control the robot, to give goal positions for navigation, and if needed, takes care of autonomous charging of the robot. After the agreement with the thesis's supervisor, the software was created with the aim to include only one robot, working station and charging station with the possibility to extend it to incorporate several robots, working and charging stations in the future.

The first part of the thesis serves as a brief introduction to the Gazebo and ROS framework, describes several ROS packages and principles behind them. Next, characteristics of sensors of the robot and some theory of how to use them in the localization system are presented.

The following part is focused on the creation of the model of the robot and the design of the simulation environment in Gazebo. The environment is modelled in such a way that it resembles intended settings in its final destination. Some of the program adjustable parameters are discussed. Twenty ArUco markers were placed in the environment to allow for successful localization of robot through image processing from the mounted camera's image. Robot's parts are modelled using Xacro language together with sensors plugin to obtain a fully functional simulation model. This is followed by the description of the used settings for localization and navigation packages.

In the experimental part, the hardware on the working station, charging station, and robot is listed. Several experiments were performed to estimate unknown parameters for the kinematics model and to calibrate IMU and obtain the yaw angle. Following experiments are focused on determining the accuracy of the marker's pose estimation with respect to the camera. To fuse the camera measurement in the localization node, the procedure to convert all camera measurements in one pose estimation in the global coordinate system is introduced. Created maps of the environment from SLAM are shown and utilized in localization and navigation. For successful navigation to the plug in the charging station, a state machine created in SMACH, is used. Expansion of this state machine allows the robot to function autonomously with the possibility of manual control by the user.

The interaction can also be composed of the two created mini-games. The first mini-game is based on following the target ArUco marker. The aim of this task is to guide the robot to the charging station by moving the target marker. The movement is performed manually in the meantime, but as more than one robot is available, the marker can be placed to the rear part of one of the robots and manually control this robot to be followed by another robot. In the second mini-game, the user controls the robot to create a map of the environment using described settings for SLAM. After that, the created map is automatically compared with the ground truth map, and the resulting comparison is shown with evaluation.

With regard to the above-given summary of accomplished work, it can be said that Master's thesis goals were fulfilled. The selected type and amount of sensors have proven sufficient for the basic navigation, however, for more robust robot behavior, some suggestions for improvement are given in the last section.

6.1 Suggestions for Further Work

While working on this thesis, many additional ideas, suggestions, and fixtures came to mind. These suggestions can be divided into two sections. First incorporates knowledge about mistakes in design both hardware and software part:

- Read data from IMU and encoders at a higher rate, average them and send them at same rate as is now
- Implement additional sensor to have at least basic knowledge about obstacles in front of robot
- Implement and test additional controller for very narrow spaces – *dwa* does not perform quite well in that environment

The second section represents interesting ideas for further project development:

- Create a map of ArUco markers automatically as part of the SLAM
- Estimate pose of camera based on all visible markers by minimizing reprojection error of all markers in image in OpenCV *solvePnP* function
- Detect obstacles from image based on features extraction
- Extend created software to incorporate several robots, working and charging stations

List of Abbreviations

SAE	Society of Automotive Engineers
ROS	Robot Operating System
BSD	Berkeley Software Distribution
OGRE	Object-Oriented Graphics Rendering Engine
GUI	Graphical User Interface
RT	Real time
HIL	Hardware-in-the-loop
TCP/IP	Transmission Control Protocol/Internet Protocol
ODE	Open Dynamics Engine
REP	ROS Enhancement Proposal
SI	System of Units
ENU	East-North-Up
IMU	Inertial measurement unit
EKF	Extended Kalman Filter
AMCL	Adaptive Monte Carlo Localization
SLAM	Simultaneous Localization and Mapping
DWA	Dynamic Window Approach
MBF	*Move_base_flex*
BT	Behavior Tree
SMACH	State machine or SMACH package
Lidar	Light Detection and Ranging
DOF	Degrees of Freedom
ICR	Instantaneous Center of Rotation
AR	Augmented Reality
SDF	Simulation Description Format

XML Extensible Markup Language

URDF Universal Robot Description Format

Xacro XML macro

RPi Raspberry Pi

HAT Hardware attached on top

USB Universal Serial Bus

SPI Serial Peripheral Interface

UART Universal Asynchronous Receiver and Transmitter

I^2C Inter-Integrated Circuit

List of Figures

List of Tables

Bibliography

[1] MODEL S OWNER'S MANUAL. In: *Tesla.com* [online]. December 17, 2018 [cit. 2019-05-18]. Available: https://www.tesla.com/sites/default/files/model_s_owners_manual_north_america_en_us.pdf

[2] ADÁMEK, Roman. *Návrh mobilního robotu s uživatelským rozhraním využívajícím rozšířenou realitu*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně. Vedoucí práce Ing. Michal Bastl.

[3] ZANCHIN, Betina Carol, Rodrigo ADAMSHUK, Max Mauro SANTOS a Kathya Silvia COLLAZOS. On the instrumentation and classification of autonomous cars. In: *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* [online]. IEEE, 2017, 2017, s. 2631-2636 [cit. 2019-05-18]. DOI: 10.1109/SMC.2017.8123022. ISBN 978-1-5386-1645-1. Available: http://ieeexplore.ieee.org/document/8123022/

[4] Gazebo Tutorials. *Gazebosim* [online]. Open Source Robotics Foundation, 2014 [cit. 2019-04-13]. Available: http://gazebosim.org/tutorials?cat=install

[5] Gazebo_ros_api. In: *Gazebosim: ROS overview* [online]. Open Source Robotics Foundation, [cit. 2019-04-13]. Available: http://gazebosim.org/tutorials?tut=ros_overview&cat=connect_ros

[6] FOOTE, Tully a Mike PURVIS. Standard Units of Measure and Coordinate Conventions. *ROS: REPS* [online]. 31-Dec-2014 [cit. 2019-04-14]. Available: http://www.ros.org/reps/rep-0103.html

[7] MEEUSSEN, Wim. Coordinate Frames for Mobile Platforms. *ROS: REPS* [online]. 27-Oct-2010 [cit. 2019-04-14]. Available: http://www.ros.org/reps/rep-0105.html

[8] FOOTE, Tully. Tf: The transform library. In: *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)* [online]. IEEE, 2013, 2013, s. 1-6 [cit. 2019-05-21]. DOI: 10.1109/TePRA.2013.6556373. ISBN 978-1-4673-6225-2. Available: http://ieeexplore.ieee.org/document/6556373/

[9] BOVBEL, Paul. Conventions for IMU Sensor Drivers: REP 145. *ROS: REPS* [online]. 02-Feb-2015 [cit. 2019-04-14]. Available: http://www.ros.org/reps/rep-0145.html

[10] THRUN, Sebastian, Wolfram BURGARD a Dieter FOX. *Probabilistic Robotics*. [1st ed.]. Cambridge: The Mit Press, 2006. ISBN 978-0-262-20162-9.

[11] DE MAESSCHALCK, R., D. JOUAN-RIMBAUD a D.L. MASSART. The Mahalanobis distance. *Chemometrics and Intelligent Laboratory Systems* [online]. 2000, **50**(1), 1-18 [cit. 2019-05-21]. DOI: 10.1016/S0169-7439(99)00047-7. ISSN 01697439. Available: https://linkinghub.elsevier.com/retrieve/pii/S0169743999000477

[12] AHN, Joseph, Moonseo PARK, Hyun-Soo LEE, Sung Jin AHN, Sae-Hyun JI, Kwonsik SONG a Bo-Sik SON. Covariance effect analysis of similarity measurement methods for early construction cost estimation using case-based reasoning. *Automation in Construction* [online]. 2017, **81**, 254-266 [cit. 2019-05-21]. DOI: 10.1016/j.autcon.2017.04.009. ISSN 09265805. Available: https://linkinghub.elsevier.com/retrieve/pii/S0926580517303242

[13] GALEANO, Pedro, Esdras JOSEPH a Rosa E. LILLO. The Mahalanobis Distance for Functional Data With Applications to Classification. *Technometrics* [online]. 2015, **57**(2), 281-291 [cit. 2019-05-05]. DOI: 10.1080/00401706.2014.902774. ISSN 0040-1706. Available: http://www.tandfonline.com/doi/full/10.1080/00401706.2014.902774

[14] MOORE, Thomas a Daniel STOUCH. A Generalized Extended Kalman Filter Implementation for the Robot Operating System. MENEGATTI, Emanuele, Nathan MICHAEL, Karsten BERNS a Hiroaki YAMAGUCHI, ed. *Intelligent Autonomous Systems 13* [online]. Cham: Springer International Publishing, 2016, 2016-9-3, s. 335-348 [cit. 2019-05-21]. Advances in Intelligent Systems and Computing. DOI: 10.1007/978-3-319-08338-4_25. ISBN 978-3-319-08337-7. Available: http://link.springer.com/10.1007/978-3-319-08338-4_25

[15] *Robot_localization wiki* [online]. [cit. 2019-04-25]. Available: http://docs.ros.org/kinetic/api/robot_localization/html/index.html

[16] Robot_localization: Kinetic. *GitHub* [online]. 2019, 15 Feb 2019 [cit. 2019-04-25]. Available: https://github.com/cra-ros-pkg/robot_localization/tree/kinetic-devel

[17] O.H. MADGWICK, Sebastian. *An efficient orientation filter for inertial and inertial/magnetic sensor arrays*. 2010. Available: http://x-io.co.uk/res/doc/madgwick_internal_report.pdf

[18] Laser_scan_matcher: Package Summary. *ROS.org* [online]. Open Source Robotics Foundation, 1 Jan 2019 [cit. 2019-05-01]. Available: http://wiki.ros.org/laser_scan_matcher

[19] SANTOS, Joao Machado, David PORTUGAL a 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)* [online]. IEEE, 2013, 2013, s. 1-6 [cit. 2019-05-21]. DOI: 10.1109/SSRR.2013.6719348. ISBN 978-1-4799-0880-6. Available: http://ieeexplore.ieee.org/document/6719348/

[20] GRISETTI, Giorgio, Cyrill STACHNISS a Wolfram BURGARD. Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters. *IEEE Transactions on Robotics* [online]. 2007, **23**(1), 34-46 [cit. 2019-05-21]. DOI: 10.1109/TRO.2006.889486. ISSN 1552-3098. Available: http://ieeexplore.ieee.org/document/4084563/

[21] GERKEY, Brian. Gmapping: Package Summary. *ROS.org* [online]. Open Source Robotics Foundation, 4 Feb 2019 [cit. 2019-05-01]. Available: http://wiki.ros.org/gmapping

[22] ABDELRASOUL, Yassin, Abu Bakar Sayuti HM SAMAN a Patrick SEBASTIAN. A quantitative study of tuning ROS gmapping parameters and their effect on performing indoor 2D SLAM. In: *2016 2nd IEEE International Symposium on Robotics and Manufacturing Automation (ROMA)* [online]. IEEE, 2016, 2016, s. 1-6 [cit. 2019-05-21]. DOI: 10.1109/ROMA.2016.7847825. ISBN 978-1-5090-0928-2. Available: http://ieeexplore.ieee.org/document/7847825/

[23] MARDER-EPPSTEIN, Eitan. Move_base. *Wiki.ROS.org* [online]. 2018-09-27 [cit. 2019-05-21]. Available: http://wiki.ros.org/move_base

[24] PÜTZ, Sebastian a Simon JORGE. Move Base Flex – A Highly Flexible Navigation Framework for Mobile Robots. *Proc. IROS 2018. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-2018), October 1-5, Madrid, Spain*. IEEE, 2018.

[25] LU, David. Inflation: Inflation Costmap Plugin. *Wiki.ROS.org* [online]. 2013-04-25 [cit. 2019-04-27]. Available: http://wiki.ros.org/costmap_2d/hydro/inflation

[26] MARDER-EPPSTEIN, Eitan, David LU a Dave HERSHBERGER. Costmap_2d: Package Summary. *Wiki.ROS.org* [online]. 2018-01-10 [cit. 2019-04-27]. Dostupné z: http://wiki.ros.org/costmap_2d

[27] ZHENG, Kaiyu. *ROS Navigation Tuning Guide*. 2017. Available: http://kaiyuzheng.me/documents/navguide.pdf

[28] FOX, D., W. BURGARD a S. THRUN. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine* [online]. **4**(1), 23-33 [cit. 2019-05-21]. DOI: 10.1109/100.580977. ISSN 10709932. Available: http://ieeexplore.ieee.org/document/580977/

[29] MARDER-EPPSTEIN, Eitan. Dwa_local_planner: Package Summary. *Wiki.ROS.org* [online]. 2018-06-14 [cit. 2019-04-27]. Available: http://wiki.ros.org/dwa_local_planner?distro=kinetic

[30] BOHREN, Jonathan a Steve COUSINS. The SMACH High-Level Executive [ROS News]. *IEEE Robotics & Automation Magazine* [online]. 2010, **17**(4), 18-20 [cit. 2019-05-21]. DOI: 10.1109/MRA.2010.938836. ISSN 1070-9932. Dostupné z: http://ieeexplore.ieee.org/document/5663871/

[31] Camera Module. *Raspberrypi.org: Documentation* [online]. Raspberry Pi Foundation [cit. 2019-04-29]. Dostupné z: https://www.raspberrypi.org/documentation/hardware/camera/

[32] The 3-Clause BSD License. *Opensource.org* [online]. Open Source Initiative [cit. 2019-04-29]. Available: https://opensource.org/licenses/BSD-3-Clause

[33] GARRIDO-JURADO, S., R. MUÑOZ-SALINAS, F.J. MADRID-CUEVAS a M.J. MARÍN-JIMÉNEZ. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition* [online]. 2014, **47**(6), 2280-2292 [cit. 2019-05-21]. DOI: 10.1016/j.patcog.2014.01.005. ISSN 00313203. Available: https://linkinghub.elsevier.com/retrieve/pii/S0031320314000235

[34] GARRIDO-JURADO, S., R. MUÑOZ-SALINAS, F.J. MADRID-CUEVAS a R. MEDINA-CARNICER. Generation of fiducial marker dictionaries using Mixed Integer Linear Programming. *Pattern Recognition* [online]. 2016, **51**, 481-491 [cit. 2019-05-21]. DOI: 10.1016/j.patcog.2015.09.023. ISSN 00313203. Available: https://linkinghub.elsevier.com/retrieve/pii/S0031320315003544

[35] Detection of ArUco Markers. *Docs.opencv.org* [online]. 22 Dec 2017 [cit. 2019-04-29]. Available: https://docs.opencv.org/3.4.0/d5/dae/tutorial_aruco_detection.html

[36] Detection of ChArUco Corners. *Docs.opencv.org* [online]. Open Source Computer Vision, 4 Jul 2018 [cit. 2019-04-29]. Available: https://docs.opencv.org/3.4.2/df/d4a/tutorial_charuco_detection.html

[37] Camera Calibration and 3D Reconstruction. *Docs.opencv.org* [online]. opencv dev team, 29 Apr 2019 [cit. 2019-04-30]. Available: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[38] Calibration with ArUco and ChArUco. *Docs.opencv.org* [online]. Open Source Computer Vision, 22 Dec 2017 [cit. 2019-04-30]. Available: https://docs.opencv.org/3.4.0/da/d13/tutorial_aruco_calibration.html

[39] CameraIntrinsics. In: *Mathworks.com* [online]. The MathWorks [cit. 2019-04-30]. Available: https://www.mathworks.com/help/vision/ref/cameraintrinsics.html

[40] STMICROELECTRONICS. *LSM9DS1: iNEMO inertial module*. 2015. Available: https://www.st.com/content/ccc/resource/technical/document/datasheet/1e/3f/2a/d6/25/eb/48/46/DM00103319.pdf/files/DM00103319.pdf/jcr:content/translations/en.DM00103319.pdf

[41] TEDALDI, David, Alberto PRETTO a Emanuele MENEGATTI. A robust and easy to implement method for IMU calibration without external equipments. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)* [online]. IEEE, 2014, 2014, s. 3042-3049 [cit. 2019-05-21]. DOI: 10.1109/ICRA.2014.6907297. ISBN 978-1-4799-3685-4. Available: http://ieeexplore.ieee.org/document/6907297/

[42] RENAUDIN, Valérie, Muhammad Haris AFZAL a Gérard LACHAPELLE. Complete Triaxis Magnetometer Calibration in the Magnetic Domain. *Journal of Sensors* [online]. 2010, **2010**, 1-10 [cit. 2019-05-21]. DOI: 10.1155/2010/967245. ISSN 1687-725X. Available: http://www.hindawi.com/journals/js/2010/967245/

[43] MARKOVSKY, I., A. KUKUSH a S. Van HUFFEL. Consistent least squares fitting of ellipsoids. *Numerische Mathematik* [online]. 2004, **98**(1), 177-194 [cit. 2019-05-21]. DOI: 10.1007/s00211-004-0526-9. ISSN 0029-599X. Available: http://link.springer.com/10.1007/s00211-004-0526-9

[44] SIEGWART, Roland a Illah Reza NOURBAKHSH. *Introduction to autonomous mobile robots*. Cambridge, Mass.: MIT Press, 2004. ISBN 0-262-19502-x.

[45] *Scanning Laser Range Finder URG-04LX-UG01: Specifications*. Hokuyo Automatic, 2009. Available: https://www.robotshop.com/media/files/pdf/hokuyo-urg-04lx-ug01-specifications.pdf

[46] Hokuyo URG-04LX-UG01 Scanning Laser Rangefinder. In: *Robotshop.com* [online]. [cit. 2019-04-29]. Available: https://www.robotshop.com/en/hokuyo-urg-04lx-ug01-scanning-laser-rangefinder.html

[47] NAJMAN, Jan. *Rozšíření robotu Car4 o palubní počítač a snímače Kinect a Hokuyo*. Brno, 2013. Bakalářská práce. Vysoké učení technické v Brně. Vedoucí práce Ing. Josef Vejlupek.

[48] Magnetic Encoder Pair Kit for 20D mm Metal Gearmotors, 20 CPR, 2.7-18V. *Pololu.com* [online]. Pololu [cit. 2019-04-28]. Available: https://www.pololu.com/product/3499

[49] WANG, Tianmiao, Yao WU, Jianhong LIANG, Chenhao HAN, Jiao CHEN a Qiteng ZHAO. Analysis and Experimental Kinematics of a Skid-Steering Wheeled Robot Based on a Laser Scanner Sensor. *Sensors* [online]. 2015, **15**(5), 9681-9702 [cit. 2019-05-21]. DOI: 10.3390/s150509681. ISSN 1424-8220. Available: http://www.mdpi.com/1424-8220/15/5/9681

[50] Documentation: SDF Specification. *SDFormat.org* [online]. Open Source Robotics Foundation, 2019 [cit. 2019-05-03]. Available: http://sdformat.org/tutorials?cat=specification&

[51] Gazebo_models. *GitHub* [online]. 25 Apr 2018 [cit. 2019-05-03]. Available: https://github.com/mikaelarguedas/gazebo_models

[52] Physics Parameters. *Gazebosim.org* [online]. Open Source Robotics Foundation, 2014 [cit. 2019-05-03]. Available: http://gazebosim.org/tutorials?tut=physics_params&cat=physics

[53] Urdf/Tutorials. *Wiki.ROS.org* [online]. Open Source Robotics Foundation, 11 Apr 2016 [cit. 2019-05-03]. Available: http://wiki.ros.org/urdf/Tutorials

[54] Xacro: Package Summary. *Wiki.ROS.org* [online]. Open Source Robotics Foundation, 23 Jul 2018 [cit. 2019-05-03]. Available: http://wiki.ros.org/xacro

[55] HOFFMAN, David. *Tvorba simulačních modelů mobilních robotů pro framework ROS.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně. Vedoucí práce Doc. Ing. Stanislav Věchet, Ph.D.

[56] Ubiquity Robotics Downloads: Raspberry Pi Images. *Ubiquityrobotics.com* [online]. 2018 [cit. 2019-05-08]. Available: https://downloads.ubiquityrobotics.com/pi.html

[57] XAVIER, Rodrigo S., Bruno M. F. DA SILVA a Luiz M. G. GONCALVES. Accuracy Analysis of Augmented Reality Markers for Visual Mapping and Localization. In: *2017 Workshop of Computer Vision (WVC)* [online]. IEEE, 2017, 2017, s. 73-77 [cit. 2019-05-12]. DOI: 10.1109/WVC.2017.00020. ISBN 978-1-5386-1451-8. Available: http://ieeexplore.ieee.org/document/8278082/

[58] BLANCO, José-Luis. *A tutorial on SE(3) transformation parameterizations and on-manifold optimization* [online]. In: . University of Malaga, 12-09-2010 [cit. 2019-03-18]. Available: http://ingmec.ual.es/ jlblanco/papers/jlblanco2010geometry3D_techrep.pdf

[59] PETERSEN, K.B. a M.S. PEDERSEN. *The Matrix Cookbook* [online]. In: . Technical University of Denmark, 16 Feb 2008 [cit. 2019-05-21]. Available: http://math.xmu.edu.cn/group/nona/books/mcb.pdf

[60] BROMILEY, P. A. *Products and Convolutions of Gaussian Distributions* [online]. In: . Machester: University of Manchester, 22. 6. 2018 [cit. 2019-05-21]. Available: http://www.tina-vision.net/docs/memos/2003-003.pdf

[61] *GitHub: Ubiquity Robotics* [online]. 2019 [cit. 2019-05-15]. Available: https://github.com/UbiquityRobotics

[62] *Wiki.ROS.org: ROS* [online]. Open Source Robotics Foundation [cit. 2019-05-23]. Available: http://wiki.ros.org/ROS

[63] Gazebo Architecture. *Gazebosim* [online]. Robotics Foundation [cit. 2019-05-23]. Available: http://gazebosim.org/tutorials?tut=architecture&cat=get_started

Appendix

A Electronic Appendixes

The enclosed DVD contains:

- / - root folder with electronic version of this thesis and following subfolders
- /Config_ARuco - calibration files for real and simulated camera
- /control - configuration for package *twist_mux* and control of velocity in simulation
- /launch - all launch files described in the thesis
- /localization - calibration file for IMU
- /map - grid map of simulation environment and ArUco markers map
- /msg - created custom messages
- /navigation - configurations file for *move_base_flex*
- /robot_description - files for creating model of robot
- /rviz - configuration files for visualization
- /SMACH - state machines
- /src - script files
- /srv - created custom services
- /Worlds - created world in Gazebo

B Figures

1. Localization
2. Validation of the estimated parameters for the kinematics model
3. Dependence of measurement error variance on ζ_g: angle from optical axes in horizontal plane

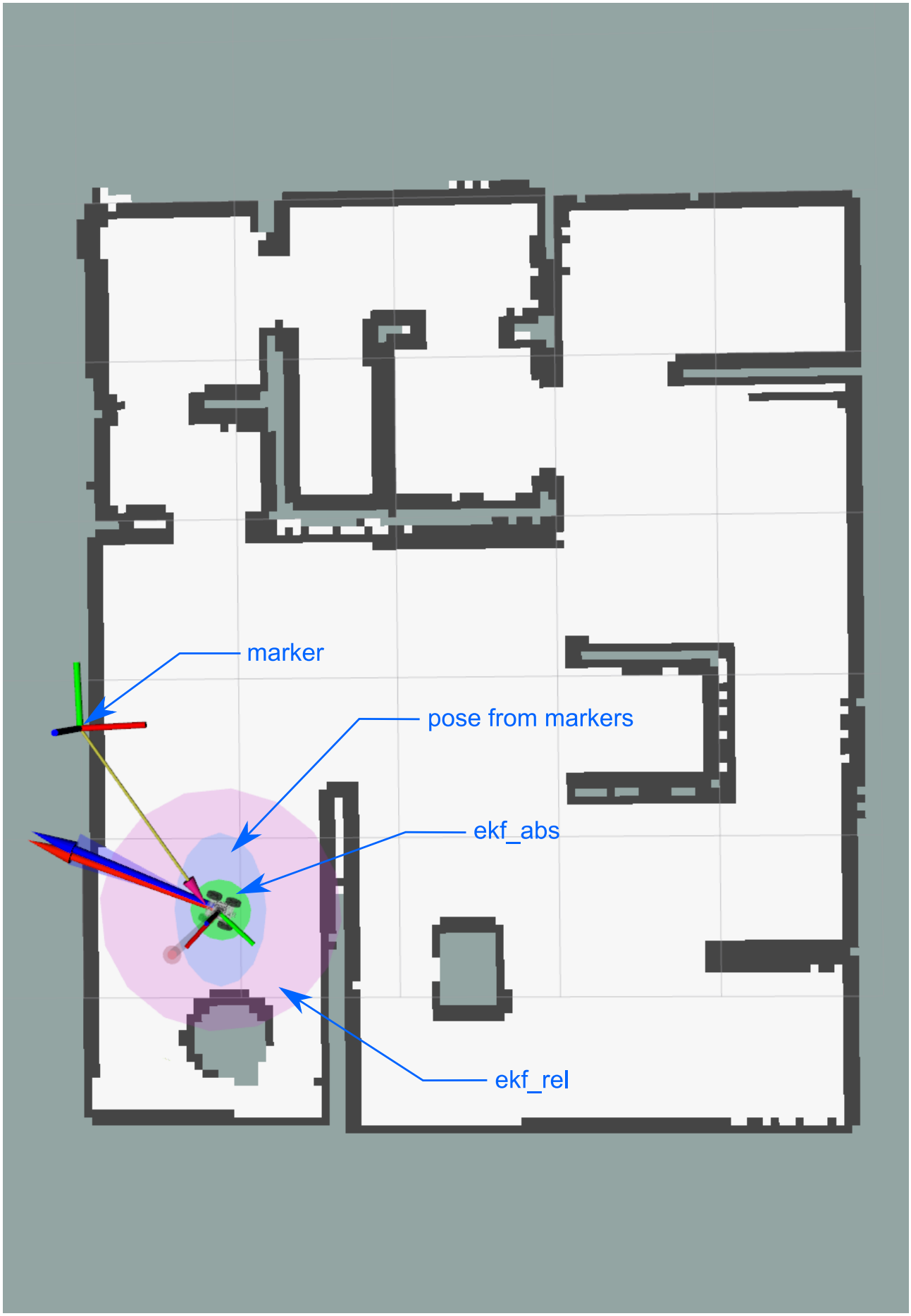


Figure 6.1: Localization: Shown are the ellipses that represents covariance matrix of position of output state from *ekf_abs* and *ekf_rel*, and the ellipse representing covariance matrix of pose from image processing.

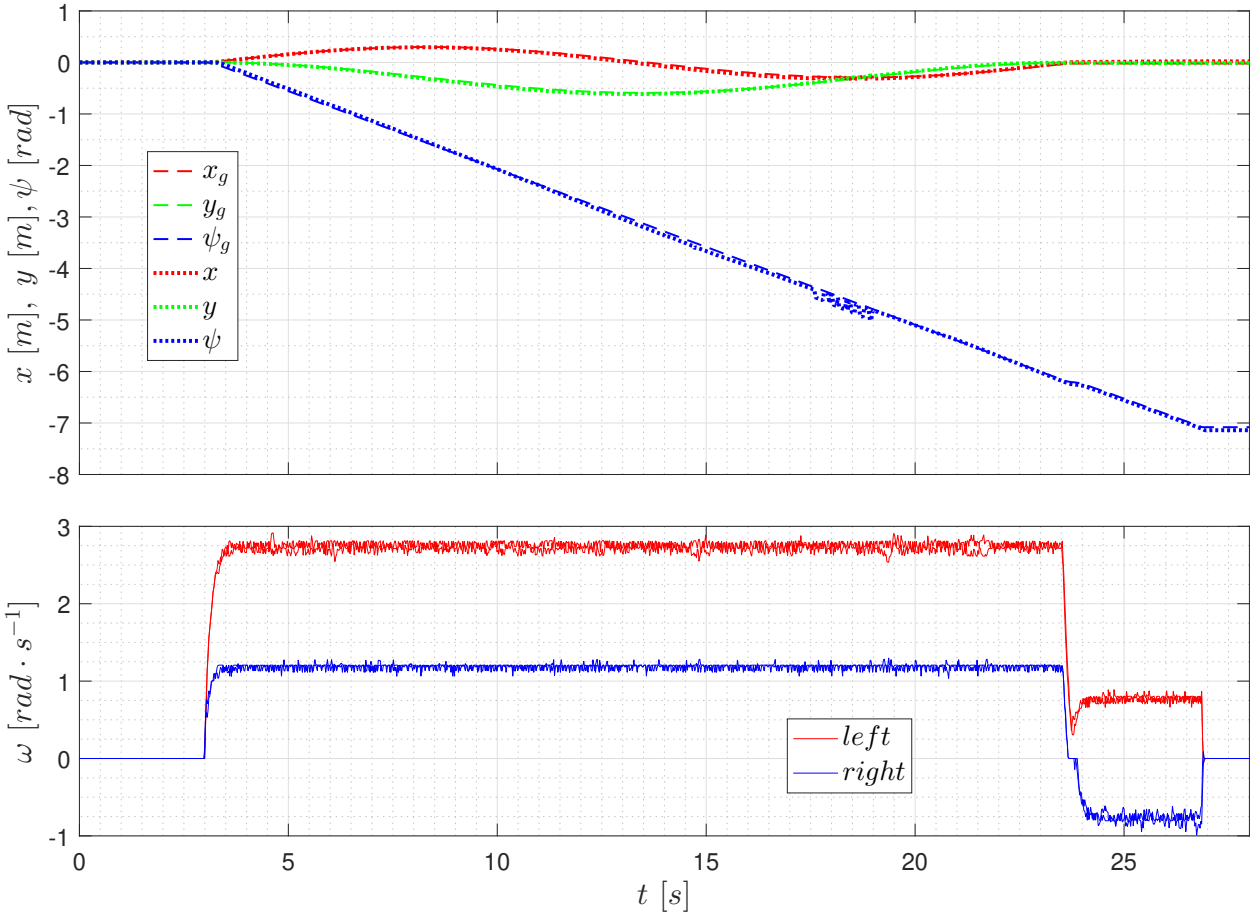


Figure 6.2: Validation of the estimated parameters for the kinematics model

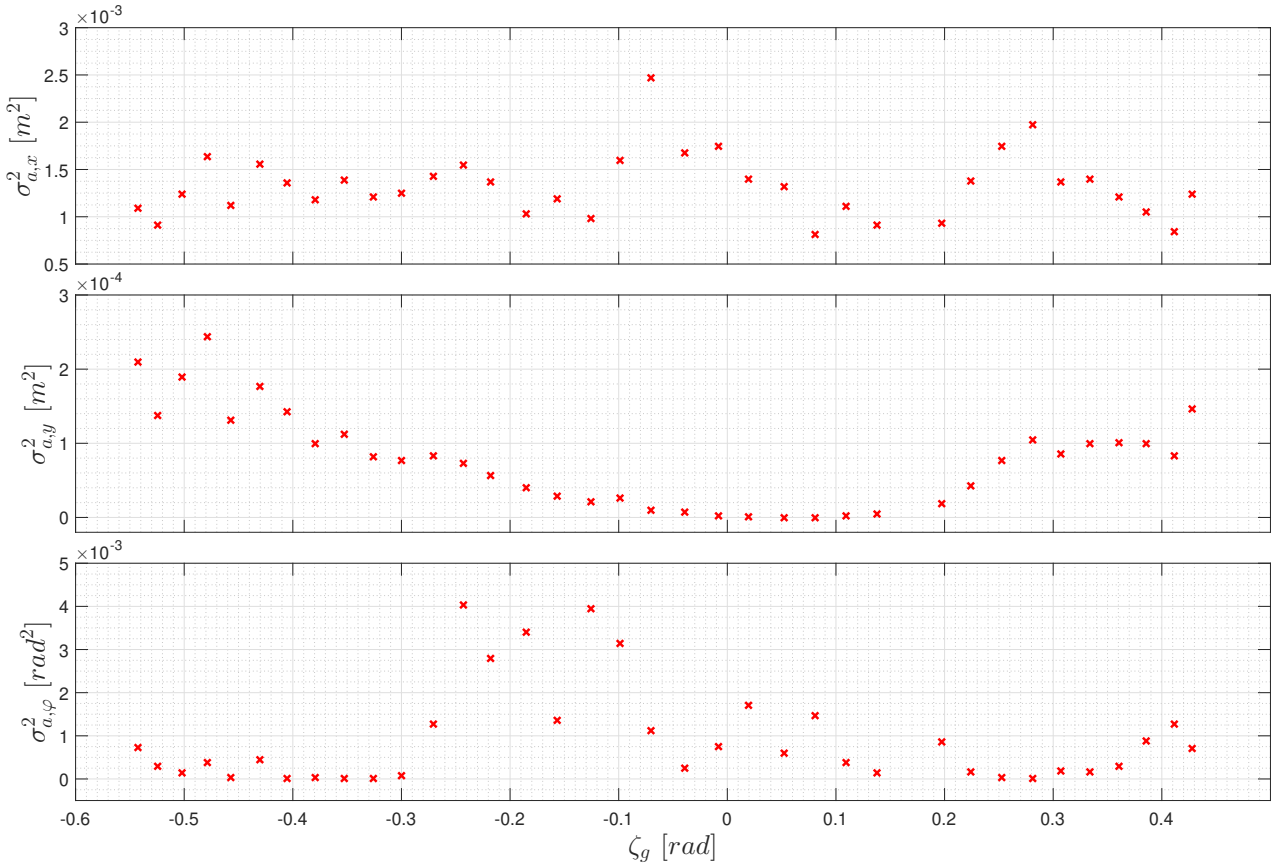


Figure 6.3: Dependence of measurement error variance on ζ_g: angle from optical axes in horizontal plane