DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

Intralogistics with the autonomous transport robot

T3_3100/3200

Course of study: Electrical engineering

Field of study: Automation

Duale Hochschule Baden-Württemberg Ravensburg, Campus Friedrichshafen

from

Alexander Ostermann & Felix Wirl

Submission date:	July 15, 2025
Processing period:	01.10.2024 - 13.01.2025 & 01.04.2025 - 15.07.2025
Student number:	5047110, 4800628
Course:	TEA22
Reviewer:	Pranav Tej Gangavarapu

Declaration

according to 1.1.14 appendix 1 to §§ 3, 4 and 5 of the study and examination regulations for bachelor's degree programs in the field of technology at the Dualen Hochschule Baden-Württemberg from 29.09.2017 in the version from 24.07.2023.

I hereby certify that i have written my T3_3100/3200 on the topic:

Intralogistics with the autonomous transport robot

independently and did not use any sources or resources other than those specified. I also certify that the submitted electronic version corresponds to the printed version.

Friedrichshafen, 15.07.2025



Alexander Ostermann & Felix Wirl

Abstract

The improvement to the Husky robot is intended to make it easier and faster to work with the robot. The first part of the work is about optimizing the Husky. Here, the given sensors are connected in a compliant manner at both hardware and software level. This also includes implementation in the robot operating system. It is important to pay attention to the linking of the coordinate transformations. Furthermore, the network construct of the Husky is optimized by placing the different subnets in a single subnet. Ultimately, the optimization serves as preparation for autonomous product transportation between A and B.

Kurzfassung

Die Verbesserung des Husky-Roboters soll dafür sorgen, einfacher und schneller mit dem Roboter arbeiten zu können. Im ersten Teil der Arbeit geht es um die Optimierung des Huskys. Hierbei werden die gegebenen Sensoren konform angeschlossen, sowohl auf Hardware- als auch auf Softwareebene. Dies schließt auch die Implementierung in das Robot operating system mit ein. Dabei ist es wichtig, auf die Verlinkung der Koordinatentransformationen zu achten. Des Weiteren wird das Netzwerkkonstrukt des Huskys optimiert, indem die unterschiedlichen Subnetze in ein einziges Subnetz gelegt wurden. Letztendlich dient die Optimierung als Vorbereitung für den autonomen Bauteiltransport von A nach B.

Contents

1 Introduction **1**

2 Theoretical background **3**

2.1 RITZ - Regionales Innovations- und Technologietransfer Zentrum GmbH 3

2.2 Zukunftsfabrik - Bodensee 4

2.3 Linux - Ubuntu 20.04 5

2.4 Robot Operating System (ROS) 6

2.5 Simulation software - Gazebo 7

2.6 Visualization software - Rviz 8

2.7 Unified Robot Description File (URDF) 9

2.8 Simultaneous Localization And Mapping (SLAM) 10

2.8.1 Filters 14

2.9 Light Detection And Ranging (LiDAR) 17

2.10 Networking 18

2.11 The Husky 20

2.12 Communication Husky - LiDAR 21

2.13 Communication Husky - PLC (OPC UA) 22

2.14 Computer Vision with OpenCV 23

2.15 Transformation tree Husky 23

3 Problem definition **26**

3.1 Actual state 26

3.2 Problems 26

3.3 Possible solutions 27

4 Implementation **30**

4.1 Networking solution 30

4.2 Coordinate transformation solution 32

4.3 Communication Husky - Zukunftsfabrik Bodensee 34

4.4 Simulating Husky in Gazebo 38
4.4.1 Setup of the simulation environment 38
4.5 Real world problem 42

5 Summary **44**
5.1 Problems during Implementation 44
5.2 Future actions 45

Bibliography **46**

List of Figures **50**

Example code **51**

List of abbreviations **52**

List of Tables **53**

Use of Artificial Intelligence-based Tools **54**

1 Introduction

The use of robots is becoming increasingly popular these days. The reason for this is the opportunity to increase competitiveness in business and society. This can be illustrated by the following figure:

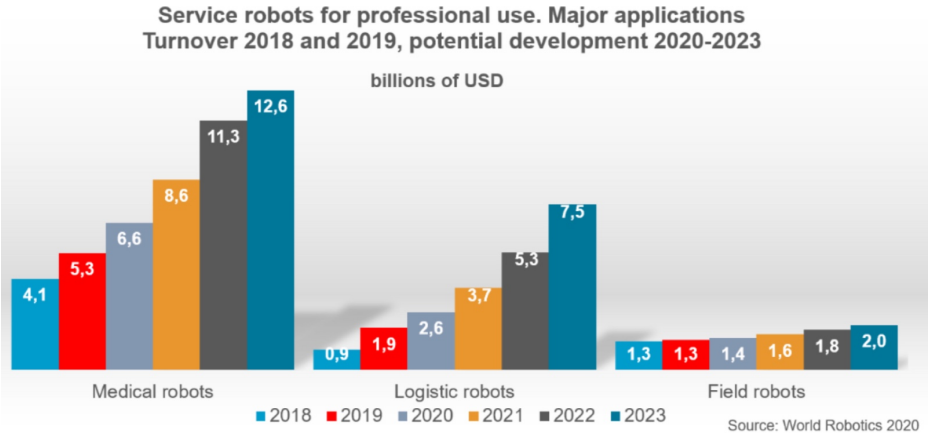


Figure 1.1: Service robots for professional use (cf. Schatton, 2024, p. 187)

Each robot is unique thanks to its sensors and actuators. For example, the sensors provide the necessary information about the system's environment and the actuators ultimately carry out the actions. This allows work steps to be carried out at a higher speed with consistent quality without the system becoming fatigued. The initial high acquisition costs are compensated for in the long term by the work performance of the robots, which means that human labour is increasingly being taken over by robots. Robots are therefore cheaper than human labour in the long term. (cf. Schatton, 2024)

The word robot comes from the Czech word "robota", which means forced labor, or from the late middle high German "robater", which means worker in forced labor. The term has its origins in science fiction stories. It was first used by the Czech writer Karel Capek in his play "R.U.R." (Rossumovi Univerzalni Roboti) in 1920. It

describes how artificial human-like automatons are created to take over human labor, but revolt against it. (cf. Dillmann and Huck, 1991)

Most robots used in industry are manipulators that are used at a fixed workstation and are not mobile. These industrial robots only work in a structured, controlled environment.

Mobile robots are fundamentally different and characterized by the following features:

- Adaptation to changes in its environment
- Learning from experience and corresponding behavioral change
- Development of an internal world view
- Independent planning and execution of complex tasks in an unknown environment (navigation planning)
- Manipulation of physical objects in the real world

The field of robotics has links to various disciplines, including computer science, electrical- (sensor technology) as well as mechanical engineering (kinematics and mechanical design), mathematics (statistical evaluation and error calculation), AI (artificial intelligence) and cognitive sciences. (cf. "Digitalisierung der Baustelle: Clever bauen mit mobilem Roboter" 2020).

The Husky as a mobile robot is the center of the work, focusing on the connectivity of the unit, for example, how a simple connection between user and robot can be created without many individual steps for registration. The aim of this work is to automatically register the robot in order to simplify the start up process after the start button is pressed. Within the second phase simulating the Husky will be implemented. Furthermore solutions to communicate with the „Zukunftsfabrik Bodensee" are presented.

2 Theoretical background

2.1 RITZ - Regionales Innovations- und Technologietransfer Zentrum GmbH

The Regional Innovation and Technology Transfer Center (RITZ) is a competence center for digitalization and mobility in Friedrichshafen. It provides the innovation structure and network in which business and science work together on the future of digitalization and mobility. This bundling of knowledge combined with the active networking of the various players increases the innovative strength. The main goal of the RITZ is to create a vibrant innovation community. The regional proximity to internationally well-known companies such as ZF Friedrichshafen or Rolls-Royce Power Systems, as well as the close proximity to Zeppelin University and the Baden-Württemberg Cooperative State University, provides the RITZ with an innovative and knowledge-promoting environment. In summary, the RITZ is a platform for companies, research and educational institutions to work together on forward-looking projects in the field of digitalization and mobility. With a modern infrastructure and a broad network, it offers comprehensive support for innovative projects and future-oriented cooperation in the region. (cf. Technologietransfer Zentrum GmbH, 2025)



Figure 2.1: Illustrative representation of RITZ (cf. *Ein Leuchtturm mit überregionaler Strahlkraft* 2024; *ZPD Zentrum für Digitalisierung in Produktion und Produktentwicklung* 2023)

2.2 Zukunftsfabrik - Bodensee

The Zukunftsfabrik Bodensee on the Fallenbrunnen campus in Friedrichshafen is a key research location for the Center for Digitalization in Production and Product Development (ZDP). Here, different research projects work together to create new ideas and make progress. The core of the factory is a conveyor system with sensors that deliver the product right on time to the four designated stations. (cf. *ZPD Zentrum für Digitalisierung in Produktion und Produktentwicklung* 2023)

To start off, our system gets the particular product version through a QR reader. A collaborative robot retrieves the product from a designated spot and places it on a workpiece carrier that runs through the conveyor system. At the second station, the right bolts for each product are loaded onto the workpiece carrier using bin picking (automatic parts removal from a container). The third station is an ergonomically optimized workstation for manual work. This is where the final assembly of the pre-product takes place. These listed work steps take place in the Produktions Technologie

Zentrum (PTZ). A mobile robot transports the product to the RITZ, where the final assembly is carried out by an automated screwing station. (cf. *ZPD Zentrum für Digitalisierung in Produktion und Produktentwicklung* 2023)

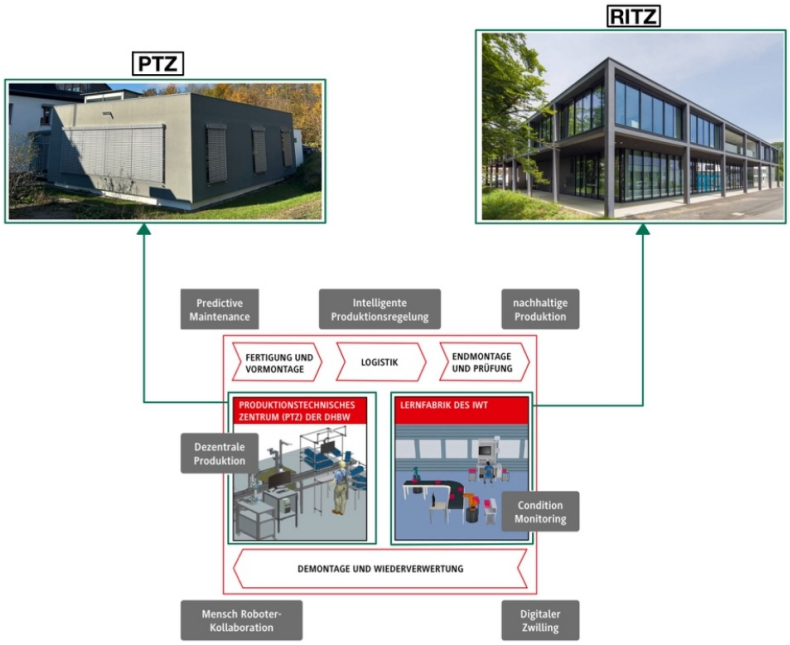


Figure 2.2: Illustrative representation of Zukunftsfabrik Bodensee (cf. *Ein Leuchtturm mit überregionaler Strahlkraft* 2024; *ZPD Zentrum für Digitalisierung in Produktion und Produktentwicklung* 2023)

2.3 Linux - Ubuntu 20.04

Linux for itself is an open-source Operating System (OS) based on the Linux kernel. There are many Linux distributions available that uses different tools to suit the requirements of the given use case. Furthermore the installation and usage of Linux is completely free. (cf. *Linux* 2024)

As mentioned the Husky uses the Linux system with the Ubuntu distribution due to its pricing, simplicity regarding robotic development and its extensibility with open source software. (cf. Ubuntu, 2024)

2.4 Robot Operating System (ROS)

ROS (Robot Operating System) is an open-source framework designed to simplify the development of robotic applications. Its primary goal is to enable developers to build complex robotic systems without having to start from scratch. To achieve this, ROS promotes the sharing of code, algorithms, libraries, and ideas within the robotics community.

ROS offers a wide range of ready-to-use algorithms and drivers, making it easier to develop autonomous robots, such as the Husky. It collects data from connected sensors and uses modular components called nodes to process and manage that data. Each node is designed to perform a single, specific task reliably—for example, one node might handle visual object detection, while another executes motion planning algorithms.

All nodes communicate through the ROS Master, which acts as a central coordinator. The ROS Master manages node registration and facilitates communication between them, allowing nodes to discover and interact with one another across a network. (cf. p. 419f Blubaugh et al., 2022)

In the following illustration (Figure 2.3) the structure of ROS is described.

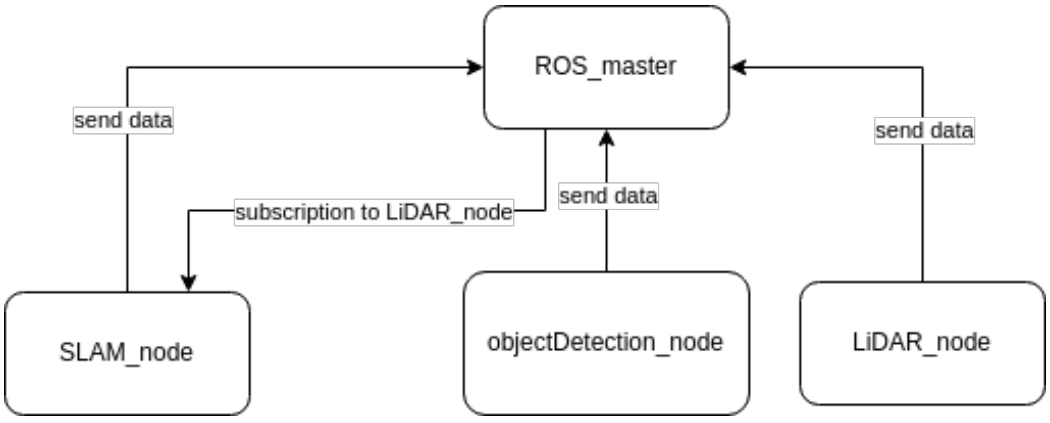


Figure 2.3: Illustrative representation of ROS

Furthermore the ROS system consists of executable packages that can either be developed by the user or can be downloaded as open source software. These packages are a collection of python or C++ source code, launch files, world models or robotic description files. To execute nodes within ROS, the rosrun command is used, allowing users to run individual ROS programs. If developers need to explore available packages or topics, they can query them using the rostopic command, which lists all active topics within the ROS system.

For more complex setups, ROS provides roslaunch, a powerful tool designed to start multiple nodes simultaneously. Launch files—written in XML format—define which nodes and parameters should be executed together, simplifying the process of starting interconnected components. This is particularly useful when deploying autonomous robots that require multiple functional modules to work seamlessly. Additionally, ROS supports scripts to automate tasks, often using Python or Bash to streamline operations like data processing and sensor calibration. In ROS communication, topics serve as the primary mechanism for data exchange. Nodes use a publish-subscribe model, where publishers send messages to specific topics while subscribers listen for relevant data. For more direct interactions between nodes, ROS services enable request-response communication. Unlike topics, services allow nodes to send a request and wait for a reply, making them ideal for tasks requiring immediate feedback, such as adjusting motor speed or retrieving sensor data. (cf. p. 426f. Blubaugh et al., 2022)

Based on given information ROS has the advantage to save time and money during development based on the open source infrastructure.

2.5 Simulation software - Gazebo

The Gazebo simulation environment is an open-source simulator that is used for the simulation of robotic systems. The system is based on the ROS framework, which has already been described in subsection section 2.4. Gazebo is used for various types of simulations, such as those involving mobile robots, stationary robot arms and complex

systems with large numbers of sensors. These examples can be simulated in 3D environment. In addition to actuators and sensors, the robot simulation environment can display shadows, textures, objects, and light. Simulations in Gazebo are carried out using plug-ins. The „Gazebo_ros_pkgs“ package connects Gazebo with ROS and maps the hardware of an existing system in Gazebo. (cf. Kriegel et al., 2025)

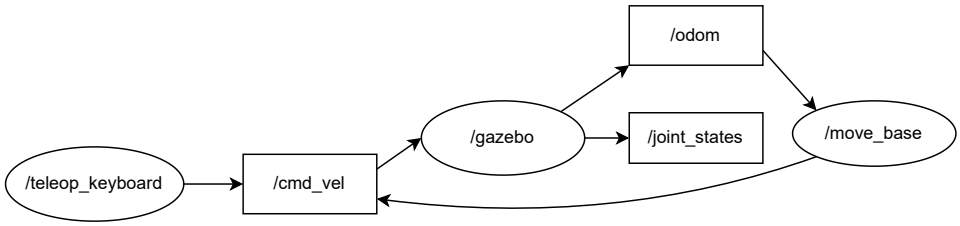


Figure 2.4: Illustrative representation of ROS - Gazebo connection (cf. p. 2 Kriegel et al., 2025)

The figure Figure 2.4 shows a section of a ROS network with an integrated Gazebo simulation. The availability of Gazebo implementations and the high compatibility between ROS and Gazebo justify the choice of this simulator.

2.6 Visualization software - Rviz

Rviz (ROS Visualization) is a 3D visualization platform within the Robot Operating System (ROS). It offers the ability to visualize and display various types of data, including image, map, and robot model data. This 3D visualization tool is commonly used to display sensor and status data in real time. Its compatibility with various robotics platforms based on the ROS software framework is also notable. Rviz is used for the development, visualization, and error analysis of complex robot systems in simulated or real environments. These applications cover the fields of education, research, and industrial robotics.

The sensor and status data are ROS data, which includes LiDAR scans, point clouds, camera images, paths, and map information.

Typical use cases include visualizing robot models described via URDF files and representing spatial coordinate systems via the tf framework. These models can display objects in terms of their position, joints, size, or mass on a surface or in 3D space. (cf. Han, 2021)

Rviz allows users to display navigation information, such as SLAM maps and obstacle detection. When used with the Gazebo simulation environment, Rviz analyzes autonomous systems. For instance, when connected to a Husky transport robot, Rviz evaluates live LiDAR data in real time and visualizes the robot's movements. (cf. Koenig and Howard, 2004)

The 3D platform is used for the scientific validation and development of algorithms, such as SLAM (simultaneous localization and mapping), multi-robot coordination, and human-robot interaction. (cf. Garzon et al., 2017)

2.7 Unified Robot Description File (URDF)

Due to the complexity of robot systems it is effective to simulate them in a secure environment before deployment. To do this, the kinematics as well as the dynamics of the robot are needed. Therefore a formatted file is the solution. Unified Robot Description File (URDF) is a formatted file written in XML with the extension „.urdf“. This file allows developers to model the robot and visualize or simulate it with various tools. (cf. Tola and Corke, 2024)

Source code 2.1 shows an excerpt from a URDF file as an example:

Source code 2.1: Sample excerpt from URDF file

```
1   <!-- Base link is the center of the robot's bottom plate -->
2   <link name="base_link">
3     <visual>
4       <origin xyz="0 0 0" rpy="0 0 0" />
5       <geometry>
```

```
6           <mesh filename="package://husky_description/meshes/
                base_link.dae" />
7         </geometry>
8       </visual>
9       <collision>
10        <origin xyz="0 0 ${base_z_size/4}" rpy="0 0 0" />
11        <geometry>
12          <box size="${base_x_size} ${base_y_size} ${base_z_size
                /2}"/>
13        </geometry>
14      </collision>
15      <collision>
16        <origin xyz="0 0 ${base_z_size*3/4}" rpy="0 0 0" />
17        <geometry>
18          <box size="${base_x_size*4/5} ${base_y_size} ${base_z_size
                /2}"/>
19        </geometry>
20      </collision>
21    </link>
```

In this URDF fragment, a robot link named base_link is defined, which represents the central part of the robot's base. The visual appearance of this link is specified using a mesh file (base_link.dae) located in the husky_description package. This mesh is used purely for visualization in simulation or Rviz. In addition to the visual element, two separate collision geometries are defined to approximate the physical shape of the base for collision detection and physics simulation. Following this concept, the entire robot is modeled similarly to enable full simulation and testing in a virtual environment.

2.8 Simultaneous Localization And Mapping (SLAM)

Simultaneous Localization And Mapping (SLAM) describes the process by which mobile robots generate maps and determine their position within them. The robot creates

a map of its environment and determines its position within it. This is a common problem in mobile robotics. (cf. Pavlovic, 2016)

At the start of the process, the initial position and environment of the robot are unknown. The following illustration (Figure 2.5) provides a description. (cf. Durrant-Whyte and Bailey, 2006)

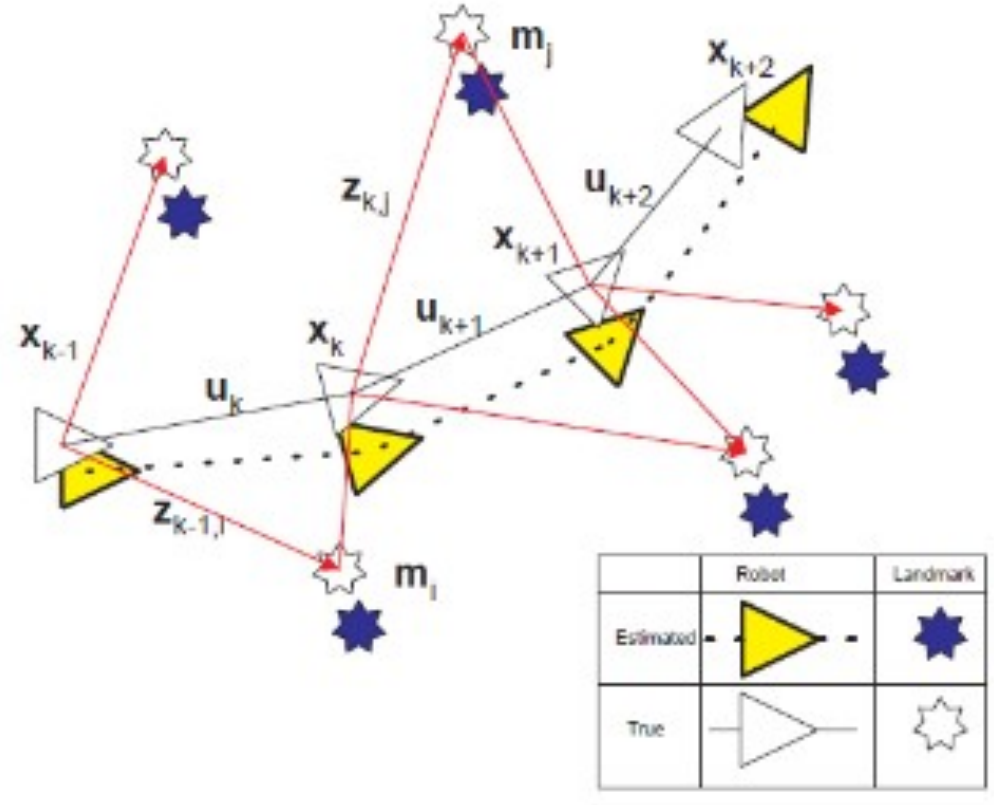


Figure 2.5: Estimation of robot position and landmarks (cf. p. 2 Durrant-Whyte and Bailey, 2006)

Landmarks, whether natural or artificial, are used to describe the surroundings. The focus is on determining the positions of the landmarks and one's own position relative to them. (cf. Aulinas et al., 2008)

The robot uses sensor measurements and control commands to perform these actions. Therefore, a motion and observation model is useful. However, in reality, this is

subject to inaccuracies and noise. This is why probability theory is used to solve the SLAM problem. This means that the position of the robot or landmarks is unknown or inexact. They are described by estimates based on probability distributions. (cf. Pavlovic, 2016)

Another problem is the data association problem. This means that each time a landmark is observed, the respective algorithm must be able to identify it uniquely. Whether it is an observable or new landmark is an important decision. If the SLAM algorithm cannot solve this problem, it will fail. (cf. Montemerlo and S. Thrun, 2007)

The SLAM formulation is based on a movement and observation model that is used to determine the robot's position and generate a map.

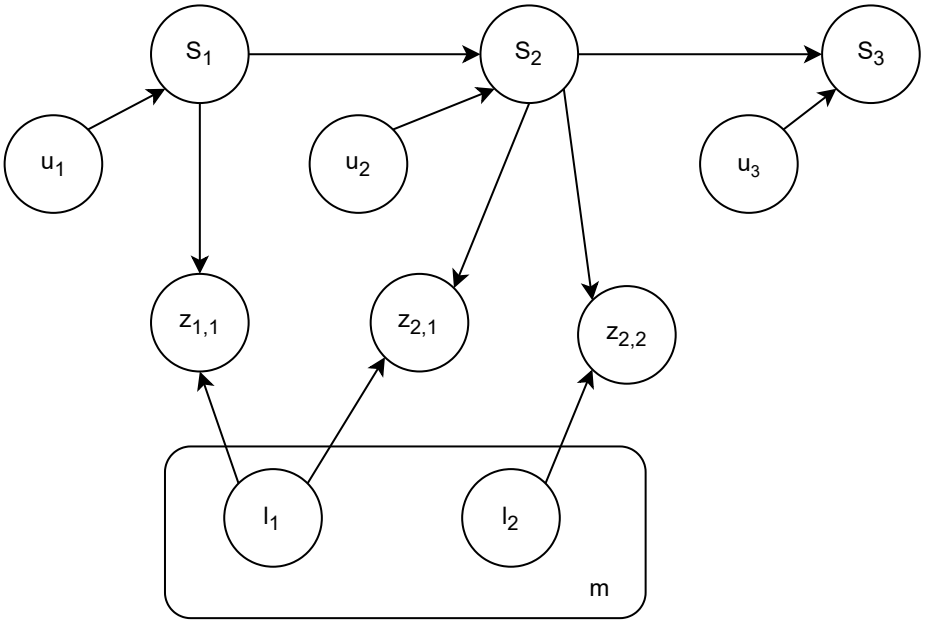


Figure 2.6: Relationship of the process variables (cf. p. 3 Chen, 2013)

The two models form the relationships between the process variables, as shown in Figure 2.6. The SLAM is shown here as a dynamic Bayesian network. (cf. Chen,

2013)

Description of the Static Dependency Between the Variables: This dependency is illustrated in Table 2.1. The individual dependencies are shown in the following list: To clarify: Condition 1 depends on Condition 2.

Table 2.1: Dependencies

Condition 1	**Condition 2**
Robot position s_t	Last position s_{t-1}
Measurement Sensor z_t at time t	Current control command u_t
	Current robot position s_t

Mathematical description of the dependencies:

$$s_t = f(s_{t-1}) + \epsilon \tag{2.1}$$

The Robot's State Transfer Function is described in Equation 2.1. The robot's current position (st) is calculated based on its previous position (st-1) at time t-1 and the last control command (ut). Function f is the robot's odometry function and is generally non-linear. The additive noise component, ϵ, describes uncertainties in the robot dynamics. (cf. Chen, 2013)

$$z_t = h(st) + \delta \tag{2.2}$$

The measurement sensor function is described in Equation 2.2. It describes the observation made at time t based on the robot position at that time. Function h is a sensor function subject to additive measurement noise δ. (cf. Chen, 2013)

It describes the observation made at time t based on the position of the robot at that time. The function h is the function of the sensor and is subject to additive measurement noise, δ. In the case of the extended Kalman filter, equations Equation 2.1 and Equation 2.2 are linearized via Taylor expansion and inserted into a multivariate

normal distribution of the form (Equation 2.3, (cf. Sebastian Thrun, Burgard, and Fox, 2000))

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\} \tag{2.3}$$

where the expected value μ and covariance matrix Σ. The motion model as:

$$p(s_t, m \mid s_{t-1}, u_t) \tag{2.4}$$

and the observation model as:

$$p(z_t \mid s_t, m) \tag{2.5}$$

List of problems that lead to uncertainty with SLAM:

- Noisy measurements due to sensors (LiDAR)
- No exact predictability of the robot movement

Filters counteract these problems and provide a statistical estimate of the true status in terms of position, orientation, and map. More information is provided in a list in subsection 2.8.1. (cf. Sebastian Thrun, Burgard, and Fox, 2000)

2.8.1 Filters

Filters are used to estimate the state of a system based on noisy measurements. Overview of various filters:

- Extended Kalman Filter (EKF-SLAM)
- Sparse Extended Information Filter (SEIF)

- Particle Filter
- FastSLAM
- DP-SLAM (Distributed Particle SLAM)

It's important to note that each algorithm has its own set of advantages and disadvantages depending on the application environment and available resources.

Extended Kalman Filter (EKF-SLAM):

The EKF filter is a special form of the Gaussian filter and estimates the robot position and the position of landmarks. To model the uncertainty using a covariance matrix, the relationships between the robot position and the landmarks, as well as between the individual landmarks themselves, are defined. Suitable for smaller environments with fewer landmarks. (cf. Aulinas et al., 2008)

Table 2.2: Advantages and Disadvantages of EKF SLAM

Advantage	**Disadvantage**
Effective exploration	Computational complexity increases with more landmarks
Reliable establishment	Sensitive to inconsistencies in nonlinear movements
Suitable for small environments	

Sparse Extended Information Filter (SEIF):

To reduce computational effort and memory requirements, a variant of the EKF, the SEIF, was developed. Using the information matrix H and the information vector b achieves the transition to the information form. The information matrix H contains data on the relative positions of the landmarks. This means that landmarks closer together have a stronger relationship than landmarks farther apart. Therefore, the reduction in computational effort is achieved by discarding the weak landmark relationships from the information matrix. (cf. Sebastian Thrun, Burgard, and Fox, 2000)

Table 2.3: Advantages and Disadvantages of SEIF

Advantage	Disadvantage
Suitability for large environments	Loss of accuracy due to approximation methods
Constant update time	

Particle Filter:

Particle filters allow flexible modeling of complex distributions. The filter has the ability to work with multimodal probability distributions and nonlinear motion and observation equations. (cf. Montemerlo and S. Thrun, 2007)

Table 2.4: Advantages and Disadvantages of Particle Filter

Advantage	Disadvantage
Very robust	Computationally intensive; increases steadily with complexity
Flexible with complex movement models	Unsuitable for high-dimensional state spaces

FastSLAM:

This SLAM algorithm uses particle filters to determine the robot's pose and extended Kalman filters to calculate landmark positions. This allows for efficient estimation in large environments. (cf. Montemerlo and S. Thrun, 2007)

Table 2.5: Advantages and Disadvantages of FastSLAM

Advantage	Disadvantage
Less storage space Shorter computing times Good scaling	Particle filters can degenerate

DP-SLAM (Distributed Particle SLAM):

The DP-SLAM uses a single grid map for all particles. This map is used for modeling robot pose and map structure. (cf. Eliazar and Parr, n.d.)

Table 2.6: Advantages and Disadvantages of DP-SLAM

Advantage	**Disadvantage**
No data association problem Suitable for environments with uncertain landmarks	High memory requirements High computing time Complex implementation

2.9 Light Detection And Ranging (LiDAR)

A LiDAR is a measurement device that is capable of measure the distance, reflection and scattering of objects by sending electromagnetic (EM) waves at every direction. It is possible due to the reflection and scattering characteristics of objects which will be hit by the EM waves. Furthermore LiDAR technology uses the infrared as well as ultraviolet EM waves to detect the objects. In the case of automotive driving the measurement of the distance is commonly used to get a three dimensional cloud of dots. Each dot has the information about its signal strength which can be used in combination with enough points to mimic the shape of an object. With the help of these data it is possible to label the objects and create a part of a 3D map with different categories. (cf. Beuth, Parl, and Gotzig, 2024)

The Husky has two different LiDAR sensors to ensure 360 degree detection of objects. At the front top there is a three dimensional (3D) LiDAR from Velodyne. One 3D LiDAR would be enough but it is not possible to cover the complete surrounding of the Husky caused by its physical structure. In other words the construct of the robot would block the LiDAR measurement system in front of itself. Therefore a second LiDAR system is installed at the opposite side as well as at a different height as the 3D LiDAR to ensure 360 degree object detection. Furthermore the second LiDAR is a two dimensional (2D) measuring system from Hokuyo. A set of 2D points in space is generated by moving or rotating measurement beam at one level. Thereby it is possible to calculate the angle and distance to objects (cf. left LiDAR Figure 2.7). On the other hand it is possible to generate a 3D point cloud by moving and sensing more than one measurement beam in order to calculate the x- y- and z-axis (cf. right LiDAR Figure 2.7). On the illustration Figure 2.7 the differences of both system

functionalities are briefly described.

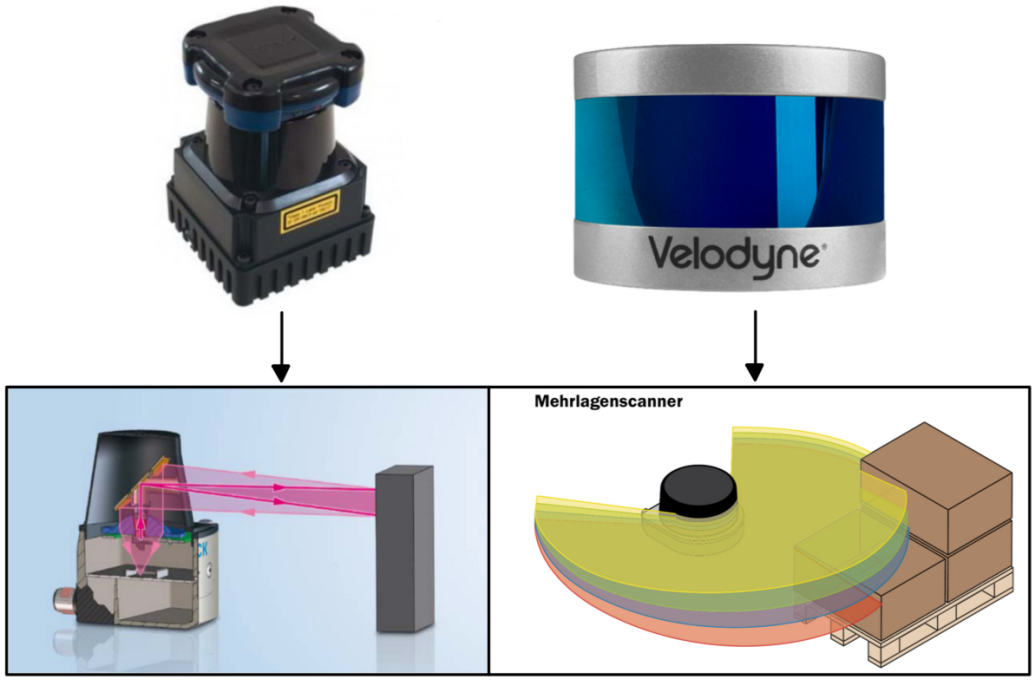


Figure 2.7: Illustrative representation of LiDAR systems (cf. elektronik, 2024; Hexagon, 2024; SICK, 2018)

2.10 Networking

Due to the Husky PC has two LiDAR sensors that are connected via ethernet to the PC it is necessary to configure the networking right in order to get the data from the sensors. Therefore the sensors as well as the ports of the PC must configured with a static IPv4 (internet protocol version four) address. Based on the number of devices the use of only IPv4 addresses is more than sufficient. In the matter of fact the IPv4 address has a length of 32 bit as outlined below. Therefore it is possible to connect nearly 2^{32} devices together.



Figure 2.8: Husky PC

An IPv4 address is a tuple with the length of 32 bit which contains the number of the network as well as the number of the host. Each number is written in decimal system between 0 and 255. The numbers are separated by a dot like „192.168.131.1“. Thereby many hosts are able to have the same network number but only one host number in one network is allowed. (cf. p. 43 Mandl, 2018)

It is possible to separate one network in multiple subnets to stop the waste of ip addresses in one network. To configure multiple subnets the network- and host number is split into subnet- and host number. To define a subnet in a network a subnet mask is needed to restrict the subnet into preferred range. The subnet mask is similar to the IPv4 address, four blocks of decimal numbers between 0 and 255. In order to get the range of the subnets a bitwise AND operation is needed. As a result of the operation the third block of the ip address describes the actual subnet. Because the Husky has the ip address of „192.168.131.1“ the subnet is „131“. (cf. p. 48f. Mandl, 2018)

From the information above it is more easy to configure all devices into the same subnet rather than working with multiple subnets.

2.11 The Husky

The Husky A200 transport robot from Clearpath Robotics is a development platform for demanding outdoor applications. It is characterized by modern and simple design, as well as a robust construction. Thanks to autonomous programming, this platform is used to autonomously perceive environments and recognize obstacles to plan routes and explore unauthorized areas. The Husky's mobility is ensured by the four wheels, whereby it can carry a load capacity of 20-75 kg and an incline of 45°, it can reach a speed of up to 1 m/s. The precise detection of the unknown environment is given through a multitude of sensors such as LiDAR and GPS. (cf. MYBOTSHOP, 2025)

The following features demonstrate the uniqueness of the robot:

- Compact ROS development platform
- Modularly expandable
- Optimized for outdoor applications

The optimization and expandability of the transport robot is made possible by the company MYBOTSHOP. The system has already been equipped with sensors such as 2D LiDAR from HOKUYO and 3D LiDAR from Velodyne to provide highly accurate information about the surroundings. This equipment provides a good prerequisites for navigation and mapping applications. By extending the transport robot with a frame, the total height is 0,74 m. The orange holders on the woden plate of the frame allow components to be placed and removed. (cf. MYBOTSHOP, 2025)

The extended version of the Husky can be seen in the following illustration.

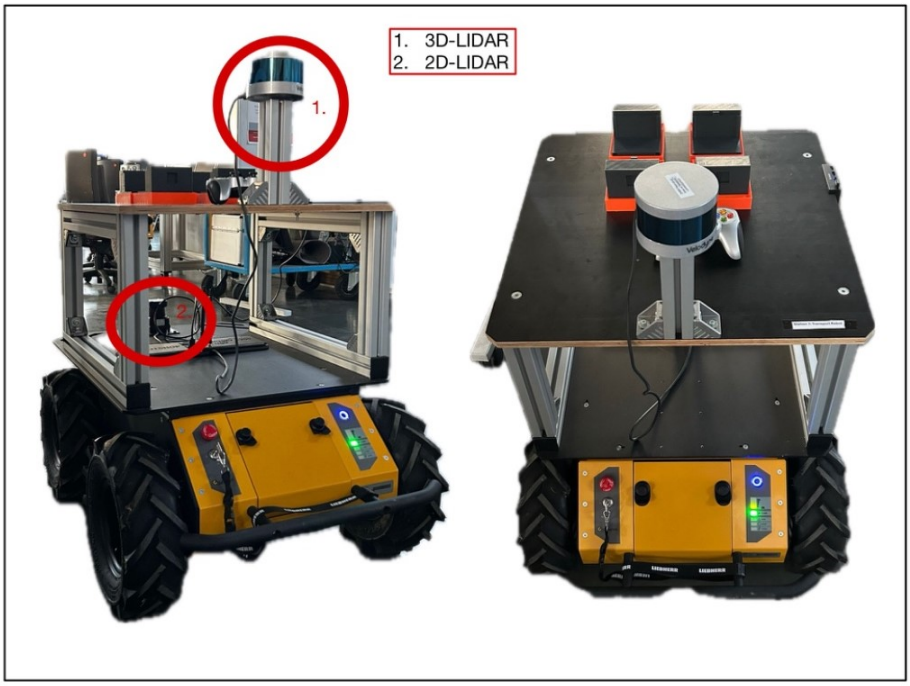


Figure 2.9: Representation of the Husky

2.12 Communication Husky - LiDAR

Both LiDAR sensors use the IP protocol to send their data to the PC. The Internet Protocol is what represents the network layer in communication technology. This protocol is responsible for translating the addresses into the IP addresses used by the network layer. This is important because devices only know each other's IP addresses, not their unique Media Access Control (MAC) addresses, which each network access receives. (cf. p. 28 Mandl, 2018)

The IP address is linked to the MAC address using the Address Resolution Protocol (ARP). When a host sends a request to a destination host with which it has not communicated for a period of time, an ARP request is sent over the LAN (Local Area Network). The ARP request basically asks everyone on the network if they know the IP address in the request. Other devices on the network that know the IP address

will respond with an ARP reply, passing the MAC address of the destination device. Alternatively, the destination device can send its MAC address itself. The fact that the MAC addresses are stored in an ARP cache for a period of time, eliminates the need to retrieve these again for each communication. (cf. p. 112 Mandl, 2018)

Once the IP and MAC addresses have been assigned and the devices are connected, a self-contained package can be sent over the network using Internet Protocol version 4 (IPv4). If the data is too large, the packages must be broken up and sent one at a time. Furthermore the IPv4 protocol does not guarantee package delivery, so data may be lost. (cf. p. 41 f. Mandl, 2018)

2.13 Communication Husky - PLC (OPC UA)

Open Platform Communications Unified Architecture (OPC UA) is a communication protocol designed for industrial environments, enabling data exchange between machines and systems. It belongs to the domain of Operational Technology (OT), which focuses on the control and monitoring of physical processes.

A key feature of OPC UA is its use of a structured address space, which functions like a directory. This structure organizes the data provided by the server and can include not only raw data points but also additional metadata—such as the unit of measurement or data type. This allows OPC UA to go beyond simple data transmission, it enables machines to interpret the data meaningfully by providing semantic context. As a result, values like control variables, sensor readings, or configuration parameters are not just transferred, but also described in a way that is understandable for both humans and machines. (cf. p. 139 Plenk, 2024)

2.14 Computer Vision with OpenCV

According to the statistics of (Statista, 2025), the market share of the use of computer vision for pictures in 2020 has risen from 4.16 billion euros to around 14.01 billion euros today. Looking into the future, this figure is expected to rise to around 31.58 billion euros by 2030.

Therefore, the use of computer vision is becoming more and more important in the field of robotics. Computer vision is a field of computer science that deals with the analysis and interpretation of images and videos. It enables machines to understand and interpret visual information, similar to how humans perceive the world through their eyes. Computer vision algorithms can be used to detect objects, recognize patterns, track movements, and extract meaningful information from visual data. This technology is widely used in various applications, including autonomous vehicles, robotics, surveillance systems, and medical imaging. (cf. p. 9f. Corke, 2023)

OpenCV (Open Source Computer Vision Library) is a popular open-source library (ca. 18 million downloads (OpenCV Team, 2025, cf.)) that provides a wide range of tools and functions for computer vision tasks. It is widely used in the field of robotics and image processing due to its extensive functionality and ease of use. OpenCV provides a comprehensive set of algorithms and functions for image processing, object detection, feature extraction, and more. It supports various programming languages, including Python, C++, and Java, making it accessible to a wide range of developers. OpenCV is particularly useful for tasks such as image filtering, edge detection, image segmentation, and object tracking.

2.15 Transformation tree Husky

In order to locate objects as well as the Husky itself, everything needs a coordinate system. When humans want to locate a tree in front of them, they say „the tree is 50m in front of me“. They tell the axis by giving the direction and the distance

by telling how far away the tree is in that particular direction. If another person is 10m to the right of the first person, the first person can only tell how far away the tree is from the second person if he knows the distance between them. In analogy to the robot, it is almost the same. The center of the robot is mostly at the center of mass of the robot and has the coordinates $\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$. To get the distance from the 3D LiDAR mounted on the front top of the Husky to the base or even at the front of the robot, it is necessary to tell the robot the transformation between the center point and the center of the LiDAR sensor. In addtion to that sometimes a orientation is also needed, which is given by the Euler angles in the rotation matrix. But the LiDAR is mounted in a way that the x-axis is pointing in the same direction as the robot, so the rotation matrix is the identity matrix. Therefore the transformation vector is only a translation vector.

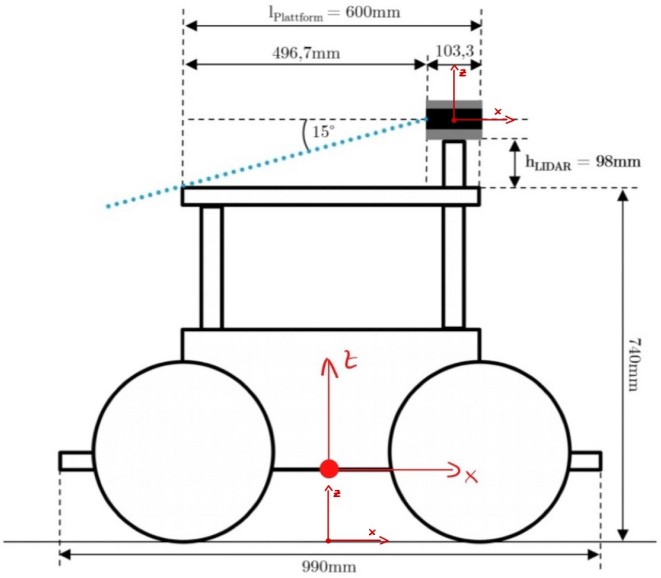


Figure 2.10: Illustrative representation of Husky (cf. Schmid and Straßburger, 2024)

From the Figure 2.10, the distance from the center point to the center of the LiDAR in x-direction can be calculated as 248.35mm. Therefore the distance in z-direction would be 910.7mm. The y-direction does not change, because the center of the Li-

DAR is at the center of the Husky in y-direction. Therefore the translation vector is $\begin{bmatrix} 248.35 \\ 0 \\ 910.7 \end{bmatrix}$ mm. (cf. Rieber, 2024, 4. Koordinatensysteme)

In general, the transformation tree in ROS looks like this:

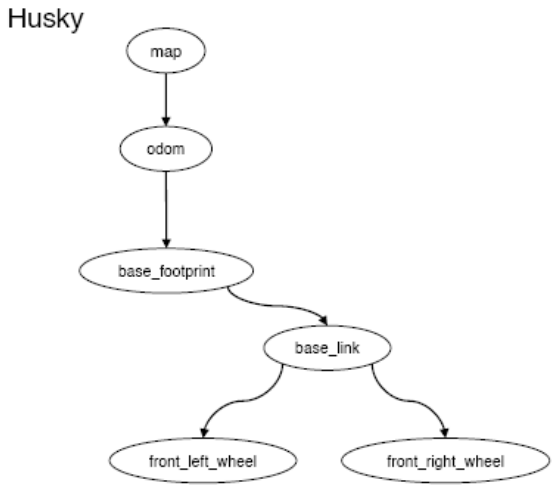


Figure 2.11: Part of Husky transformation tree

Each ellipse describes a frame and the arrows are the transformations between the frames.

Transforms are very important, because otherwise the calculation has to be done with trigonometry, which becomes too complex. Therefore the relationship between coordinate frames are maintained by the tf-package in ROS. (cf. Wiki, 2025)

3 Problem definition

3.1 Actual state

Before making any changes to the robot, the previous student's paper was read and understood. This is essential to understand the actual physical as well as the software construct of the Husky. Actually, the 2D laser is implemented in software and physical. On the other hand, the 3D LiDAR is only physically and accessible through some routing commands like the students described in their thesis earlier. In addition, the Velodyne LiDAR is on a different subnet, which should not be the case. Right now, the robot is only controllable through a Logitech controller, and even that only works after connecting to the Husky remotely and rebooting it.

3.2 Problems

If the ROS services are turned on, they have to be restarted from a Linux bash („sudo service mbs_Husky restart“). It is necessary to ssh (secure shell) into the Husky PC to execute the given command. Otherwise it is also possible to connect a keyboard, mouse and monitor to the Husky and execute the command in a terminal. But this is time consuming and inefficient. Only then it is possible to use Husky properly. Furthermore, the different subnets are not ideally configured regarding to the number of devices. In addition, the 3D LiDAR is only accessible after some routing commands inside the secure shell. Therefore, the Husky is not able to drive itself due to missing data from the configured components. It is also necessary to create a description file for the LiDAR to link it to the robot. This is critical so that the robot's position in relation to the objects can be calculated correctly. Otherwise, safe autonomous driving is not guaranteed.

During further work more problems with the Husky appeared. A significant problem is the lack of connection between the "Zukunftsfabrik Bodensee" network and the ROS system. At the moment the Husky does not know when it is allowed to drive. Furthermore, the robot is not able to navigate itself, based on the lack of further implementation. There can be many problems during the implementation of selfdriving alghorithms. Therefore it is going to be simulated at first to encounter possible problems or risks. In addition to that, the Husky does not know its own position in the real world, caused by a possible malfunction of the Inertial Measurement Unit (IMU) sensor or internal software issues.

3.3 Possible solutions

1. To solve the first problem (restarting ROS services), it is necessary to go through the startup files to find the script that is causing the error. After that, the robot should be able to boot without errors, so that it is possible to just press the start button and use it properly.

2. Furthermore, the networking should be solved by correctly configuring the IP addresses of both Ethernet ports as well as the addresses of both LiDAR sensors. Both ports need static ipv4 addresses on the same subnet as the robot. The LiDAR sensors also require static ip addresses in the same subnet as the ports in order to communicate with them without further commands.

3. After creating a description file, the physical distance between the center of the robot and the center of the 3D LiDAR must be calculated correctly. These calculated coordinates are added to the Husky transformation tree to link the base coordinate system to the LiDAR coordinate system.

4. In order to get a driving allowed signal from the „Zukunftsfabrik Bodensee“ to the Husky there are two possible solutions:

4.1 It is possible to observe the trays with a simple USB-Webcam and run an algorithm to count the objects placed at the Husky. To simplify the detection high contrast objects were placed on the trays (cf. Figure 3.1).

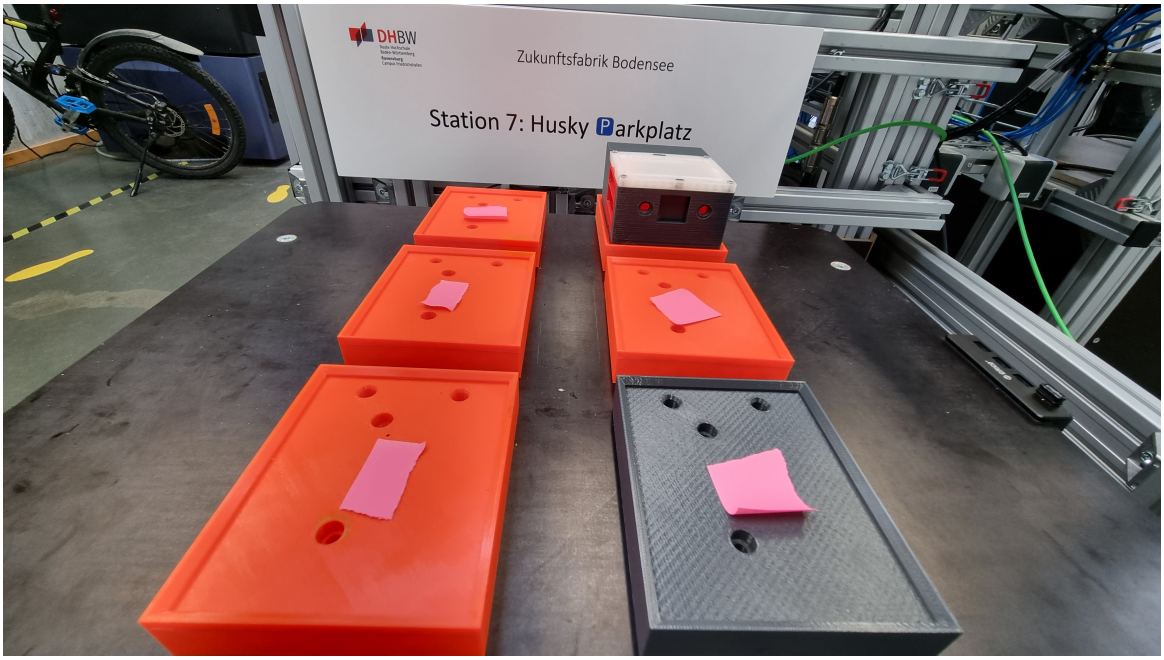


Figure 3.1: Example image of USB-Webcam

The algorithm then counts the number of high-contrast objects, which are not detectable when a component is placed on the trays.

4.2 Furthermore, a network connection between the Husky and the „Zukunftsfabrik Bodensee“ could be established in order to gather data from the PLC via OPC UA.

5. To encounter the risk of damaging the Husky or its sourroundings, a simulation will be implemented with Gazebo. Within that simulation it is possible to implement a simple waypoint navigation to gain experience regarding autonomous navigation and behavior of the Husky during selfdriving.

6. To solve the malfunction of the IMU, it needs to be tested accordingly to elimi-

nate that error. If the IMU was not the problem, the software must be examined closely or updated.

4 Implementation

4.1 Networking solution

To solve problem number two static IP addresses needs to be set. The changes can be made either graphically through the user interface or by editing the robot's netplan file. Editing the netplan file ensures that these network settings are applied at startup. It is therefore necessary to create a netplan file to manage the desired connections. To edit the netplan file, it must be opened with the following command:

```
sudo nano /etc/netplan/netconfig_husky.yaml
```

This command creates (if not existing) or opens the netplan file in a text editor. Now it is important to know the syntax of yaml in order to write a new netplan command. To simplify the editing, a code editor like „Visual Studio Code“ is more suitable. The command to open the file would then be:

```
code /etc/netplan/netconfig_husky.yaml
```

According to (ubuntuusers, 2024) it is necessary to give the netplan program the correct header as shown in Source code 4.1.

Source code 4.1: Netplan configuration file in YAML

```
1 network:
2   version: 2
3   ethernets:
4     eno1: # Velodyne 3D LiDAR
5       dhcp4: no
6       addresses:
7         - 192.168.131.30/24  # Static IP address for eno1 on your PC
8     en1sp: # Hokuyo 2D LiDAR
9       dhcp4: no
10      addresses:
11        - 192.168.131.20/24  # Static IP address for en1sp on your PC
12  routes:
13    - to: 192.168.131.32/24  # Velodyne IP address
14      via: 192.168.131.30
15    - to: 192.168.131.21/24  # Hokuyo IP address
16      via: 192.168.131.20
```

After the header (Source code 4.1, line 3) it is possible to define a custom netplan. The Husky has a default IP address of "192.168.131.1", which leads to the subnet of „131“ as said in section 2.10. Therefore, it was decided that all connected devices would be configured on the same subnet for simplicity. The 3D LiDAR was configured with an IP address of „192.168.131.31“ as described by the previous students in their paper (cf. Schmid and Straßburger, 2024). While the 2D LiDAR already had the IP address „192.168.131.21“, which is in the right subnet. Whereas the address of the 3D LiDAR had to be changed on the web interface to „192.168.131.32“. In order to get data from the sensors, the Ethernet ports of the PC must also be in the same subnet. So in Source code 4.1 the lines from 4 to 16 have to be added.

At first, the file sets static IP addresses for both ports (eno1 and en1sp), so the automatic IP address assignment (DHCP) must be turned off (line 5). Then it is possible to assign a static IP address to the port by adding the lines 6f. To establish the connection, it is necessary to add the routing commands from the devices to the ports, as shown in Source code 4.1. from the line 12 to the line 16.

After saving the file it is crucial to apply the netplan with the following command:

```
sudo netplan apply
```

Otherwise the netplan configuration is not applied correctly.

4.2 Coordinate transformation solution

To solve problem number three the implementation of the 3D LiDAR has to be implemented right. Therefore a new launch file named „custom_launch_velodyne“ was created in the same directory as the package „velodyne_pointcloud“. This file contains the calculated transformation of section 2.15 as well as a link to start the velodyne driver.

```
<launch>
    <node pkg="tf" type="static_transform_publisher"
    name="Velodyne_LIDAR"
    args="0.24835 0 0.9107 0 0 0 base_link Velodyne_LIDAR 90"/>

    <include file="$(find velodyne_pointcloud)/
    launch/VLP16_points.launch"/>
</launch>
```

It is necessary to start the Velodyne driver to create a point cloud. Therefore it was implemented in the „custom_launch_velodyne“ file.

Now it is possible to start the 3D LiDAR with the custom coordinate transformation as well as the velodyne point cloud driver with following command:

```
roslaunch velodyne_pointcloud custom_launch_velodyne.launch
```

Only then the detected points of the LiDAR are visible in Rviz (cf. Figure 4.1).

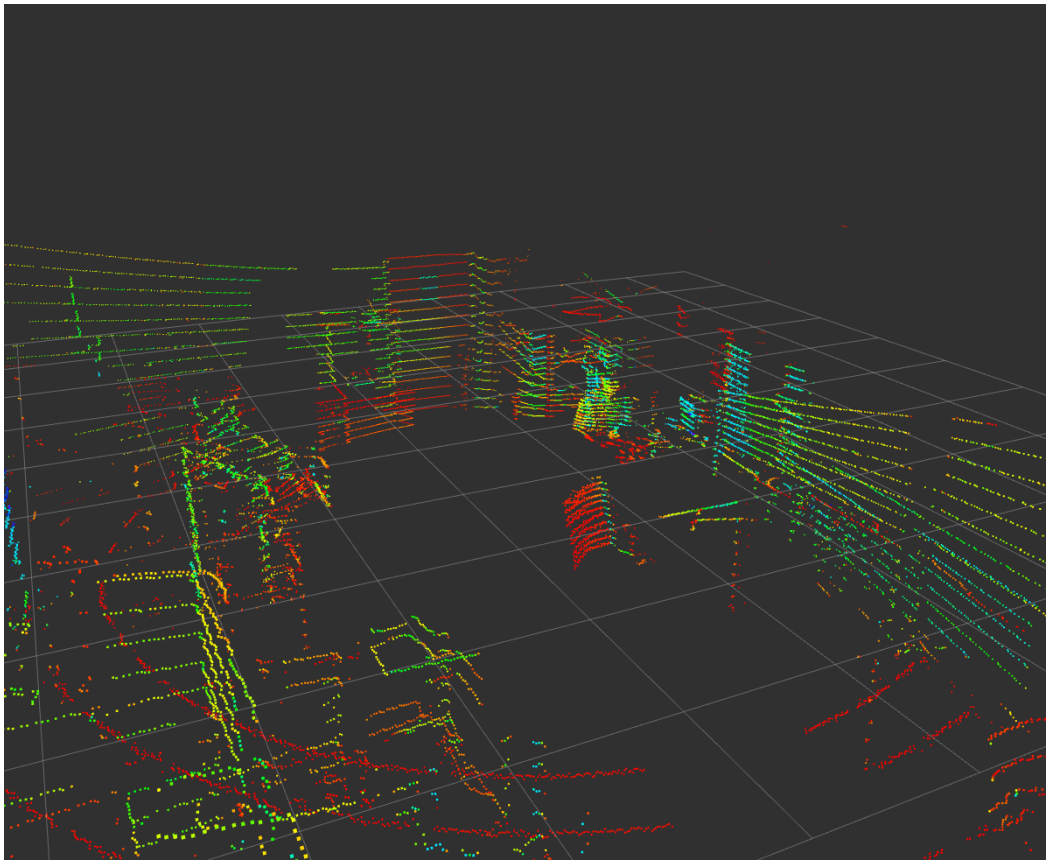


Figure 4.1: Point cloud of 3D LiDAR in Rviz

4.3 Communication Husky - Zukunftsfabrik Bodensee

In order to know when the Husky is allowed to drive (problem number four), a USB-webcam can be used to detect the pieces loaded on the Husky. The sequence could look like this (cf. Figure 4.2):

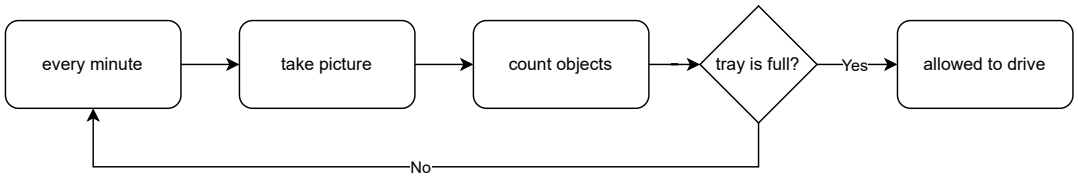


Figure 4.2: Flow chart computer vision

The code behind „take picture“ and „count objects“ could look like this (cf. Source code 4.2):

Source code 4.2: Simple color detection with OpenCV

```
1
2  import cv2
3  import numpy as np
4
5  # Load the image
6  image = cv2.imread('~/Desktop/testImage.jpg')
7
8  # Convert the image to HSV color space
9  hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
10
11 # Define the range for pink/purple hues in HSV
12 lower_pink = np.array([160, 80, 80])  # Adjusted for stronger neon
       intensity
13
14 upper_pink = np.array([175, 255, 255]) # Higher saturation and
       brightness
15
16 # Create a mask for pink/purple objects
```

```
mask = cv2.inRange(hsv, lower_pink, upper_pink)

# Find contours of the detected objects
contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

# Initialize counter
count = 0

# Draw bounding boxes around detected objects and count them
for contour in contours:
    if cv2.contourArea(contour) > 100:  # Filter out small objects
        count += 1
        x, y, w, h = cv2.boundingRect(contour)
        cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 4)

# Annotate the image with the counted number
cv2.putText(image, f'Count: {count}', (10, 30), cv2.
    FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)


# Save the resulting image
cv2.imwrite('~/Desktop/resultOfTestImage.jpg', image)

print(f'Number of pink/purple objects detected: {count}')
```

After running the code, the result can look like this (cf. Figure 4.3):

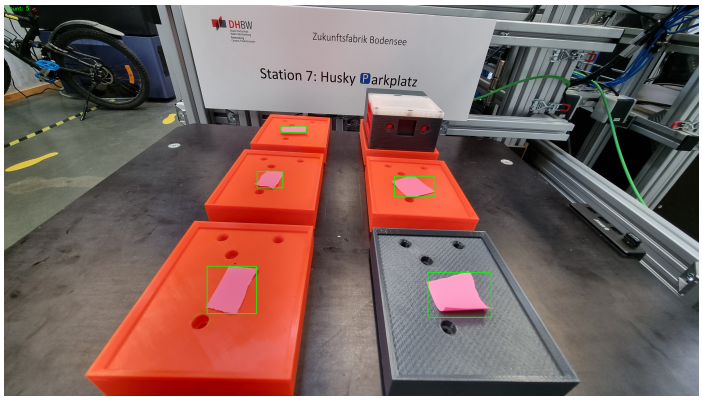


Figure 4.3: Result of high-contrast object detection

With this algorithm it is possible to make a decision whether the Husky is allowed to drive or not. This can be forwarded to the ROS master who will initiate the self driving logic.

Furthermore if a network connection between the Husky and the „Zukunftsfabrik Bodensee“ exists, it is possible to gather data directly from the PLC with the help of OPC UA (cf section 2.13). Therefore, a client must be written in order to connect the Husky PC to the OPC UA server of the PLC. The following code represents a simple client that connects to the OPC UA server and observes a certain variable (cf. „allowed_to_drive“ from Source code 4.3):

Source code 4.3: Simple OPC UA client with python

```
import logging
from opcua import Client, ua
import time

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s [%(levelname)s] %(message)s",
    handlers=[
        logging.FileHandler("opcua_client.log"),
        logging.StreamHandler()
    ]
)

# OPC UA server and node configuration
server_url = "opc.tcp://192.168.0.1:4840"  # Replace with your PLC's address
node_id = "ns=3;s=::Program:Main.allowed_to_drive"  # Replace with actual NodeId

class SubHandler:
    def __init__(self):
        self.last_value = None

    def datachange_notification(self, node, val):
        logging.info(f"Data change on {node}: {val}")
        if isinstance(val, bool):
```

```
26              if self.last_value is False and val is True:
27                  logging.info("allowed_to_drive changed from FALSE to
                        TRUE!")
28              self.last_value = val
29
30  def main():
31      client = Client(server_url)
32      try:
33          logging.info("Connecting to OPC UA server...")
34          client.connect()
35          logging.info("Connected to OPC UA server")
36
37          node = client.get_node(node_id)
38          logging.info(f"Monitoring node: {node}")
39
40          handler = SubHandler()
41          subscription = client.create_subscription(1000, handler)
42          handle = subscription.subscribe_data_change(node)
43
44          logging.info("Monitoring started. Press Ctrl+C to stop.")
45          while True:
46              time.sleep(1)
47
48      except Exception as e:
49          logging.error(f"Error: {e}")
50      finally:
51          client.disconnect()
52          logging.info("Disconnected from server")
53
54  if __name__ == "__main__":
55      main()
```

Within Source code 4.3 the script establishes a connection between the Husky PC as a client and the PLC as a server. Furthermore, it creates a subscription that monitors the required variable. If the OPC UA server notices a change of the variable's value, it notifies the client immediately. This process can be used to get the information if the Husky is allowed to drive or not. In addition to that, every other variable from the PLC could be read by the Husky via OPC UA which could be useful in the future.

4.4 Simulating Husky in Gazebo

To ensure a safe development process of the autonomous navigation of the Husky is implemented with Gazebo (solution to problem number five). Therefore a Husky repository is cloned into a local catkin workspace so that it will not affect the original ROS workspace. After that it is necessary to call „catkin_make“ to rebuild the cloned packages so that ROS notices them as well as self-made packages. Furthermore to start a simulation, it is crucial to source the new catkin workspace with the following command:

```
source ~/catkin_ws/devel/setup.bash
```

After that the simulation can be started with the following command:

```
roslaunch husky_gazebo empty_world.launch
```

This command starts the Gazebo simulation with a Husky robot in an empty world. The simulation can be controlled with the keyboard.

4.4.1 Setup of the simulation environment

To set up the simulation environment, the following command has to be executed within the sourced catkin workspace:

```
roslaunch husky_gazebo husky_playpen.launch
```

This command starts the Gazebo simulation with a Husky robot in a playpen. The playpen is a simple environment with walls and obstacles to test the navigation of the Husky. This can be seen in Figure 4.4

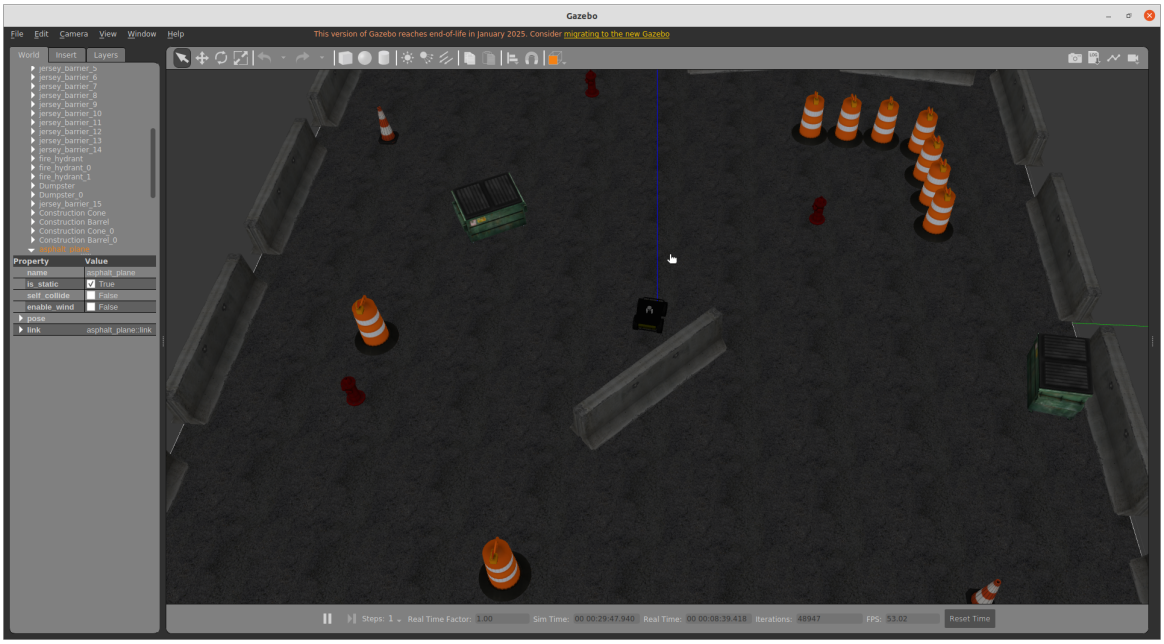


Figure 4.4: Gazebo simulation setup

Then the 3D LiDAR information has to be converted into 2D in order to get the navigation stack working. This is done to simplify the navigation and reduce the use of resources for the simulation.

```
rosrun pointcloud_to_laserscan
pointcloud_to_laserscan_node cloud_in:=/points scan:=/scan
```

After that it is necessary to start the self- locating navigation stack (gmapping) with the following command:

```
roslaunch husky_navigation
mapping_only_ptz.launch scan_topic:=/scan
```

This is a custom launch file that launches the gmapping algorithm and passes the laser scanner argument from the command before. In addition to that, it is necessary to launch „move_base“ to plan a path based on gmapping and given waypoints.

```
roslaunch husky_navigation move_base.launch
```

Only then it is possible to execute a self-written python script that offers waypoints to the „move_base“ package. In Source code 4.4 is an example script that connects to the package and provides waypoints. The script also verifies if the Husky reached its goal otherwise it will continue to drive towards this point.

Source code 4.4: Simple waypoint navigation example

```
#!/usr/bin/env python3

import rospy
import actionlib
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
from geometry_msgs.msg import Pose, Point, Quaternion

def create_goal(x, y, yaw=0.0):
    goal = MoveBaseGoal()
    goal.target_pose.header.frame_id = "map"
    goal.target_pose.header.stamp = rospy.Time.now()
    goal.target_pose.pose = Pose(
        position=Point(x=x, y=y, z=0.0),
        orientation=Quaternion(x=0.0, y=0.0, z=0.0, w=1.0) # for yaw
            quaternions must be calculated correctly
    )
    return goal

def send_goals(client, waypoints):
    for i, pose in enumerate(waypoints):
        x, y = pose
        goal = create_goal(x, y)
        rospy.loginfo(f"Sending goal {i+1}: {pose}")
        client.send_goal(goal)
        client.wait_for_result()
        rospy.loginfo(f"Reached goal {i+1}")

if __name__ == "__main__":
    rospy.init_node("husky_navigator")

    # MoveBase client
```

```
31      client = actionlib.SimpleActionClient("move_base",
            MoveBaseAction)
32      rospy.loginfo("Waiting for move_base action server...")
33      client.wait_for_server()
34
35      # Define waypoints (x, y)
36      waypoints = [
37          (1.0, 1.0),
38          (2.0, 1.5),
39          (3.0, 0.5),
40          (2.0, -1.0),
41          (0.5, -0.5),
42      ]
43
44      send_goals(client, waypoints)
45      rospy.loginfo("Navigation completed.")
```

In order to visualize topics, Rviz is needed. Rviz can be started, including the robot model, with the follwing command:

```
roslaunch husky_viz view_model.launch
```

Within Rviz it is possible to visualize data by adding the desired topics, as seen in Figure 4.5

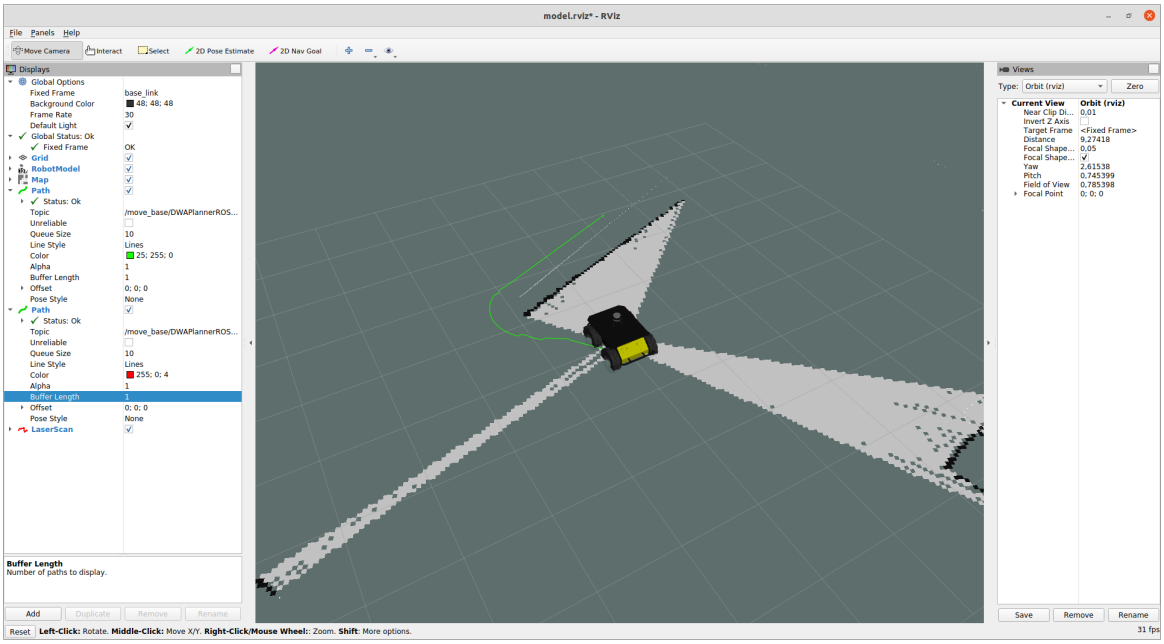


Figure 4.5: Rviz simulation setup

4.5 Real world problem

Due to the lack of time, there is no solution to the possible malfunction of the IMU or softwarebugs. After the simulation was done in Gazebo it is possible to transfer that to the real-world Husky with no effort. But right now the Husky has the problem that its base link, as well as every other link that is linked to the base link is locating itself multiple times at the same time within the map (cf. Figure 4.6).

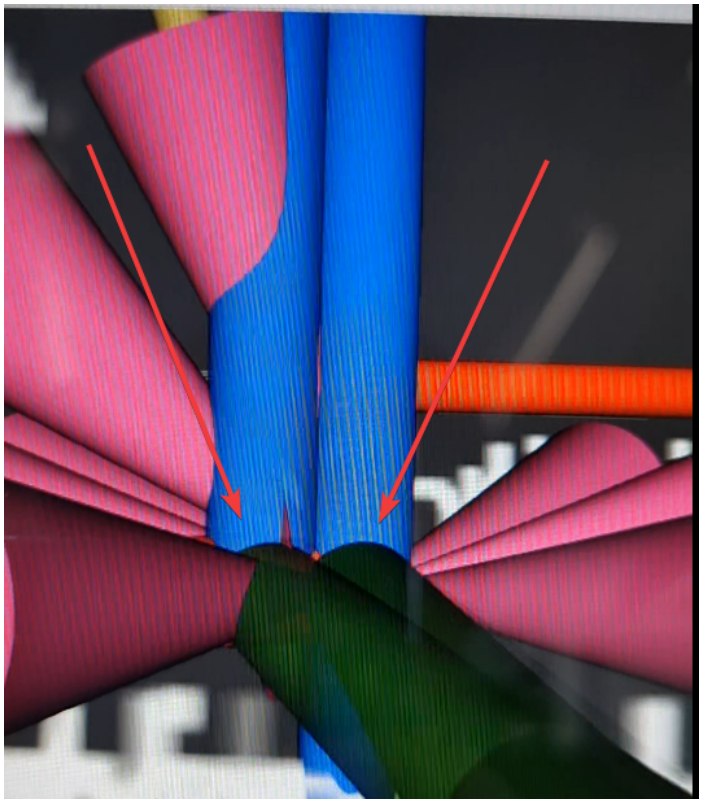


Figure 4.6: Multiple base links

As a result of that, the Husky does not know where it is because its position changes between two points multiple times per second. Therefore it was not possible to transfer the navigation process from the simulation to the Husky.

5 Summary

5.1 Problems during Implementation

Unfortunately, not all problems could be solved. The network problem took the most time, which meant that we could not work on the other sites. Most of the time, only one of the two LiDARs was able to send data, even though they were on the same subnet. The problems were the lack of knowledge about networks on the one hand and the insecurities with Linux on the other hand. However, the infrastructure of the Husky was adapted to ensure better utilization. Furthermore, the 3D LiDAR was linked to the base point of the robot using coordinate transformation to ensure efficient and easy calculation of the detected points. As a result, the robot is perfectly set up for the simulation part.

Using a simple camera would work to get a „driving allowed“ signal, but it comes with different problems. The first problem is that the construction of the Husky must be adapted in order to get a clean vision to all the trays. Furthermore, the camera only has one purpose (observe the trays), which is not sustainable due to the limiting resources of the PC.

An OPC UA connection would be the better solution. It is simple to set up and in the future, if more data from the PLC is needed, it can be added seamlessly. But it is crucial to have a stable network connection between the Husky PC and the „Zukunftsfabrik Bodensee“ network. At first it seemed like an easy task but wether the industrial PC (which is connected to the PLC) as a wifi card nor has the predictive maintanace PC. Therefore it was difficult to get a stable connection running.

During the implementation of self-driving on the real Husky, the problem occurred that it has two different positions. This problem has not yet been solved.

5.2 Future actions

In the next phase, the Husky's abilities will be transferred to a simulation. Autonomous driving can then be efficiently tested in this simulation. This will also make it easier to adjust parameters and optimize the navigation algorithm. The goal is to find out how the robot can drive autonomously from the PTZ to the RITZ. If the simulations are successful, the simulation will be tested in the real world.

Furthermore the communication between the Husky and the PTZ, as well as the RITZ, must be implemented. The Husky must know when it is allowed to start driving from PTZ to RITZ or vice versa. Hereby the PLC from the PTZ should be the superior system that issues the release.

The next step after the second phase is to get familiar with the simulation and try to solve the selflocating error on the real Husky. Then the navigation process can be easily transferred to the Husky. Furthermore it is strongly recommended to upgrade the ROS version noetic to ROS2. Eventually the selflocating problem will be gone by then. Furthermore, ROS2 offers more possibilities due to its better compatibility and community support.

Bibliography

Aulinas, J. et al. (Oct. 2008). "The SLAM Problem: A Survey". In: *Artificial Intelligence Research and Development, Proceedings of the International Conference of the Catalan Association for Artificial Intelligence (CCIA 2008)*. Sant Martí d'Empúries, Spain.

Beuth, Thorsten, Christoph Parl, and Heinrich Gotzig (2024). "LiDAR". In: *Handbuch Assistiertes und Automatisiertes Fahren: Grundlagen, Komponenten und Systeme für assistiertes und automatisiertes Fahren*. Ed. by Hermann Winner et al. Wiesbaden: Springer Fachmedien Wiesbaden, pp. 357–377. ISBN: 978-3-658-38486-9. DOI: 10.1007/978-3-658-38486-9_16. URL: https://doi.org/10.1007/978-3-658-38486-9_16.

Blubaugh, David Allen et al. (2022). "Noetic ROS Further Examined and Explained". In: *Intelligent Autonomous Drones with Cognitive Deep Learning: Build AI-Enabled Land Drones with the Raspberry Pi 4*. Berkeley, CA: Apress, pp. 417–436. ISBN: 978-1-4842-6803-2. DOI: 10.1007/978-1-4842-6803-2_12. URL: https://doi.org/10.1007/978-1-4842-6803-2_12.

Chen, Y. (Feb. 2013). *Algorithms for Simultaneous Localization and Mapping*. Published on February 3, 2013. URL: https://cseweb.ucsd.edu/~yuc007/documents/re_report.pdf (visited on 07/03/2025).

Corke, Peter (2023). *Robotics, Vision and Control: Fundamental Algorithms in Python*. 3rd ed. Springer Tracts in Advanced Robotics. Springer. DOI: 10.1007/978-3-031-06469-2. URL: https://link.springer.com/book/10.1007/978-3-031-06469-2.

"Digitalisierung der Baustelle: Clever bauen mit mobilem Roboter" (June 2, 2020). In: *Forschung Kompakt*. URL: https://copilot.cloud.microsoft/?fromcode=bingchat&redirectid=F885CAA0A08D45FDB8D8AB2F96D9DB64&auth=2 (visited on 01/07/2025).

Dillmann, Rüdiger and Martin Huck (1991). *Informationsverarbeitung in der Robotik*. 1st ed. Springer-Lehrbuch. Springer-Verlag Berlin Heidelberg, pp. XI, 337. ISBN: 978-3-540-53036-7. DOI: https://doi.org/10.1007/978-3-642-50188-3.

Durrant-Whyte, H. F. and T. Bailey (2006). "Simultaneous Localisation and Mapping (SLAM) Part 1: The Essential Algorithms". In: *IEEE Robotics and Automation Magazine* 13.2, pp. 99–110.

Ein Leuchtturm mit überregionaler Strahlkraft (2024). Online. URL: https://www.ir-bauen.com/de/case-studies/ritz-friedrichshafen (visited on 11/29/2024).

elektronik, reichelt (2024). *UTM-30LX-EW HOKUYO UTM-30LX-EW LiDAR*. Online. URL: https://www.reichelt.de/de/de/hokuyo-utm-30lx-ew-LiDAR-utm-30lx-ew-p320261.html?PROVID=2788&gad_source=1&gbraid=0AAAADwnxtav2lqcyuIaaolot_Cm86R5n&gclid=CjwKCAiAxqC6BhBcEiwAlXp45xyyTOlAYlS1txuoMGBA1pw1q5e3l9IuYkuy7umi_wm3BfzzUVRPwRoCmeQQAvD_BwE&&r=1 (visited on 11/29/2024).

Eliazar, Ariel J. and Richard Parr (n.d.). "DP-SLAM: Fast, Robust Simultaneous Localization and Mapping Without Predetermined Landmarks". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* ().

Garzon, M. et al. (May 2017). *Using ROS in Multi-robot Systems: Experiences and Lessons Learned from Real-World Field Tests*. Springer.

Han, Shaowen (Feb. 2021). *Steuerung gängiger Industrieroboter mit ROS*. Dresden. URL: https://tudresden.de/bu/bauingenieurwesen/cib/ressourcen/dateien/publikationen/Projekt-_Diplomarbeiten/PA-HSW-Final.pdf?lang=de.

Hexagon (2024). *Velodyne Puck*. Online. URL: https://autonomoustuff.com/products/velodyne-puck-vlp-16 (visited on 11/29/2024).

Koenig, Nathan and Andrew Howard (2004). *Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator*. Los Angeles, USA. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=79f91c1c95271a075b91e9fdca43d6c31e4cbe17.

Kriegel, K.I. et al. (2025). *Simulation eines mobilen Manipulators mit Gazebo*. Published on 12.05.25. URL: https://www.cea-wismar.de/asim2021/tagungsband/data/ASIM_WS_2021_paper_11.pdf.

Linux (Nov. 23, 2024). Online. URL: https://en.wikipedia.org/wiki/Linux (visited on 11/24/2024).

Mandl, Peter (Oct. 25, 2018). *Internet Internals. Vermittlungsschicht, Aufbau und Protokolle*. Springer Vieweg Wiesbaden. ISBN: 978-3-658-23535-2. DOI: https://doi.org/10.1007/978-3-658-23536-9.

Montemerlo, M. and S. Thrun (2007). *FastSLAM: A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics*. Springer.

MYBOTSHOP (2025). *Clearpath Husky A200*. Online. URL: https://www.mybotshop.de/Clearpath-Husky-A200 (visited on 01/03/2025).

OpenCV Team (2025). *About OpenCV*. URL: https://opencv.org/about/ (visited on 06/30/2025).

Pavlovic, D. (2016). *Analyse und Vergleich von SLAM-Algorithmen*. Published on 05.09.2016.

Plenk, Valentin (2024). "Protokolle: OPC UA". In: *Angewandte Netzwerktechnik kompakt: Dateiformate, Übertragungsprotokolle und ihre Nutzung in Java-Applikationen*. Wiesbaden: Springer Fachmedien Wiesbaden, pp. 139–156. ISBN: 978-3-658-43530-1. DOI: 10.1007/978-3-658-43530-1_8. URL: https://doi.org/10.1007/978-3-658-43530-1_8.

Rieber, Prof. Dr.-Ing Jochen M. (2024). *Robotik. Vorlesung im Studium Elektrotechnik / Automation*. Script. (Visited on 11/29/2024).

Schatton, Dipl. Ing. Heike (2024). *Automatisierung*. Script. (Visited on 12/22/2024).

Schmid, Daniel and Paul Straßburger (July 15, 2024). "Inbetriebnahme eines Transportroboters für eine neue Produktionslinie". Studienarbeit T3 3100 & T3 3200. Studienarbeit. Duale Hochschule Baden-Württemberg Ravensburg, Campus Friedrichshafen.

SICK (July 2018). *Funktionsweise und varianten von LiDAR-Sensoren*. Online. URL: https://cdn.sick.com/media/docs/5/25/425/whitepaper_lidar_de_im0079425.pdf (visited on 11/29/2024).

Statista (2025). *Computer Vision - Weltweit*. https://de.statista.com/outlook/tmo/kuenstliche-intelligenz/computer-vision/weltweit. Abgerufen von Statista Inc.

Technologietransfer Zentrum GmbH, RITZ Regionales Innovations- und (Jan. 7, 2025). *RITZ Innovationszentrum*. URL: https://ritz-innovationszentrum.com.

Thrun, Sebastian, Wolfram Burgard, and Dieter Fox (2000). *Probabilistic Robotics*. MIT Press.

Tola, Daniella and Peter Corke (2024). "Understanding URDF: A Dataset and Analysis". In: *IEEE Robotics and Automation Letters* 9.5. Accepted author manuscript, peer reviewed version, pp. 4479–4486. DOI: 10.1109/LRA.2024.3381482. URL: https://doi.org/10.1109/LRA.2024.3381482.

Ubuntu (2024). *Powering robotics in every industry*. Online. URL: https://ubuntu.com/robotics (visited on 11/26/2024).

ubuntuusers (Nov. 23, 2024). *Netplan*. Online. URL: https://wiki.ubuntuusers.de/Netplan/ (visited on 12/10/2024).

Wiki, ROS (2025). *tf - ROS Wiki*. URL: http://wiki.ros.org/tf#view_frames (visited on 01/09/2025).

ZPD Zentrum für Digitalisierung in Produktion und Produktentwicklung (Dec. 2023). Imagebroschuere. URL: https://www.ravensburg.dhbw.de/fileadmin/Ravensburg/Dokumente_Bilder_Contentbereich/FIT/DHBW_RV_FN_Imagebroschuere_ZDP.pdf (visited on 11/28/2024).

List of Figures

1.1 Service robots for professional use (cf. Schatton, 2024, p. 187) 1

2.1 Illustrative representation of RITZ (cf. *Ein Leuchtturm mit überregionaler Strahlkraft* 2024; *ZPD Zentrum für Digitalisierung in Produktion und Produktentwicklung* 2023) 4
2.2 Illustrative representation of Zukunftsfabrik Bodensee (cf. *Ein Leuchtturm mit überregionaler Strahlkraft* 2024; *ZPD Zentrum für Digitalisierung in Produktion und Produktentwicklung* 2023) 5
2.3 Illustrative representation of ROS 6
2.4 Illustrative representation of ROS - Gazebo connection (cf. p. 2 Kriegel et al., 2025) 8
2.5 Estimation of robot position and landmarks (cf. p. 2 Durrant-Whyte and Bailey, 2006) 11
2.6 Relationship of the process variables (cf. p. 3 Chen, 2013) 12
2.7 Illustrative representation of LiDAR systems (cf. elektronik, 2024; Hexagon, 2024; SICK, 2018) 18
2.8 Husky PC 19
2.9 Representation of the Husky 21
2.10 Illustrative representation of Husky (cf. Schmid and Straßburger, 2024) 24
2.11 Part of Husky transformation tree 25

3.1 Example image of USB-Webcam 28

4.1 Point cloud of 3D LiDAR in Rviz 33
4.2 Flow chart computer vision 34
4.3 Result of high-contrast object detection 35
4.4 Gazebo simulation setup 39
4.5 Rviz simulation setup 42
4.6 Multiple base links 43

Example code

2.1 Sample excerpt from URDF file 9

4.1 Netplan configuration file in YAML 31
4.2 Simple color detection with OpenCV 34
4.3 Simple OPC UA client with python 36
4.4 Simple waypoint navigation example 40

List of abbreviations

2D	two dimensional
3D	three dimensional
AI	Artificial Intelligence
CV	Computer Vision
DP	Distributed Particle
EKF	Extended Kalman Filter
EM	electromagnetic
etc.	et cetera
IMU	Inertial Measurement Unit
IP	Internet Protocol
LiDAR	Light Detection And Ranging
OPC UA	Open Platform Communications Unified Architecture
OS	Operating System
PTZ	Produktionstechnisches Zentrum
RITZ	Regionales Innovations- und Technologietransfer Zentrum GmbH
ROS	Robot Operating System
SEIF	Sparse Extended Information Filter
SLAM	Simultanious Localization And Mapping
ZPD	Zentrum für Digitalisierung in Produktion und Produktentwicklung

List of Tables

2.1 Dependencies 13
2.2 Advantages and Disadvantages of EKF SLAM 15
2.3 Advantages and Disadvantages of SEIF 16
2.4 Advantages and Disadvantages of Particle Filter 16
2.5 Advantages and Disadvantages of FastSLAM 16
2.6 Advantages and Disadvantages of DP-SLAM 17

1.1 List of artificial intelligence-based tools used 54

Use of Artificial Intelligence-based Tools

This study utilized artificial intelligence (AI) tools. The table 1.1 offers a comprehensive overview of the applied tools and their respective purposes.

Table 1.1: List of AI-based tools used

Tool	Description of use
DeepL	• Help with the translation of individual sections