

Development of an intelligent lidar-based 3D navigation system for the Unitree Go1

M a s t e r w o r k

**at the
Hof University of Applied Sciences Faculty of
Computer Science Computer Science
Degree program: Master Computer Science M.Sc.**

**submitted to:
Prof. Dr. Christian Groth
Alfons-Goppel-Platz 1
95028 Hof**

**submitted by:
Jonas Kemnitzer
Waldstraße 1
95183, Zedtwitz**

Hof, September 13, 2023

Any sufficiently advanced technology is indistinguishable from magic.

- Arthur C. Clarke

Acknowledgments

A number of people have supported me in the preparation of this thesis. It is with great gratitude that I would express my appreciation to these people.

Special thanks go to my supervisor, Prof. Dr. Christian Groth, who made it possible for me to work on this topic and contributed to the completion of this thesis at all times with his knowledge and personal assessment. His guidance during the process helped me to implement my thoughts and ideas in a meaningful way.

I would also like to thank my friends and fellow students Noah Lehmann and Ronja Gesell, who stood by my side during the correction process and supported me with their knowledge.

Finally, I would like to thank my family, who have supported me in every way during my academic career. Their love, patience and belief in me have given me the strength to face and overcome the challenges of this work.

Table of contents

1	Introduction	1
1.1	Background and motivation	1
1.2	Objectives of the work	2
1.3	Scope of application and restrictions.....	2
1.4	Content structure of the work	3
2	Fundamentals and state of research	4
2.1	Basics	4
2.1.1	Robotics.....	4
2.1.2	Components of robots	7
2.1.3	Robot programming	9
2.1.4	Artificial intelligence.....	12
2.1.5	LiDAR technology and artificial intelligence.....	15
2.2	Challenges and possibilities of four-legged robots.....	19
2.3	Related work.....	20
3	Commissioning the Unitree Go1 robot	22
3.1	Hardware components.....	22
3.1.1	Overview	22
3.1.2	External connections	24
3.1.3	Internal consideration	25
3.1.4	Overview RS-Helios-16P LiDAR	29
3.2	Software components	31
3.2.1	Analysis of the UnitreeCameraSDK.....	32
3.2.2	Analysis of the ultrasonic sensors.....	35
3.2.3	Simulation of the Unitree Go1	38
3.2.4	Additional resources for the Go1	40

4	Implementation	42
4.1	Description of the experimental setup	42
4.2	Test execution	44
5	Results analysis	51
5.1	Evaluation of the scenarios	51
5.1.1	Consideration of indoor scenario	52
5.1.2	Consideration of outdoor scenario	54
5.1.3	Key findings	57
5.2	Limitations of the system	57
6	Conclusion	59
6.1	Summary	59
6.2	Future research approaches	61
A	Appendix: Illustrations	63
B	Appendix: Code	71
C	Appendix: Tables	78

List of illustrations

2.1	Representation of a ROS graph	10
2.2	Chronological development of AI from 1930 to 2000 with relevant milestones	12
3.1	Overview of the components of the Go1 robot.....	24
3.2	Connections and ports on the surface of the Go1 robot	25
3.3	Top view of the internal structure of the Go1 robot and assignment of components within it	26
3.4	Top view of the internal structure of the Go1 head and regulation of the components contained therein	27
3.5	Architecture diagram of the Go1	27
3.6	Architectural diagram of the MIT Cheetah.....	28
3.7	Excerpt from the RSView program	31
3.8	Distribution of the UDP packets of the RS-Helios-16P sensor	31
3.9	Excerpt from the RViz program for visualizing the joint angles of the Go1	38
3.10	Excerpt from the Gazebo program for visualizing the Go1	39
4.1	Mounting the LiDAR sensor on the Go1	43
4.2	Diagram of the hdl_graph slam nodes	46
4.3	Nodes and topics of the rqt graph for the hdl_graph slam algorithm .	49
5.1	Photo with corresponding determined 3D point cloud environment of the foyer and SLAM graphs of the IISYS	52
5.2	Different views of the mapped environment of the indoor scenario (FAST GICP).....	53
5.3	Incorrect environment map generated by HDT OMP.....	54

5.4	Different views of the mapped environment of the outdoor scenario (FAST GICP)	55
5.5	Comparison of the 3D map for inclines and stairs without floor detection (left) and with (right)	56
5.6	Representation of the faulty 3D map due to inclined reference measurements.....	56
A.1	Output of the installed ROS packages on the NVIDIA Xavier NX	63
A.2	Output of the installed ROS packages on the Raspberry Pi	64
A.3	Excerpts from the web application and display of the person recognition software	65
A.4	Excerpts from the web application and display of the various Cameras	66
A.5	Excerpts from the iOS app.....	66
A.6	Error message output when attempting to save the sensor data. 67	
A.7	Output of the error message when using the CameraSDK	67
A.8	Screenshot from the configuration settings of the LiDAR	68
A.9	Error message output when converting LiDAR formats .	68
A.10	Illustration of the Unimate robot arm	69
A.11	Illustration of the spot robot	70

List of tables

2.1	Table on the classification of robot types according to various characteristics according to [Mai16]	6
3.1	Overview of the range of motion of the joints	23
3.2	Main components and their tasks in Go1	26
3.3	Determined IP address of the robot hardware	29
3.4	Table on the technical specifications of the RS-Helios-16P sensor	30
4.1	Characteristics of the two measurement scenarios.....	43
4.2	Description of the recording options for the measuring points	44
4.3	Characteristics of the generated bag files	49
C.1	Characteristics of the Raspberry Pi on board the Go1	78
C.2	Characteristics of the NVIDIA Xavier NX on board the Go1	78
C.3	Characteristics of the NVIDIA Jetson Nanos on board the Go1.....	79

List of abbreviations

AI	Artificial intelligence
ML	Machine Learning
DL	Deep Learning
ANN	Artificial Neural Networks
IMU	Inertial Measurement Unit
CPU	Central Processing Unit
ROS	Robot Operating System
DARPA	Defense Advanced Research Projects Agency
DDS	Data Distribution Service
SPOF	Single Point of Failure
LiDAR	Light Detection and Ranging
SLAM	Simultaneous Localization and Mapping
RRT	Rapidly-exploring Random Tree
DOF	Degrees of Freedom
PMSM	Permanent magnet synchronous motor
FOC	Field Oriented Control
AVE	Absolute Value Encoder
MCU	Main Control Unit
RPM	Rounds Per Minute
OS	Operating System
BMS	Battery Management System
SDK	Software Development Kit
WWAN	Wireless Wide Area Network
PCD	Point Cloud Data
UDP	User Datagram Protocol
MSOP	Main Data Stream Output Protocol
DIFOP	Device Information Output Protocol
I/O	Input and Output
GUI	Graphical User Interface

LCM.....	Lightweight Communications and Marshalling
URDF	Unified Robot Description Format
NDT	Normal Distribution Transform
VGICP	Voxelized Generalized Iterative Closest Point Algorithm
ICP	Iterative Closest Point
SVD.....	Singular Value Decomposition
RANSAC	Random Sample Consensus
XML.....	Extensible Markup Language
NaN.....	Not a Number
APE	Absolute Pose Error
RPE.....	Relative Pose Error

1 Introduction

This chapter defines the background and objectives of the thesis. This includes the scope of application, which defines the framework conditions, as well as further limitations regarding the scope of the thesis. An outline of the content structure rounds off the introductory chapter.

1.1 Background and motivation

The use as well as the implementation of human-robot systems are becoming increasingly relevant in today's society, be it through their use in private households with the help of vacuum robots or high-risk environments such as industrial plants or disaster areas to assist in the rescue of people. Light Detection and Ranging (LiDAR) sensors can help by creating a 3D map of the environment using so-called point clouds. This helps

help robots gain a better understanding of their environment. Above all

Especially when considering safety-relevant aspects, reliable detection of objects and people is essential to prevent damage. In contrast to conventional sensors, LiDAR sensors have an advantage against a wide range of

external influences such as lighting conditions, which - in addition to their high level of detail and speed¹ - **extremely** suitable as part of the development of navigation systems for robots.

The emergence of autonomous vehicles and smart city applications will further increase the importance of such developments. In dense urban environments, for example, the use of lidar-based robotic systems can enable increased safety for pedestrians and more efficient traffic and urban planning. This plays

¹This means that it is possible to scan large areas in a relatively short time and obtain a large amount of data.

LiDAR sensors are also able to record spatial 3D information without detailed images of people.

1.2 Objective of the work

The focus of this thesis is to investigate the feasibility of a navigation system using four-legged robots. In particular, a Unitree Go1 Edu quadruped² robot is to be integrated into the operating system of the Hof University of Applied Sciences. This is done with the help of the LiDAR sensor mounted on the back and common artificial intelligence (AI) methods for localizing and recognizing the environment in the immediate vicinity of the robot. The focus is on the development of a functional demonstrator with which several tests can then be carried out under real conditions.

1.3 Scope of application and restrictions

As mentioned above, there is already a limitation in the choice of location. The university building and surrounding grounds were chosen as the environment for the test. It offers complex scenarios such as stairs and narrow corridors as well as larger areas to test the robot's navigation and localization software.

The Hof University of Applied Sciences is providing two Unitree Go1 robots as hardware. These provide a range of functions, libraries and sensors to develop the prototype. For more information on the structure of the robot, see chapter 3.

²Quadruped is a word for the form of locomotion in which four limbs are used to carry weight and move.

1.4 Content structure of the work

The thesis is divided into six chapters. The introduction begins by describing and defining the background, objectives and scope of this work.

This is followed by a chapter on the necessary fundamentals of robotics and AI in order to introduce and explain terms relating to related technologies. After providing an overview of the underlying technologies, the challenges and possibilities of the project to use a four-legged robot on the university campus are **presented**. This section is rounded off by a reference to existing work that addresses problems relevant to the project. Chapter 3 deals with the commissioning of the robot. An overview of the hardware and software components of the Unitree Go1 is given here. In addition to this, the main focus of the following chapter is on the actual implementation. To this end, the individual steps for implementing the navigation and localization algorithms are **listed**.

Finally, the last two chapters provide an analysis of the results and a conclusion in which the previously defined tests are evaluated. In the course of this, the limitations of the overall system are discussed. A brief outlook on future research possibilities and a summary of the research results form the conclusion of the work.

2 Fundamentals and state of research

The aim of this chapter is to provide basic knowledge about robotics, such as programming, as well as approaches in the field of AI. Building on this, an introduction to the topic of Simultaneous Localization and Mapping (SLAM) is given. Aspects that describe the possibilities and challenges of the project are also highlighted. Finally, relevant work in the field of the use of four-legged robots is discussed.

2.1 Basics

The field of robotics can be regarded as an interdisciplinary science due to the combination of several sub-areas. For example, the field includes other areas of knowledge such as control engineering, computer science, mechanical engineering, mathematics, electrical engineering, sensor technology and sub-areas of AI [Spa19].

2.1.1 Robotics

The first modern robots were created at the beginning of the 20th century [Fau22]. The first modern programmable robot³ was patented in 1954 by the American inventor George Devol. He and Joseph Engelberger brought the Unimate (see Appendix A.10) onto the market in the 1960s. This was the first hydraulically operated robotic arm that could execute commands stored on a magnetic drum. Just one year later, this invention was used for manual welding work in an assembly line at General Motors and paved the way for fully automated welding machines.

³This was actually a robotic arm/manipulator

production lines, which are familiar from many industries today [ENG23].

However, the first mobile robots that could move in a specific environment, albeit remotely controlled, were not developed until a few years later - in 1968. Between 1966 and 1972, the Stanford Research Institute in the United States developed the Shakey robot, which executed a command after simple programming and is thus considered the world's first mobile autonomous robot [Fra19].

The development of four-legged robots took its greatest leap forward with the TITAN series from the Japanese developer Shigeo Hirose [Sha21]. These four-legged robots were the first in the world to be equipped with foot-mounted pressure sensors and a position sensor.⁴ were equipped. Later, in 1986, TITAN IV was developed with additional functions, such as converting crawling into trotting by gradually increasing speed. TITAN V and TITAN VI were developed for purposes such as weight mechanics, dynamic walking and other features. The series continues up to the latest version, the TITAN-XIII [Kit+16].

Not to be neglected is the contribution of the Massachusetts Institute of Technology and Boston Dynamics in the development of four-legged robots. The Little Dog [Sta10], a small four-legged robot launched in 2010, had four legs, each powered by three electric motors. This robot was developed for the Defense Advanced Research Projects Agency (DARPA). The robot was able to climb and move dynamically. In 2013, the Massachusetts Institute of Technology developed a highly efficient four-legged robot called MIT Cheetah [Ble+18]. This 70-pound robot consumed very little energy as it trotted continuously at a speed of 5 km/h for up to an hour and a half.

A nowadays well-known four-legged robot is also developed by Boston Dynamics. developed. The series bears the name Spot (see Figure A.11). It was presented on 23 March 2016. In March 2021, the Spot was used by SpaceX to inspect potentially dangerous areas around the crash site of rockets [Dyn23] [Ind21].

The future of four-legged walking robots is promising. In contrast to their counterparts on wheels, caterpillars or in the air, robots with legs can move excellently in rough terrain and unstructured environments. Due to their ability to climb and crawl, they are ideally for certain types of applications.

⁴an instrument for determining the angular deviations of the axes

The new systems can be used for applications and areas of use (e.g. for accessibility requirements) in which robots on wheels would not work. They can also be equipped with a variety of sensors, e.g. cameras, alarm cameras, LiDAR or gas sensors to measure air pollution. These sensors facilitate autonomous navigation and enable intelligent localization and object recognition. They also open up a wide range of possible applications, such as the inspection of machinery or the rescue of people from crisis areas.

Robot types

There are different characteristics and ways of categorizing robots. Nevertheless, a precise classification of robots is often ~~impossible~~ Due to the arbitrary combination of properties or drive types, a wide variety of combinations are possible. The best-known type of robots are those used in industrial automation. However, there are a number of other types of robots. Table 2.1 shows a subdivision considering various characteristics.

according to design	according to use	Others
Stationary robots		
Industrial robots		
Articulated arm robots	Service robot	BEAM (biology, electronics, aesthetics and mechanics)
SCARA robot	household helper	
Delta robot		
autonomous mobile robots	Exploration robot	
Working robot	Toy robot	Swarm robots
Walking robot	Transport robots	Nanorobots
rolling robots	Military robots	bionic robots
Flying robots	Rescue robot	
cognitive robots	medical robots	
humanoid robots (androids)		

Table 2.1: Table on the classification of robot types according to various characteristics according to [Mai16]

2.1.2 Components of robots

Modern robot systems can vary in terms of their equipment and purpose. Basically, however, five core components can be identified, which will be considered below [Par20].

Actuators

An actuator is the part of a robot that helps to carry out physical movements by converting energy into mechanical force⁵. Simply put, it is the component that enables movement. The best-known types of actuators include electric, hydraulic and pneumatic actuators [Spa19].

Electric actuators use an external energy source such as a battery to generate motion energy. Hydraulic actuators use a non-compressible fluid to generate motion using a piston or rotary valve. Finally, pneumatic actuators use compressed air.

The different types have advantages and disadvantages. While hydraulic systems are usually more powerful and precise, the movement is slower. In contrast, pneumatic systems are faster but less precise⁶. Electric drives enable extremely precise control and positioning. Another feature of electric actuators is their low operating costs [Ph.20].

Sensors

Sensors help robots to perceive their environment or to pick up measured values via internal components or external environmental factors. Similar to actuators, there are also a number of different types that differ depending on the task [ROB19]. The sensors mentioned here will be assigned later in Chapter 3 using the Go1 robot.

⁵The further distinction between translatory and rotary actuators is explained in this paper. not considered, as these are of minor importance

⁶For example, the position can also change if the load changes.

Environmental sensors

These include light sensors, sound sensors and temperature sensors, for example. For light sensors mainly photoresistors and photovoltaic cells are used. Sound sensors are used to convert sound waves into electrical signals.

Proximity sensors

Most proximity sensors are used as distance sensors. They are also commonly referred to as distance sensors. Examples of this are infrared or ultrasonic sensors.

Position sensors

Position sensors are used to provide the robot with information über seine Lage in der environment or the position of the installed components, such as joints, to deliver. An Inertial Measurement Unit (IMU), for example, provides a combination of speed, orientation and gravity.

In summary, sensors provide important data for the robot. This data includes, among other things: Position, size, orientation, speed, distance, temperature, weight, force and many other factors that help robots perceive their environment and perform tasks. With the increasing demand for automation and computer-controlled machines, sensors are becoming more and more important.

Control system

These include the central processing unit (CPU) and any program that controls the robot. In the anatomy of robots, CPUs function in a similar way to the human brain. The data is fed in with the help of sensors. In cooperation with the CPU, other hardware and a software component, the information is processed and sent to the control systems as commands. Section 2.1.3 deals with ways of programming robots.

End-effectors

Another component that most robots have in common are end effectors. The term refers to the tools on board the robot that perform the work and interact with the environment or a workpiece. Modern medical robots, for example, use small scalpels and cameras to enable delicate operations, while large industrial robots are usually equipped with welding torches, screwdrivers or paint sprayers.

Power supply

Power supplies can take various forms. Stationary robots, e.g. in factories, are supplied directly with power like most devices. Mobile robots generally have high-performance batteries, while robot probes and satellites are usually equipped with solar cells to generate energy from the sun. After providing an overview of the most common components, the following section deals with the programming of robots.

2.1.3 Robot programming

Robot programming is the creation of instructions that enable a robot to perform a specific task. These instructions are usually written in a programming language that is understood by the robot's control system. The desired behavior of the robot can be defined, broken down into smaller, more manageable tasks and then implemented using code, for example. Nowadays, there are various approaches to programming robots. Frequently mentioned methods are, for example, online, online programming or teach pendant [Mai16].

With regard to the programming of robots, Robot Operating System (ROS) has established itself as the standard. It facilitates the development of more complex scenarios and offers a range of functions. The latest version, ROS2, extends and improves the predecessor in many aspects [JC21] [MF13]. The aim of ROS is to create a standard for programming robots and at the same time to offer standard software components that can be easily integrated into user-defined robot applications.

applications can be integrated [Rob21].

Contrary to its name, ROS is not an operating system. It functions more as middleware or a kind of framework⁷. It provides functionalities and built-in libraries for communication between components. It mainly supports Python (rospy) and C++ (roscpp) as programming languages. Connections for Java (rosjava) or JavaScript (roslnodejs) are also available.

The main players in the ROS ecosystem are so-called nodes⁸. Each node fulfills an inherent closed task. ROS enables communication between these nodes using topics⁹ [Rob16d], services¹⁰ [Rob16c] and messages¹¹ [Rob16b]. Together, these components form a network that can be represented as a graph [Rob16a]. Figure 2.1 shows an example of a calculation graph.

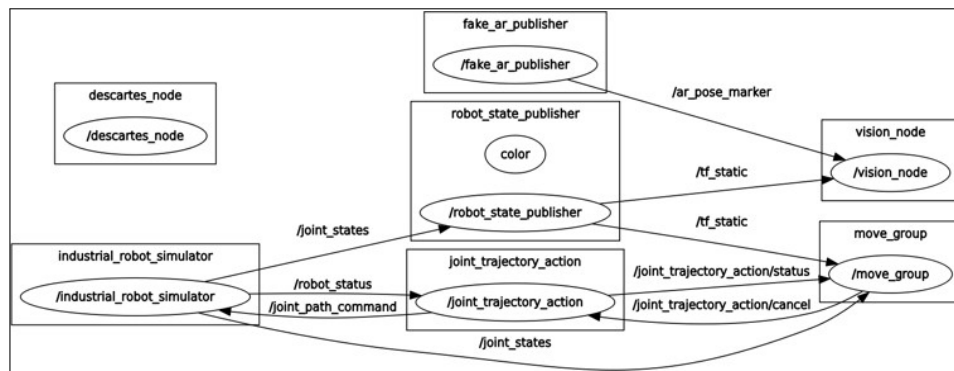


Figure 2.1: Representation of a ROS graph from [Web17]

A robot system usually comprises various nodes. For example, one node controls a range finder, another carries out localization and yet another node carries out path planning. The ROS master enables name registration for the nodes registered in the network. Without the master, the nodes would not be able to find each other, exchange messages or call up services.

A message is a data structure that consists of typed fields. Supported are primitive standard types (integer, floating point, Boolean values etc.) as well as

⁷A framework is a framework for software development and programming that provides basic structures and programming tools

⁸German

Node⁹ German

topics¹⁰

German

services

¹¹German News

Arrays of primitive types. Messages can contain arbitrarily nested structures and arrays.

The messages are exchanged between the nodes based on the publish/subscribe pattern [Tar12] with the help of topics. The topic is a label that is used to identify the content of the message. A node that is interested in a certain type of data subscribes to the corresponding topic.

Finally, services help with data-intensive communication. In contrast to topics, which send information to each other according to the publish-subscribe model, services follow a request-response model.

The advantages of ROS for programming are many and varied. The modular architecture enables good scaling of projects. The large community also provides a wide range of different packages [Rob23b]. In addition to community-based packages, ROS also provides a range of other development tools. These include including the simulation software RViz and Gazebo as well as tools for debugging applications (rqt_gui). In addition, ROS simplifies the use of device drivers and interface packages for sensors and actuators. However, ROS is complex to get started with and can therefore be overwhelming for newcomers.

One reason for this is the amount of knowledge required in areas such as the Linux operating system, programming in Python or C++ and basic engineering and computer science. Currently, with the ROS Noetic distribution, ROS will reach its end-of-life cycle in May 2025. A switch to ROS2 will therefore be unavoidable in the coming years.

ROS2 attempts to avoid many of the problems of ROS and relies on new approaches [Rob23a]. The primary/subordinate middleware is switched to Data Distribution Service (DDS). DDS promises increased efficiency, reliability, lower latency, scalability and improved configurable parameters. ROS2 has a base library written in C called `rcl` on which further libraries are built. This is one of the main reasons why ROS2 is able to support more languages than just Python and C++, e.g. Java and C#. Furthermore, Catkin is replaced by Ament as a build system. The elimination of the master node also eliminates a single point of failure (SPOF).

2.1.4 Artificial intelligence

The term AI is no longer an abstract concept that has long been characterized by science fiction novels and films. Nowadays, the term has little to do with the old ideas. However, it is clear that AI has become an integral part of modern society, for example in the context of product recommendations on Amazon, the automatic recognition of faces in images, autonomous vehicles and the Siri voice assistant.

The central object of AI research is to imitate human cognitive abilities, for example by using an algorithm to independently discover structures and correlations in input information without human intervention.¹² discovered. [CSI23].

Historically, the development of AI can be divided into different phases [Xia22]. Figure 2.2 shows relevant milestones in the development of AI over the last few decades. The pioneer of the first developments in this area is above all Alan Turing. In his essay [Tur09], he dealt with the question of whether machines could think and make decisions as rationally and intelligently as a human being. The resulting Turing Test is still used today.

The first definition of the term emerged in 1956 at the Dartmouth Summer Research Project. Here, the basic principles of AI were conceptualized by John McCarthy, Alan Newell, Arthur Samuel, Herbert Simon and Marvin Minsky, among others [McC+06].

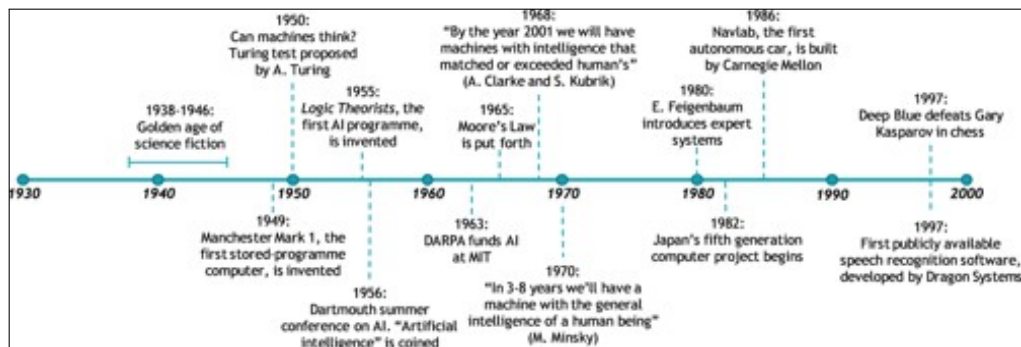


Figure 2.2: Development of AI over time from 1930 to 2000 with relevant milestones from [OEC19]

A lack of funding and declining interest led to a slowdown in research in the 1970s. Due to the increase in cheap computing capacity

¹²These include numbers, texts or images

computing power and the accompanying hardware, AI regained importance. Other milestones in the following years were the chatbot A.L.I.C.E [Wal09] and IBM's Deep Blue chess software [IBM23a], which beat the then world champion Gary Kasparov.

Nowadays, the further development of AI is primarily the subject of computer science. Due to the increase in computing and storage capacity and the further development of technologies such as big data and cloud computing, the performance and availability of AI applications is increasing. Technological progress leads to better and cheaper sensors that capture more reliable and larger amounts of data, which further improves AI software [Enn23]. Terms that occur in connection with AI are machine learning (ML) and deep learning (DL). These form subcategories of the term AI [RN21].

Machine Learning

ML uses statistical learning algorithms to develop intelligent systems. ML systems can learn and improve automatically without being explicitly programmed.¹³ be programmed. Recommendation systems for music and video streaming services are examples of ML. Machine learning algorithms are divided into three categories [Gër19]: supervised, unsupervised and reinforcement learning.

In the case of supervised learning¹⁴ the model is trained using an already labeled¹⁵ data set is used to train the model. This is comparable to a teacher who provides his students with continuous feedback and hints on both the solution path and the correctness of the solution. Two types of problems are considered in supervised learning [RN21]. So-called classification problems¹⁶ and regression problems¹⁷. In classification problems, a prediction is made about the membership of discrete values in a class or group. In contrast, regression problems deal with continuous values, e.g. in the prediction of house prices based on the size of the house, the location and other factors.

¹³This does not mean that the software writes its own source code. But rather the algorithm for recognizing structures and relationships

¹⁴Supervised learning

¹⁵The existence of so-called *ground truth* data

¹⁶dt. Classification problems

¹⁷dt. Regression problems

Unsupervised learning can be seen as the opposite of supervised learning¹⁸[IBM23b] can be considered. Algorithms in this category sometimes work without a complete and cleanly labeled database. The aim of these algorithms is to recognize underlying patterns in input data and to group them in a meaningful way. The most important applications of unsupervised learning include clustering, finding association rules and detecting anomalies.

The last category consists of reinforcement learning algorithms¹⁹. These work by performing actions in a closed environment in order to maximize a predefined total reward. Various actions are carried out and their influence on the total reward is evaluated. After sufficient trials, the algorithm learns which actions bring a greater reward and defines a behavioral pattern [SB18]. Areas of application include robotics and the gaming industry. Relevant algorithms for robotics, such as for the navigation and localization of robots, are discussed in more detail in Part 2.1.5.

Deep learning

DL is not discussed in detail in this paper. A brief definition follows for completeness. DL is the subfield of ML that deals with models based on multi-layered artificial neural networks (ANN). These are modeled on the human brain and how it works. Mathematical models can thus be used to map similar structures. The word deep stands for the fact that the models have many layers of nodes and connections. DL has led to a variety of models, techniques, problem formulations and applications [GBC16]. These include automated driving and the automatic recognition of traffic signs and people as well as medical research in the examination of e.g. MRI scans. Ultimately, DL has also found its way into the smart home sector or consumer applications such as cell phones.

¹⁸unsupervised learning

¹⁹German best-practice learning

2.1.5 LiDAR technology and artificial intelligence

The use of LiDAR sensors has become standard, particularly in the field of autonomous driving. In addition to the automotive industry, robotics and geographical information systems are also utilizing the properties of this technology. This is partly due to the high measurement precision and the **ability** to quickly map large areas²⁰. Furthermore, such systems have a high resilience to external influences. Only reflections or very severe weather conditions can influence the measurement process [LIG20].

LiDAR systems emit light pulses and measure the time it takes for the light beam to return (see equation 2.1). *ToF* stands for Time of Flight, i.e. the time the light beam is traveling, and *c* is the speed of light.

$$D = \frac{\frac{ToF}{c}}{2} \quad (2.1)$$

The direction and distance of what the pulse hits are recorded as a data point. is drawn. The resulting set of data points is referred to as a point cloud²¹. Each point is defined by its position with *x*, *y*, *z* coordinates and additional attributes such as a color for the intensity. The points can then be rendered to create a detailed 3D model of the environment. However, the use of this form of representation poses challenges.

For example, the rapidly increasing amount of data points²². In contrast to the traditional approach to image processing, in which an image usually has the form of a pixel matrix, the point cloud representation has the disadvantage that there is no predefined grid. This means that there is no systematic connection between the individual points. Finally, permutation invariance is another relevant point. This states that the same result can be expected regardless of the order in which the points of the point cloud appear. This makes sense because, for example, the display of the points can vary depending on the direction of the measuring device [Gai+16] [Eng+17] [Hac+17].

In the following, the interaction of AI, in particular ML, and the creation of environment maps as well as the localization of robots will be discussed.

²⁰dt. record²¹ dt.

point cloud²² up to ~
10⁸ points

Mapping and navigation

ML offers the field of robotics a range of tools for the development of complex behaviors. Autonomous mobile systems must simultaneously estimate their position in an unknown environment and create a map. This challenge is also known as the SLAM problem.

Until about 2010, such problems were solved using filter-based approaches [DWB06]. Due to the wide availability and further development of nonlinear optimization algorithms, Graph-SLAM has been the most popular and efficient approach for autonomous mobile systems since 2010 [Hen+20] [IPP21] [Wan+18]. In this approach, important points on the map (so-called landmarks) and positions of a robot are represented by a graph, whereby the SLAM problem can be solved by nonlinear optimization techniques [Cad+16] [Hes+16]. While SLAM algorithms already show progress in simple applications, they can reach their limits in demanding dynamic robot movements or environments [Cad+16] [Spe+21].

The fundamental problem of SLAM is to determine the movement path/trajectory of the robot and the position of all environmental features without knowing the true position of the environmental features or the robot itself [DWB06] [ZS14].

Since the emergence and rapid improvement of cost and efficiency in recent years, LiDAR sensors have played a key role in environmental sensing and mobile intelligent systems [MSD18]. However, the integration of 3D point clouds into existing localization systems poses further problems, especially in terms of accuracy and matching the update rate of other systems used, such as an IMU or camera systems [Bur20].

Graph-SLAM and scan-matching algorithm

LiDAR sensors provide 3D scans of the environment at a high frequency (see section 3.1.4), which - as already mentioned - leads to large amounts of data. The changes between individual scans are often minimal, with only small changes in the translation and rotation of the robot. To reduce the complexity and amount of data,

often only selected scans, so-called keyframes, are used to create a graph that is then used to add new scans. Thus, a new node is only **added** to the graph if the robot's pose has changed significantly in terms of translation and rotation.

With each scan of a LiDAR sensor, a point cloud with measurement points is created, as **mentioned** in the previous section. In order to use these point clouds for mapping or localization, the challenge is to match the current scan with an already known scan of the surrounding area. This approach is called scan matching. Two algorithms have emerged as common options for this problem. The classical Iterative Closest Point (ICP) algorithm and the generalized version (GICP) [BM92]. In addition, the Normal Distribution Transform (NDT) is mentioned in this context [BS03]. In the following, NDT and the Voxelized Generalized Iterative Closest Point Algorithm (VGICP) are considered in more detail. To do this, the basic idea behind ICP should **first** be explained, as the following algorithms are based on it. Algorithm definition 1 describes the step-by-step function of the ICP algorithm.

An alternative approach to ICP is the NDT algorithm. In contrast to the ICP method, this approach does not require an explicit determination of the **point** correlation. Instead, NDT divides the scanned area into a grid and calculates the normal distribution of the points contained in each cell [BS03]. The main advantage of this approach is that no explicit calculation is required to determine the relationships between points. However, the efficiency of NDT depends on the setting of the parameters, e.g. the grid size (2D) or the voxel size (3D). The optimal voxel size essentially depends on the environment and the sensor properties such as the number of points. If an incorrect voxel size is selected, this will lead to errors. In addition, there must be enough points within a voxel, otherwise the determination of the covariance matrix is prone to errors [Koi+21] [HBMO22].

Since the hdl graph slam algorithm used in chapter 4 uses VGICP, the This approach is briefly explained below. VGICP can be seen as a combination of the methods mentioned above. For this purpose, the ICP approach is extended by determining the distribution of points in voxels, similar to the NDT method, which is achieved by aggregating the individual point distributions within the voxels. The covariance matrix in VGICP is then calculated from the individual point distributions. This leads to correct covariance matrices, even for individual points within a voxel.

Algorithm 1: Steps of the Iterative Closest Point Algorithm

Input: Point clouds:

$$P_{Source} = \{x_1, \dots, x_n\}$$

$$Q_{Target} = \{p_1, \dots, p_n\}$$

Output: Transformation matrix T , which transforms the initial point cloud P so that it corresponds as closely as possible to the target point cloud Q .

1 point allocation:

$\forall p_i \in P$ find $q_{nearest}$ in Q by e.g:

$$q_{nearest} = \arg \min_{q_j \in Q} \|p_i - q_j\|$$

2 Calculate centers of gravity:

(I) Focus P :

$$P : p^- = \frac{1}{n} \sum_{i=1}^n p_i$$

(II) Focus Q

$$Q : q^- = \frac{1}{n} \sum_{i=1}^n q_{nearest}$$

3 Calculate covariance matrix H and perform Singular Value Decomposition (SVD) [KL80] to obtain rotation matrix R :

$$H = \sum_{i=1}^n (p_i - p^-)(q_{nearest} - q^-)^T$$

$$H = U \Lambda V^T$$

$$R = V U^T$$

4 Calculation of translation t based on R :

$$t = q^- - R p^-$$

5 Adjustment of points:

$$p_i^0 = R p_i + t$$

6 Repeat 1 - 5 until convergent.

xels. The determination of related pairs of points in VGICP is carried out using a Single-to-multiple-distribution model [Koi+21].

2.2 Challenges and possibilities of four-legged robots

The integration of four-legged robots into everyday life presents a number of challenges and opportunities. Points to be considered include the physical design and adaptability to different types of terrain, the navigation of the robot, safety considerations to avoid collisions, for example, energy management tailored to the scenario and connectivity to possibly other actors in the overall system and ultimately educational opportunities.

With regard to the physical design, a suitable material should be chosen for the limbs and body that is both robust and adaptable. Above all, resilient materials such as aluminum alloys or carbon fiber-reinforced composite

materials have proven themselves over the years due to their properties [Mat19].

As mentioned above, the advantage of four-legged robots is that, in contrast to robots on chains, for example, they can also move on uneven terrain and easily overcome obstacles. By using simulation environments such as the Isaac Gym [DEV23] from NVIDIA, optimal motion sequences can be trained in advance with the help of ML algorithms.

Safety must not be ~~neglected~~ ^{related} the development of robotic systems. In environments where people and animals are primarily present, appropriate protective measures must ~~be~~ ^{be} account. For this purpose, shock absorbers or protective plates can be used to protect both robots and bystanders in the event of a collision. The implementation of additional algorithms for collision detection and avoidance [Mol+13] helps the robot to react to obstacles in good time. This allows collisions to be partially avoided.

As described in more detail in Chapter 3, the robot's battery life is limited.

limited. If rough terrain, stairs or the robot's ability to run are ~~be~~ ^{be} account, this has an impact on the battery life. Efficient energy management leads to a longer operating time. As the Unitree Go1 robot is supplied without a dedicated charging station, future use requires, for example

Energy recovery systems such as regenerative braking systems can also be used. However, such technologies only make a limited contribution to a longer service life. The use of the robot in the university environment also leads to educational opportunities. Technology enthusiasts, professors, students and pupils can use the robot to gain an insight into concepts such as problem solving, programming and robotics.

Finally, the navigation of the robot is a particularly complex project. The fusion and combination of technologies such as SLAM, path planning algorithms such as A* or Rapidly-exploring Random Tree (RRT) as well as efficient obstacle avoidance is essential to enable a robust navigation system of the robot. Other approaches include, for example, terrain analysis. The analysis of terrain features helps the robot to adapt its path to ensure stability and efficient locomotion. Since the advantage of four-legged robots is their efficient movement in rough terrain, they are particularly suitable for use in military or humanitarian sectors. In general terms, the term "service robot" is used here. The term "service robot" refers to robots that actively support people in their tasks, whether as guide dogs, navigational aids in the university environment guide visitors to their destination or for carrying objects.

2.3 Related work

The range of relevant research is diverse and extends from basic topics such as the locomotion of robots to complex cloud systems. In A Study on Locomotions of Quadruped Robot [TKDT14], the walking behavior of quadruped robots in flat terrain is investigated. The experiment analyzes the position of the center of gravity of the robot body in different motion sequences (forwards, backwards, turning, etc.). In Trot Gait Design and CPG Method for a Quadruped Robot [Zha+14] the trot as locomotion is analyzed in more detail. The work follows the idea of the six determinants of gait [HSL13] and designs a trot for quadruped robots.

In addition to basic research, AQuRo: A Cat-like Adaptive Quadruped Robot With Novel Bio-Inspired Capabilities [Sap+21] attempts to make the climbing and overcoming of obstacles of four-legged robots more efficient by imitating a cat structure. A novel paw structure is proposed for this purpose, which makes it easier to climb and overcome obstacles.

This makes it possible to climb steep ladders.

With regard to more complex overall systems, a cloud-based human-robot system is proposed in A Cloud-based Quadruped Service Robot with Multi-Scene Adaptability and various forms of Human-Robot Interaction [Ma+20]. The approach here is to generalize the application possibilities of a quadruped robot in different scenarios as much as possible.

In A1 SLAM: Quadruped SLAM using the A1's Onboard Sensors [CD22], the SLAM functionalities of the A1 robot (the predecessor of Go1) are analyzed and evaluated against Google's Cartographer algorithm. The paper FRL-SLAM: A Fast, Robust and Lightweight SLAM System for Quadruped Robot Navigation [Zha+21] considers a robust, fast and performant SLAM approach specifically for quadruped robots called FRL-SLAM. In particular, this approach counteracts problems such as the body rocking. Finally, [MB+22] deals with a consideration of various existing SLAM algorithms and an evaluation of these.

3 Commissioning the Unitree Go1 robot

The central content of this chapter is to provide an `overview` of the hardware and software components of the Go1 robot in its factory state. The features previously mentioned in chapter 2.1.2 are prescribed here. This is followed by an analysis of the robot's software and simulation options.

3.1 Hardware components

Figure 3.1 shows various views of the Go1 robot. When folded, it has a size of $0.54 \times 0.29 \times 0.1$ (length, width, height) meters. When stationary, $0.645 \times 0.28 \times 0.4$ meters. If the battery is inserted, the weight is 12 kg. It is estimated at an additional 1.5 kg. The maximum load capacity is specified as $\sim 5 - 10$ kg²³. The three main components of the robot are the body, head and the four limbs that serve as actuators.

3.1.1 Overview

The body contains the majority of the installed hardware. This includes the internal hardware components such as the battery, sensors and cameras as well as the `joints` including the motors of the leg actuators. As can be seen in Figure 3.1, the joints and the motors for controlling the `joints` are installed at the front and rear of the body. The motors `are` responsible for the rotation of the hips and the

²³The *EDU* version used here has a higher load capacity. With conventional models, it is 3 kg.

thigh. Four further joint/motor components are located at the upper end of the legs; these are responsible for bending. They are part of the limbs and are not permanently attached to the body.

The motors are the GO-M8010-6 model of the permanent magnet synchronous motor (PMSM) design with a maximum torque of $23.7 \text{ N} \cdot \text{m}$ (Newton/meter). If you look at the product sheet of the motor, features such as a built-in Field Oriented Control (FOC) control unit, temperature sensors and an Absolute Value Encoder (AVE) are listed.²⁴ are listed. One motor weighs 530 g and communicates via an RS-485 interface [Robb]. Together, the three motor/joint components form a unit with three degrees of freedom (DOF)²⁵. The robot thus achieves a total of 12 DOF, which ensures appropriate maneuverability. Ta-

Table 3.1 provides an overview of the range of motion of the joints.

Joint designation	Range of motion
Lateral hip	$-40^\circ \sim +40^\circ$
Anterior hip	$-218^\circ \sim +45^\circ$
Knee	$\text{joint}+24^\circ \sim +132^\circ$

Table 3.1: Range of motion of the joints

When looking at the limbs, the upper and lower parts (thigh and knee) are connected by a knee joint, which ensures extension and flexion of the leg construction. Pressure sensors are located at the end of the actuators²⁶. These are used to measure the load values at the point of impact and pass them on to the Main Control Unit (MCU).

There are a total of five stereo camera systems on the robot, each with a viewing angle of $150^\circ \times 170^\circ$. Two of them are installed on the head. A further three are located on the left and right sides and the abdominal region of the body. In addition to the head region, it should be noted that two LED strips are installed on the sides, which display different color tones, and a loudspeaker located back on the head. Figure 3.4 shows a top view of the disassembled head section.

²⁴Identify the absolute position as well as the speed and direction of the encoder.

²⁵dt. Degrees of freedom

²⁶Also only available in the *EDU* version

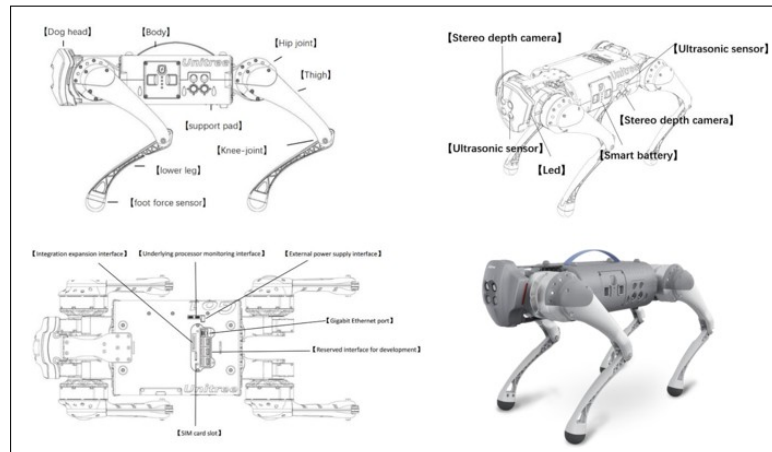


Figure 3.1: Overview of the components of the Go1 robot from [Robe]

According to the data sheet [Robe], four ultrasonic sensors are installed around the head and torso. On closer inspection, however, only three were found. According to the data sheet, an ultrasonic sensor placed at the rear is planned, but not installed and only planned for future installations. The detection range of the sensor is between 5 - 200 cm.

To control the robot, a remote control is included that connects to the robot via radio signal. Pre-programmed tricks and functions such as climbing stairs and dancing are stored on the remote control to demonstrate the range of possible movements. The follow-me function can also be activated using a transmitter that can be placed on the belt, for example. The robot automatically follows the transmitter and avoids obstacles. However, some tests have shown that this function is more suitable for outdoor areas than in enclosed spaces. After the main external sensors and larger components have been considered, a look is taken at the connections placed at the rear.

3.1.2 External connections

Various connections are available on the upper side of the cabinet to the outside. Figure 3.2 shows a plan view of the available connections. The connections include 3 x HDMI and 3 x USB-A connections. These allow access to the NVIDIA Jetson's and the Raspberry Pi. There is also 1 x RJ-45 and 1 x SIM card slot. In addition to an XT-30U (24V, 30A) connection to connect the

To supply the robot with power via a power supply unit, there are also 2 x USB C ports. These provide an interface to the RS-485 connection of the motors. With The last connection is a 40-pin Centronics connector, which can be used to access various connections that are connected to the hardware inside the dog. For example, the installation of a GPS system would be conceivable here. This concludes the external consideration of the hardware installed on the robot. The following section provides an insight into the internal structure of the robot and the hardware it contains.

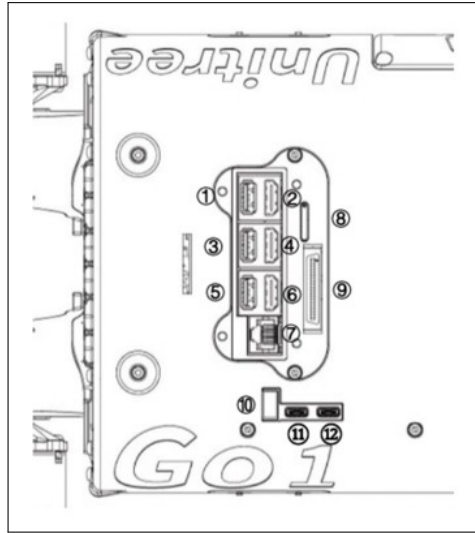


Figure 3.2: Connections and ports on the surface of the Go1 robot from [Robe]

3.1.3 Internal consideration

Figure 3.3 shows the inside of the robot after the body has been opened. The largest part of the interior is occupied by the slot for the battery. The Battery Management System (BMS) is located behind it. To the left of the battery slot are an NVIDIA Jetson Nano, the Raspberry Pi and an NVIDIA Xavier NX. Their main task is to manage the assigned camera modules and ultrasound sensors. The NVIDIA Jetson Nano in the head area also has the task of capturing and forwarding the ultrasonic data from the forward-facing ultrasonic sensor. Table 3.2 shows an overview of the main components of the robot and their tasks. Tables C.1, C.2, C.3 in the appendix provide an overview of the robot's hardware components.

the main specifications of the individual components. The paper Integration of a Unitree Go1 Quadruped Robot into a University Ecosystem [Leh23] provides an extended view of the components and their specification.

Component	Task
Raspberry Pi	Web hosting, app connection, monitoring, libraries, ultrasound, wifi module
Nano 1 - 2	LED control, audio output, ultrasound, video services
Xavier NX	Video services, AI processes

Table 3.2: Main components and their tasks in Go1

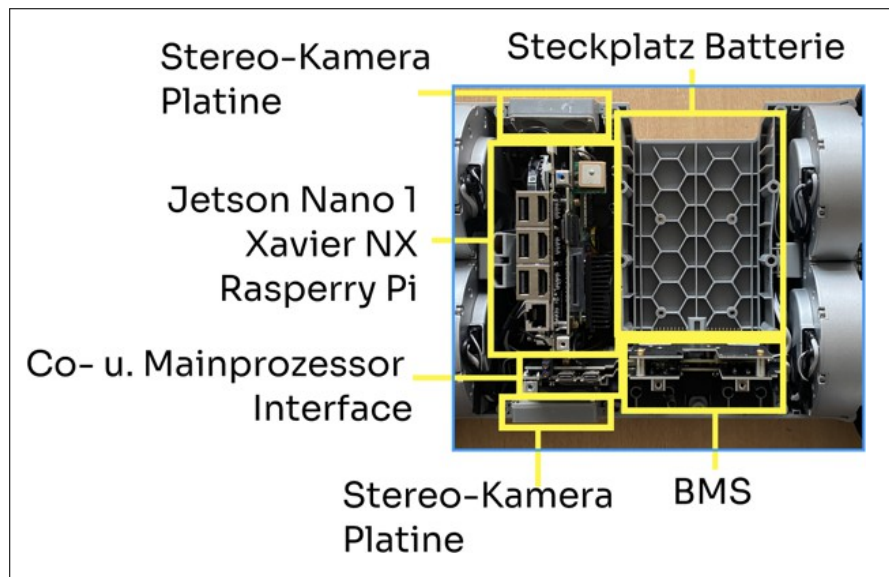


Figure 3.3: Top view of the internal structure of the Go1 robot and assignment of components within it

If the robot is not operated via an external power supply using the previously mentioned XT-30U, this is done using the battery. A power supply unit or adapter to supply the robot with power is not included in the scope of delivery. The GO1- BT03 battery is a lithium-ion battery that is available in three different versions. With 4500 *mAh* (milliampere hours), as well as the standard version with 6000 *mAh* and the long-range version with 9000 *mAh* [Robc] [Ele]. In the case of the robot used, this is the **standard** version. During the

Laboratory tests have shown that a charging cycle to fully charge the battery takes ~ 1.5 hours. Depending on the utilization of the robot, i.e. which functionality is used (tricks, climbing stairs, etc.), recharging is necessary after ~ 1 hour.

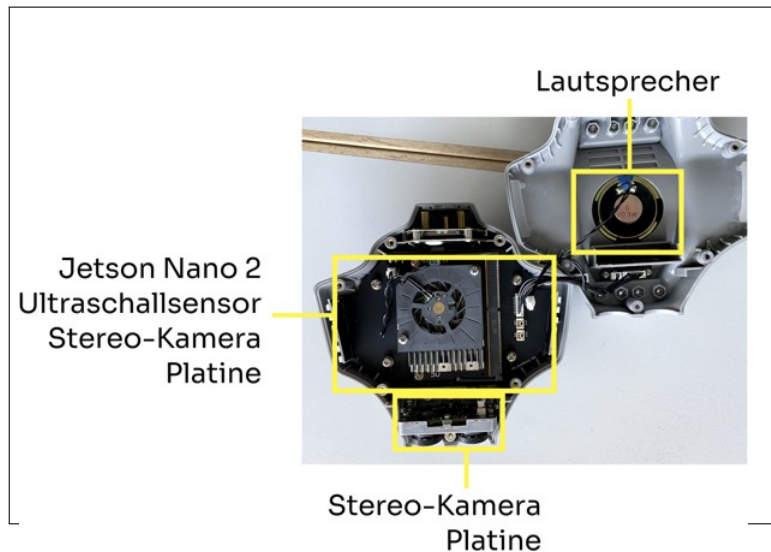


Figure 3.4: Top view of the internal structure of the Go1 head and regulation components contained therein

In order to gain a better understanding of the interaction of the individual components, it is useful to represent them in a diagram. Figure 3.5 shows an architecture diagram of the components of Go1 and their communication channels. It should be noted here that the structure and the control and monitoring systems are modeled on the MIT Cheetah and are even identical in parts. A comparison with the documentation of the MIT Cheetah is therefore advisable.

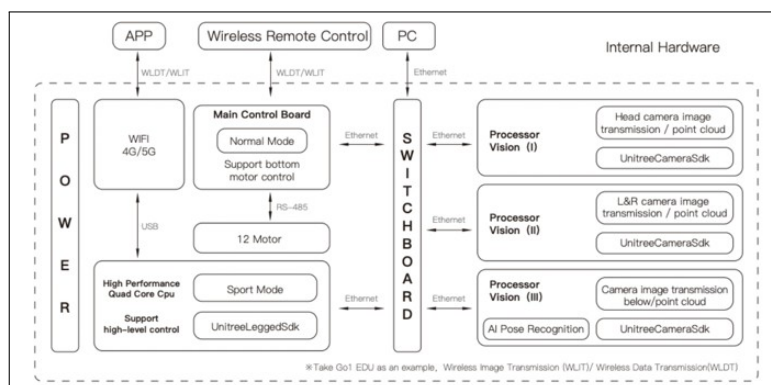


Figure 3.5: Architecture diagram of the Go1 from [MAV]

The diagram in Figure 3.6 shows the communication architecture of the MIT Cheetah in detail. In the Unitree design, the Raspberry Pi as well as the NVIDIA Jetson Nanos and the NVIDIA Xavier NX replace the interface to the MCU labeled UP. The RS-485 network and motion processing are almost identical. In the case of the Go1, the MCU is an STM32H743 with MicroROS [ROS] [MAV]. The BMS can only be read and manipulated using manufacturer libraries. Direct access to the MCU is only possible in read mode via third-party libraries.

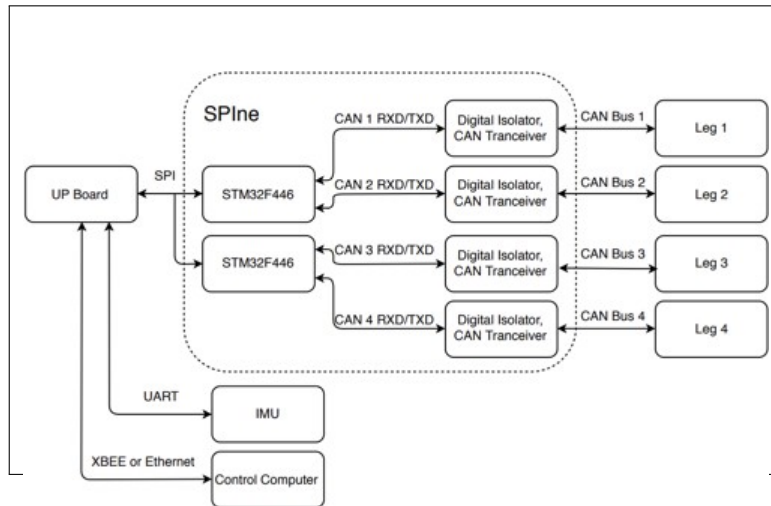


Figure 3.6: Architecture diagram of the MIT Cheetah from [Kat18]

Further details regarding the implementation of individual communication paths and control systems are omitted below, as these play a subordinate role for the content of this work. The version A low cost modular actuator for dynamic robots [Kat18] describes the underlying communication architecture between the motors and the MCU in more detail. Also of interest in this context are the publications of the Biometric Robotics Lab²⁷. Further resources for the Go1 are listed in chapter 3.2.4. In the following, the topic of connectivity is briefly considered and features of the LiDAR sensor used are explained before the software components are then considered.

²⁷To be found at: <https://biomimetics.mit.edu/publications>

Connectivity

A detailed consideration of the robot's connectivity is given in Integration of a Unitree Go1 quadruped robot into a university ecosystem [Leh23], where the 4G/5G capabilities of the robot are also considered.

All directly controllable components are connected to a switch via Ethernet - as can be seen in Figure 3.5. The robot's switch can be accessed from outside via a connection (see Figure 3.2). All components are registered in the network with the IP 192.168.123.0/24. Table 3.3 shows the distribution of the IP addresses.

In addition to the wired connection, the Raspberry Pi also has a Wireless Wide Area Network (WWAN) module to publicize the network to the outside world. This wireless connection can be used as an access point to the robot system. The Raspberry Pi serves as the gateway.

Component	IP address
MCU	192.168.123.10
Raspberry Pi	192.168.123.161
NVIDIA Jetson Nanos:	192.168.123.13/14
NVIDIA Xavier NX	192.168.123.15

Table 3.3: Determined IP address of the robot hardware

3.1.4 Overview RS-Helios-16P LiDAR

The LiDAR mounted on the Go1 using rails is the RS-Helios-16P from RoboSense. It uses 16 beams to measure the surroundings in a 360° environment. The range is ~ 150 m, whereby the precision decreases with increasing distance²⁸. The scanning frequency is 10 Hz (Hertz) or 20 Hz at a rotation speed of 600 or 1200 rounds per minute (RPM)²⁹. Table 3.4 lists further technical features [Roba].

²⁸±2cm (1m to 100m); ±3cm (10cm to 1m); ±3cm (100m to 150m)

²⁹dt. revolutions per minute

Property	Value
Laser wave length	905nm (nanometer)
Vertical field of vision	30° ; (-15° ~ +15)°
Horizontal resolution	0, 2° / 0, 4°
Vertical resolution	2°
Data throughput	~ 288,000 points / ~ 576,000 points (double return)
Output	UDP packets via Ethernet
weight	~ 900g

Table 3.4: Table on the technical specifications of the RS-Helios-16P sensor from [Roba]

The rslidar sdk [RL] is helpful when using the LiDAR. The library provides functions for interacting with the LiDAR. These include sending point cloud data using ROS and recording the point cloud in ros- bag format. The RSView tool can be used to visualize the Point Cloud Data (PCD). It offers the user a range of options. These include the transfer of live data, the recording and playback of recorded files and some tools for editing (crop, distance measurement, etc.) the point cloud. During some tests, the data could be visualized well with the help of the program, but recording and saving the data in the point cloud was difficult.

formats such as PCD, LAS, PCAP is not possible, as an error message is displayed each time (see Appendix Figure A.6) was output by the program. In addition, according to the user manual, it should be possible to record live data, but the button required for this is not implemented in the program.

As mentioned above, the LiDAR uses User Datagram Protocol (UDP) packets for communication. Two specific protocols are used. Namely Main Data Stream Output Protocol (MSOP) and Device Information Output Protocol (DIFOP). The MSOP protocol sends the sensor data, i.e. the data of the point cloud (distance, angle, etc.), while various configuration information of the LiDAR is output via the DIFOP channel. The packet size of both protocols is 1248 bytes. The standard ports are 6699 for MSOP and 7788 for DIFOP. An exact structure of the packages can be found in Figure 3.8. The detailed procedure for configuring and implementing the LiDAR is described in Chapter 4.

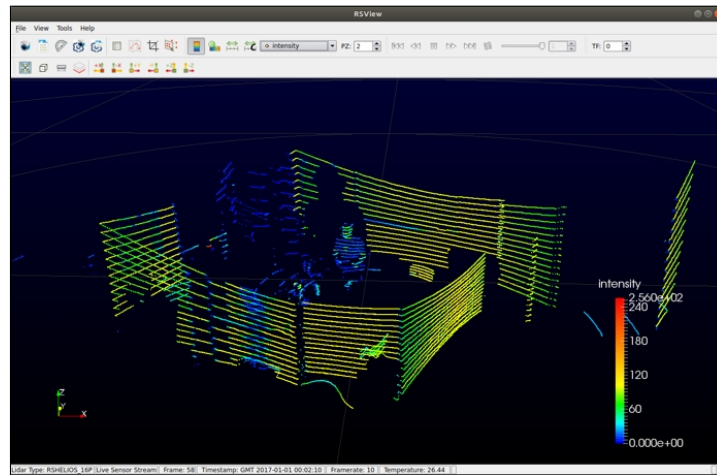


Figure 3.7: Excerpt from the RSView program

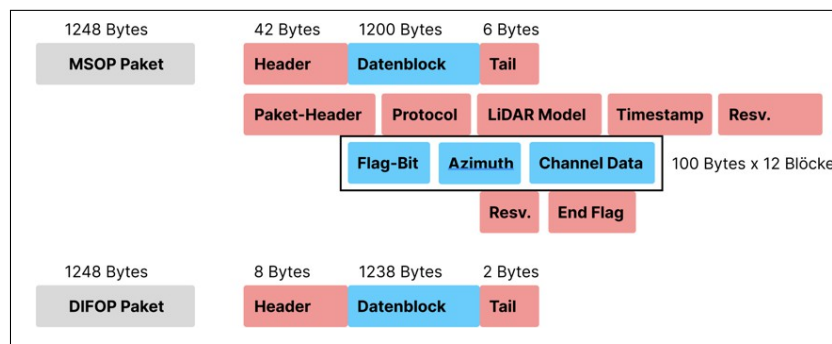


Figure 3.8: Distribution of the UDP packets of the RS-Helios-16P sensor

3.2 Software components

The examination of the NVIDIA Jetson Nanos and NVIDIA Xavier NX using a dump revealed that the operating system is an Ubuntu distribution of version 18.04.5 LTS (Bionic Beaver). The operating system (OS)³⁰ of the Raspberry Pi is Debian GNU/Linux 10 (buster). There are various ROS packages on the NVIDIA Jetson Nanos, NVIDIA Xavier NX and Raspberry Pi. Figures A.1 and A.2 in the appendix show a list of the installed ROS packages. The ROS packages on the NVIDIA Jetson Nanos and the NVIDIA Xavier NX are identical.

An app for iOS and Android is also available for the robot. The app can be installed via iOS using the TestFlight application. If a wireless connection

³⁰German operating system

If there is a connection to the robot's network, this provides a number of functions. The main functions of the app include a simulator and a status display for the BMS, the joint positions and the current status of the robot (versions of firmware, app, etc.).

The simulator can be used to create a digital clone of the Go1 in a limited space. move around. It is controlled via the cell phone display. A direct

It is possible to control a real robot using the app. A button provides the option of displaying the video stream of the five camera systems. An interface for graphical programming using blocks is also available. However, the options here are limited. Simple commands such as setting the LEDs to a specific color after a timer has expired are possible. The app is also used to test the loudspeakers. If a LiDAR is placed on the robot and connected, it is possible to start Unitree's SLAM procedure. Attempts to start the application with the factory settings were unsuccessful. Figure A.5 in the appendix shows an excerpt from the iOS app. The web application provides the same functionality and has only minor differences. It is possible to update the firmware using the web application. In addition, the AI software promised in the advertising video can be displayed via the web application. However, the limited resolution of the camera and an unreliable recognition algorithm make this application almost unusable. An illustration of the built-in AI recognition can be found in Appendix A.3 and A.4.

3.2.1 Analysis of the UnitreeCameraSDK

The UnitreeCameraSDK can be used to access various functions of the built-in camera. This section serves to take a closer look at the library.

Depth and color images can be captured using the Software Development Kit (SDK). It is also possible to obtain the calibration parameters of the camera. As dependencies³¹ are CMake version 2.8 or higher, OpenGL, GLUT and X11. OpenGL is a cross-language and cross-platform programming interface for rendering 2D and 3D graphics. The OpenGL Utility Toolkit (GLUT) is a library that mainly performs input and output (I/O) tasks with the host operating system. X11 supports bitmap displays. It provides the

³¹dt. Dependencies

basic framework for a Graphical User Interface (GUI) [Robd].

Instructions on how to use the SDK cannot be found on the associated GitHub page <https://github.com/unitreerobotics/UnitreecameraSDK>. Instead, Unitree provides a video on the YouTube platform explaining how to use it. However, it should be noted that this tutorial is incomplete. A link to an external document below the video provides further details. These are also incomplete or only available in Chinese. The following is an introduction to the library.

On the robot (transmitter)

First of all, some ROS nodes must be switched off, as they could otherwise interfere with the SDK. Use while the nodes are still in operation is not possible.

```
1 ps - aux | grep point_cloud_node | awk '{ print $2 }' | xargs kill -9
2 ps - aux | grep mqtt Control Node | awk '{ print $2 }' | xargs kill -9
3 ps - aux | grep '}' | xargs kill -9
4 ps - aux | grep live_human_pose | awk '{ print $2 }' | xargs kill -9
   rosnode | awk '{ print $2 }' | xargs
```

Listing 3.1: Bash code for switching off the previously mentioned ROS nodes

The next step is to make some changes to the configuration file for the parameters on the robot. The file `trans_rect_config.yaml` is available for this purpose. Two parameters are of particular importance here, namely `IpLastSegment` and `Transmode`. The former defines the IP address of the recipient and the latter specifies the type of transmission.³² transmission.

```
1 IpLastSegment : !! opencv-matrix rows
2 : 1
3 co ls : 1
4 dt : d
5 data : [ 9 9 .]
6 Transmode : !! opencv-matrix
7 rows : 1
8 co ls : 1
9 dt : d
10 data : [2.]
```

Listing 3.2: Excerpt from the configuration file for the camera parameters

³²0-camera right, 1-stereo, 2-camera right (rectified), 3-stereo (rectified)

The library can then be compiled. If a build directory is not yet available, this must also be created.

```
1 cd UnitreecameraSDK
2 mkdir build
3 cd build
4 cmake ..
5 make
```

Listing 3.3: Creating the build directory and compiling

The example putImageTrans.cpp example provided in the SDK for starting the Transmission can now be used. One with your own comments version can be found in the appendix under B.1.

```
1 ./ bins / example_putImageTrans
2 [ UnitreeCameraSDK ][ INFO ] Load camera flash parameters
3 OK! [ StereoCamera ][ INFO ] Initialize parameters OK! [
4 StereoCamera ][ INFO ] Start capture . . . hostIp + port
5 String : host =192 .168 . 123. 99 port =9201
6 Framerate set to : 30 at NvxVideoEncoder Set ParameterNv MMLiteOpen
7 Block : BlockType = 4
8 ===== NVMEDIA : NVENC =====
9 NvMMLiteBlockCreate : Block : BlockType = 4
10 H264 : Profile = 66 , Level = 40
```

Listing 3.4: Executing the binary and confirming the camera stream

The robot now sends UDP packets to the previously specified IP address in the robot's network. The next step involves receiving the files and playing them back.

On the receiver system

If there is no build folder in the UnitreecameraSDK directory, the step from Listing 3.3 must be repeated. The supplied file example getimagetrans.cpp is used to receive the video stream. A more detailed look at the file with comments can also be found in the appendix under B.2.

Depending on whether the receiver device is a 64-bit AMD architecture or ARM-based processor, the decoder settings in the example getimagetrans.cpp file must be modified.

For AMD-64 based processors:

```
1 std :: string udpstrBehindData = " ! application / x- rtp , media =
2 video , encoding - name = H264 !
3 rtph264 depay ! h264 parse ! avdec_h 264 ! videoconvert ! appsink ";
```

Listing 3.5: Decoder settings for AMD based processors

For ARM based processors:

```
1 std :: string udpstrBehindData = " ! application / x- rtp , media =
2 video , encoding - name = H264 !
3 rtph264 depay ! h264 parse ! omxh264 dec ! videoconvert ! appsink ";
```

Listing 3.6: Decoder settings for ARM based processors

Once the `./bins/example getimagetrans` file has been successfully executed, the camera image is displayed. It should be noted that due to time constraints and a defective camera board, it was not possible to successfully transfer the other options provided, e.g. depth image, point cloud. Although these were tested during the analysis, an error message always appeared. This error message is documented in the appendix under A.7. A first assumption is that there is a discrepancy between the configuration of the camera and the OpenGL decoder.

After the camera SDK has been examined, instructions for the Go1's ultrasonic sensors follow.

3.2.2 Analysis of the ultrasonic sensors

Unitree does not provide detailed instructions on how to use the ultrasonic sensors. Only a `readme`³³ file was found, which allows conclusions to be drawn about the use. The following instructions are based on tests carried out on the robot and may therefore be incomplete. The individual steps for evaluating the ultrasound data are described below.

First, a list of the ROS topics running on the Raspberry Pi is generated using `rostopic list`. The complete output can be found in the appendix under B.4.

³³A file that usually contains instructions for using the underlying software.

The topics determined in Listing 3.7 are interesting.

```

1 pi@raspberrypi : rostopic list
2 ...
3 / range_ultrasonic_face
4 / range_ultrasonic_left
5 / range_ultrasonic_right
6 ...

```

Listing 3.7: Section of the ROS Topics running on the Raspberry Pi

A closer look at the structure of the topics using the command `rostopic echo /range-ultrasonic right` resulted (Listing 3.8):

```

1 pi@raspberrypi : rostopic echo / range_ultrasonic_right
2 ---
3 header :
4   seq : 14996
5   stamp :
6     secs : 1636154800
7     nsecs : 45547136 frame :
8   id: " camera_right "
9 radiation_type : 0
10 field_of_view : 1 . 57079637051
11 min_range : 0 . 050000007451
12 max_range : 2.0
13 range : 0 . 363781124353
14 ---

```

Listing 3.8: Structure of the ROS topic for the suspected ultrasonic sensors

The range, min range and max range are in meters. This assumption results from the fact that during the test, there was a wall on the right side of the robot at a distance of ~ 30 cm. Investigations of the Raspberry Pi directory allow further conclusions. The following directories can be found under the path `pi@raspberrypi: home/Unitree/autostart/utrack` (Listing 3.9).

```

1 pi@raspberrypi : ~/ Unitree / autostart /
2                                     utrack1
3 s - la total 36
4 drwxr - xr - x6 pi pi 4096 Feb 16    2022 .
5 drwxr - xr - x 14 pi pi 4096 Aug 18 05: 11 ...
6 drwxr - xr - x4 pi pi 4096 Feb 16    2022 catkin_utrack

```

```

6 drwxr - xr - x4 pi pi 4096 Feb 16      2022 g1_ukd 2udp
7 drwxr - xr - x4 pi pi 4096 Feb 16      2022 g1_ultrasonic_lcm
8 -rwxr - xr - x1 pi pi   968 Feb 16      2022 run . sh
9 drwxr - xr - x6 pi pi 4096 Feb 16      2022 ultrasonic_listener_example
10 -rwxr - xr - x1 pi pi   138 Feb 16      2022 utrack . sh
11 -rw -      r--r--1 pi pi 5 Feb 16      2022 version . txt

```

Listing 3.9: Output of the utrack directory in the Go1 autostart folder

The folders `g1_ultrasonic_lcm` and `ultrasonic_listener_example` seem to be relevant for the ultrasonic sensors. The latter directory also contains the aforementioned readme file. However, the steps described therein were not successful in reading out the ultrasonic sensors. Although the data is generated, it is not received. It should be noted that the sensor values can be published and subscribed to using Lightweight Communications and Marshalling (LCM). As an alternative approach, it is possible to read out the data from the ultrasonic sensors using the unitree legged sdk. For this purpose, the `example_walk` file is modified so that the `rangeObstacle` method is output instead of the pre-programmed commands (see Appendix B.3). After the file has been compiled and executed, the values are displayed in the terminal (Listing 3.10).

```

1 rangeObstacle = [ Head : 1. 725300 , Left : 0. 273141 , Right : 0 . 363781 ]

```

Listing 3.10: Example output of the ultrasonic sensor values

The reason for the faulty transmission using LCM could not be determined during the tests. Various situations (e.g.: `triggerSport` on/off, recompiling the project, etc.) were tested, but without success. The binary files in the `g1_ultrasonic_lcm` directory are also questionable. Starting these also brought no further findings. No source files are available for this, nor is there any explanation. Further investigations in this area are therefore conceivable for future work.

3.2.3 Simulation of the Unitree Go1

To test some functions of the robot, such as controlling individual motors, it makes sense to use a simulation environment to prevent damage. The purpose of this section is to explain the steps for simulating the Go1. The unitree ros package offers all the essential functions for simulating the robot. In the meantime, Go1 robots are also integrated in all major frameworks and tools. For example, in the CHAMP framework [cha] or in NVIDIA's Omniverse software [Nvi]. Tests of the simulation using the Omniverse software were not carried out due to time constraints, but could be considered in the future.

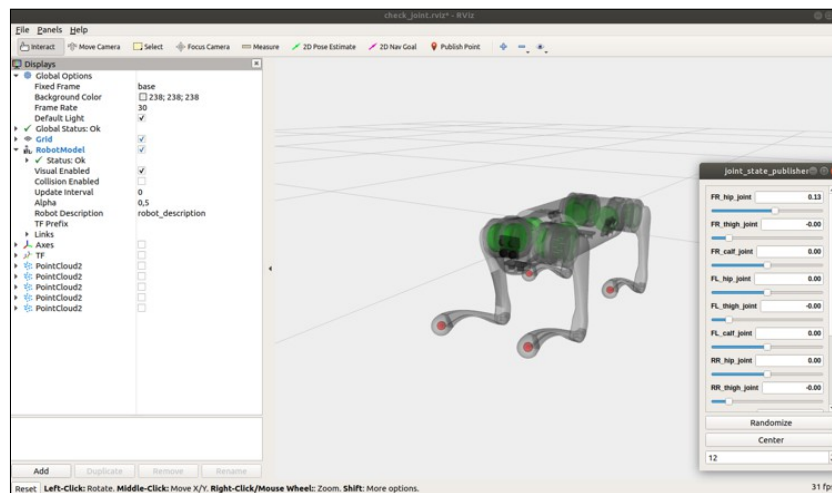


Figure 3.9: Excerpt from the RViz program for visualizing the joint angles of the Go1

The simulation was carried out on a computer with Ubuntu 18.04, 62 GB RAM, a Ryzen 9 3900X 12-core processor and an NVIDIA GeForce RTX 2090 Super with 8 GB memory. CUDA was also installed on the system for hardware acceleration. The simulation was carried out using Gazebo and RViz. Gazebo and RViz are tools that are used in robotics and simulation environments to support the development and visualization of robotic systems. Gazebo provides a physics-based simulation environment in which robots, sensors and other environmental objects can be visualized. RViz, on the other hand, is a component of the ROS environment and serves as a visualization tool for sensor data, robot models and other objects.

or laser scans. The Unified Robot Description Format (URDF) is used to model the robot. An Extensible Markup Language (XML) file is used to define physical properties such as kinematics, geometry, structure and components of the robot. Once all the dependencies have been installed, the `roslaunch go1 description laikago rviz.launch` command starts the visualization program as shown in Figure 3.9. The individual joint angles can now be simulated using the sliders. The effect of the angle changes is displayed immediately. The simulation of the complete robot is carried out using Gazebo. For this purpose, an identical image of a real Go1 is rendered. Furthermore, different environments (planar plane, stairs, houses, etc.) can be displayed. To launch the environment, `roslaunch unitree gazebo normal.launch rname:=go1 wname:=stairs` is used. The parameter `rname:` specifies the model name and `wname:` the file name of the environment to be loaded. First, the robot is loaded into the environment in the horizontal state.

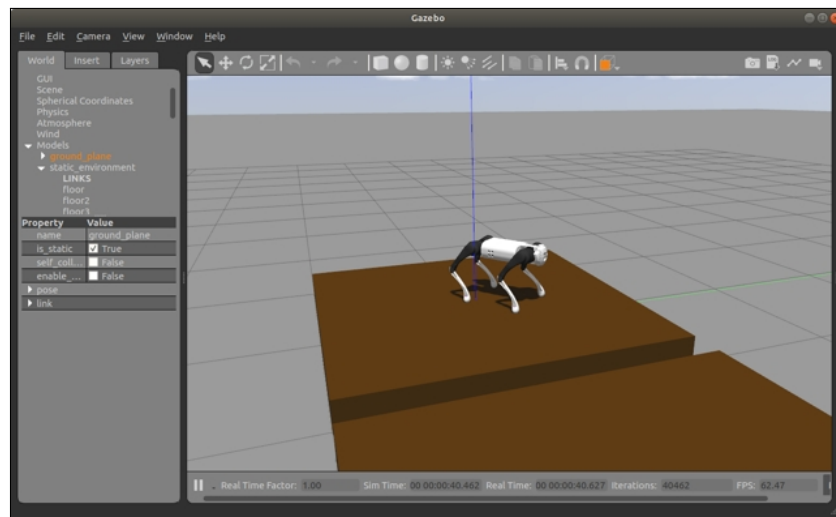


Figure 3.10: Excerpt from the Gazebo program for visualizing the Go1

loaded. To activate the motors, another ROS node is required, which is started using `roslaunch unitree controller unitree servo`. The robot is now set to a stationary state. Further nodes can now be used to simulate robot movements, e.g. `roslaunch unitree controller unitree external force`. External influences such as a bump can now be generated using the arrow keys. Test functions for e.g. SLAM in the simulation are not available from Unitree and must be developed independently.

3.2.4 Additional resources for the Go1

The website <https://www.docs.quadruped.de/> provides an initial introduction to the handling of the Go1. In addition to the documentation, further manuals can be found here, for example the Z1 robot arm. The scope ranges from an overview of the robot to individual steps for simulating the robot. An overview of various manuals can be found at <https://www.docs.quadruped.de/projects/go1/html/operation.html>, for example on using the camera SDK.

The official GitHub repository of Unitree can be found at <https://github.com/unitreerobotics>. All essential SDKs can be found there. This includes `unitree ros`, a package for simulating the robot in low-level mode, i.e. controlling the torque, position and angular velocity of the robot joints. A simulation of the high-level functionalities, including walking, is not included. The simulation is performed by Gazebo and requires ROS-Melodic or ROS-Kinetic. Another library is the `unitree legged sdk`. This **enables** functions to influence the control of the robot. This includes both high-level and low-level commands. The `unitreecameraSDK` - as previously analyzed in more detail - provides various options for accessing the robot's stereo cameras. The SDK **enables** streaming of depth and color images and **provides** information on calibration. In addition to the libraries already mentioned, there are a number of others. These include the `unitree actuator sdk` to control the installed motors individually and the two packages `unitree ros to real` and `unitree ros2 to real` to send commands to the robot using ROS.

It is important to mention at this point that although the official documentation both on the website and in the GitHub repositories provides a good introduction to using the robot, it is largely incomplete, incomprehensible or only available in the original Chinese language. It is therefore advisable to take a look at the alternative sources listed below for further information, albeit with a critical eye, as much has not been confirmed or **checked** by Unitree.

The HangZhou Yushu Tech Unitree Go1 [MAV] repository is worth mentioning here. In this context, the Go1 robot is examined in detail from several points of view. Under the Slack group The Dog Pound animal control for Stray robot dogs, other enthusiasts and users can be found, who can exchange information with each other.

The forum is also a good place to exchange information on various aspects of the robot, be it the internal structure of motors, hardware and recommendations for handling the SDKs or independent implementations to extend the functionality. The forum <https://community.droneblocks.io/> is also a good place to go in this regard. Finally, the DroneBlocks YouTube channel (can be found at <https://www.youtube.com/@droneblocks>) and the associated GitHub repository <https://github.com/dbaldwin> are a good place to go to deepen your knowledge of the programming options for Go 1.

4 Implementation

This chapter deals with the implementation of the project to realize an intelligent navigation system on the Unitree Go1 using the RS-Helios-16P LiDAR. The individual steps such as the preparation of the robot, the selection of the test environment and the test runs are prepared and explained. The various scenarios are then described, which are then used to create a map of the environment using LiDAR.

4.1 Description of the experimental setup

To implement the project, the LiDAR is mounted on the robot. A rail system on the back is used for this. It is worth mentioning at this point that the manufacturer does not provide any instructions for mounting the sensor. The design is therefore based on our own ideas. The screws used for this are pushed into the rail system with the head. To prevent the screw head from turning, a few millimeters are ground off to the side. The LiDAR is then attached to the screws and tightened with screw nuts. Finally, the sensor is connected to the robot's interfaces, the RJ-45 and XT-30U for power output. Figure 4.1 shows the finished assembly.

The next step is to configure the basic settings of the LiDAR. The manufacturer provides a web interface for this purpose, which can be accessed using the IP address of the sensor. For the tests, the IP was set to 192.168.123.60. Appendix A.8 shows an excerpt from the web interface settings. The Destination IP Address parameter was set to the IP address of the NVIDIA Xavier NX. MSOP and DIFOP ports remain unchanged.

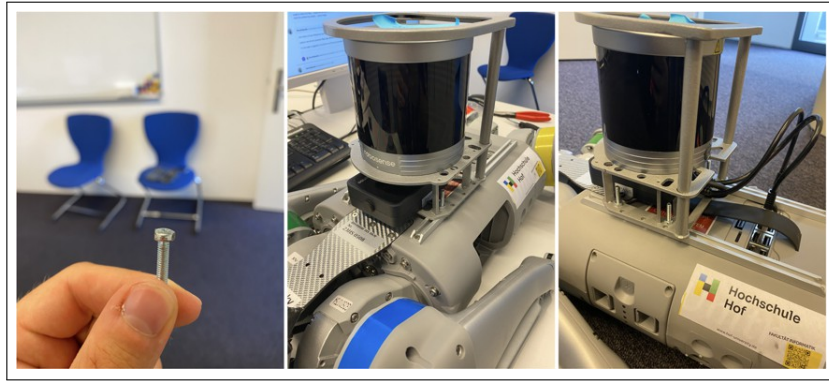


Figure 4.1: Mounting the LiDAR sensor on the Go1

In order to obtain a test environment that is as diverse as possible, two scenarios were chosen for the tests, one in a building and one outdoors. The foyer of the Institute for Information Systems at Hof University of Applied Sciences serves as the indoor test track. The outdoor test is located directly around the Alfons-Goppel-Platz at Hof University of Applied Sciences. Both test routes offer suitable environments, such as stairs or rough terrain and inclines. At the beginning of the tests, it was ensured that the robot's battery was fully charged. Table 4.1 lists the key data of the selected tests. The average speed is calculated from the estimated distance covered and the recording time.

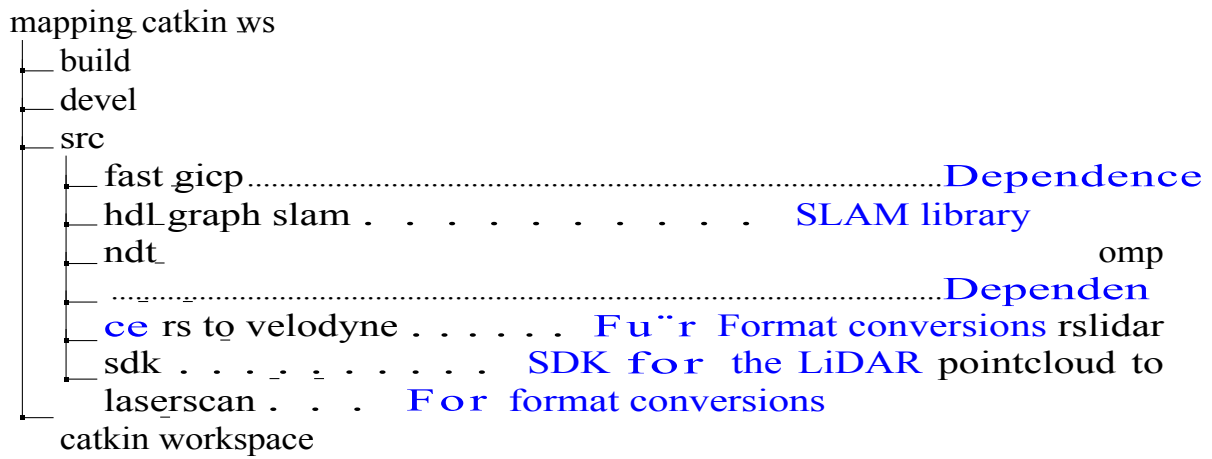
Scenario	Time t (seconds)	~Distance m (meters)	Speed v^- (m/s)
Indoor (IISYS foyer)	249	124	0,497
Outdoor (HS-Hof)	615	585	0.951

Table 4.1: Characteristics of the two measurement scenarios

It is clear here that the mapping process in the indoor area occurs at a significantly lower average speed v^- . In contrast, the average speed in the outdoor area is almost twice as high. An evaluation of the two scenarios follows in Chapter 5. A note applies to the distance. Due to the fact that it was not possible to determine the exact distance traveled, this is an estimate.

4.2 Carrying out the experiment

For the implementation of the real-time SLAM algorithm on the robot, hdl graph slam³⁴ was used. This is an open-source ROS library for 3D LiDAR applications. It is based on a 3D graph SLAM with NDT-like scan matching methods to determine the trajectory. The hdl graph slam is suitable for six degrees of freedom. In addition to scan matching, other sensor inputs such as IMU or GPS can also be used as boundary conditions for motion determination. Since Unitree does not provide a direct ROS interface for tapping IMU data, this option is not available. For the implementation on the robot, a separate catkin workspace is first created³⁵ on the robot's NVIDIA Xavier NX. This has the following structure:



Furthermore, some changes are made in the SDK of the LiDAR sensor. Two different types can be selected as the format type for the emitted points. XYZI and XYZIRT. Table 4.2 summarizes the recording options for the measuring points.

Descriptor	Descriptor
X,Y,Z	Spatial coordinates of the LiDAR point
I	Reverse of the measuring point
R	Ring number from which the point originates
T	Timestamp

Table 4.2: Description of the recording options for the measuring points

³⁴³⁵ https://github.com/koide3/hdl_graph_slamcatkin is a CMake based build tool for ROS

As the latter setting provides considerably more information, the file format in the file under `src/rslidar_sdk/CMakeLists.txt` is adjusted. The type of build tool is also set to CATKIN.

```
1  set ( POINT_TYPE XYZIRT )
2  set ( COMPILE_METHOD
    CATKIN )
```

Listing 4.1: Changes in the LiDAR SDK

Further settings must be made under `src/rslidar_sdk/config/config.yaml`. The parameters `send_packet_ros` and `send_point_cloud_ros` are set to false and true. The lidar type is assigned the model used, RSHELIOS 16P. Finally, the name of the ROS topic can be specified via which the LiDAR should publish the point cloud data. The measurement is sent via the `/rslidar_points` topic.

```
1  send_packet_ros : false
2  send_point_cloud_ros : true
3  ...
4  lidar :
5  - driver :
6      lidar_type : RSHELIOS_16P
7  ...
8  ros_send_point_cloud_topic : / rslidar_points
```

Listing 4.2: Changes to config.yaml in the LiDAR SDK

Since the recorded point clouds of the RS-Helios-16P contain Not a Number (NaN) values and most libraries output error messages due to this and are aligned to Velodyne sensors, a conversion of the format is recommended. The package `rs to velodyne`³⁶ takes this task and removes NaN values at the same time. During some test runs, an error message often appeared when converting the formats (see Appendix A.9). When looking at the conversion code in the `rs to velodyne.cpp` file, the following error was found (see Listing 4.3):

³⁶https://github.com/HViktorTsoi/rs_to_velodyne

```

1 POINT_CLOUD_REGISTER_POINT_STRUCT ( RsPointXYZIRT ,
2   ( float , x, x)( float , y, y)( float , z, z)( float , intensity ,
3   intensity ) ( uint16_t , ring , ring )( double , timestamp , timestamp
4   ))

```

// Change the type from uint8_t , intensity , intensity to float

Listing 4.3: Changes to the rs_to_velodyne.cpp file

The intensity variable was previously defined as type `uint16_t`. However, the format of the LiDAR used is of type `float`. The reason for this is probably that the library was not further developed for some time and changes have since been made to the format by the manufacturer. Changing the variable type ensures that the error message is resolved. A specially created launch file simplifies the launch process so that the commands do not have to be entered individually each time.

```

1 <launch>
2   <node pkg = " rs_to_velodyne " name = " rs_to_velodyne " type = "
      rs_to_velodyne " args = " XYZIRT XYZIRT " output = " screen " >
3   </ node >
4   </ launch >

```

Listing 4.4: Launch file for the rs to velodyne node

The next step is to configure the `hdl_graph_slam` algorithm. This is followed by a brief explanation of how it works. Four nodes are relevant. Figure 4.2 shows the sequence of the algorithm and the relevant nodes.

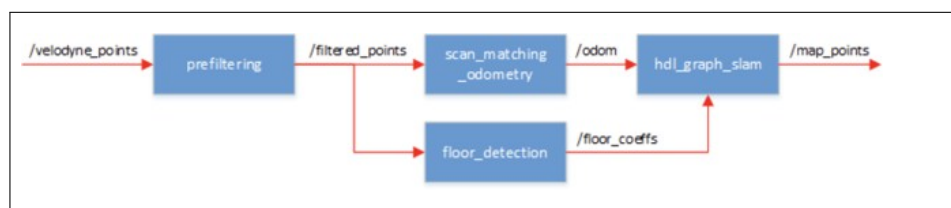


Figure 4.2: Diagram of the hdl graph slam nodes from [koi23]

The incoming point cloud under the topic `/velodyne_points` is first filtered. This is done in the `prefiltering` node. Here, two threshold values `distance near thresh` and `distance far thresh` are used to identify measurement points that are too close or too far away.

are removed can be filtered out. Furthermore, so-called outliers³⁷ can be removed using two methods. There is a radius-based method and a statistical method for this purpose. The former method takes the neighbors of a point within the radius specified in `radius radius`. If the value of the minimum number of neighbors is below the value `radius min neighbors`, the point is removed. In contrast, outliers can be determined with the statistical method using a standard deviation `statistical stddev`.

The scan matching described in 2.1.5 takes place in the scan matching odometry node. The sensor position is estimated by applying scan matching from successive measurements, from which the movement path of the robot is determined. For scan matching, the ICP, GICP, NDT and VGICP methods described in section 2.1.5 can be selected and the corresponding parameters configured. Since a flat surface is usually assumed as the floor in buildings, a further boundary is introduced using the floor detection node. This optimizes the graph so that the detected floor is on the same level for all detected scans. The level is calculated using Random Sample Consensus (RAN-SAC) [FB81].

The following settings were made for the use of the algorithm in the interior. In the launch file:

```

1  <launch>
2  ...
3  <arg name = " enable_floor_detection " default = " true
4  <arg " />
5  <arg name = " enable_gps " default = " false " />
6  <arg name = " enable_imu_acc " default = " false
7  ... name = " enable_imu_ori " default = "
8  <arg name = " points_topic " default = "/ velodyne_points
9  ... " />
10 <param name = " tilt_deg " value = " 0.0 " />
11 <param name = " sensor_height " value = " 0.5 " />
12 <param name = " height_clip_range " value = " 0.5
13 " />
14 ...
</ launch >

```

Listing 4.5: Excerpts from the launch file for the hdl graph slam algorithm

Functions relating to the consideration of GPS and IMU have been switched off.

³⁷dt. Outlier

as these are not available. The incoming message³⁸ of the point cloud of type `sensor_msgs/PointCloud2` is set to `velodyne points`. This corresponds to the type that the previously set `rs` to `velodyne node` publishes. It can also be assumed that the LiDAR is placed level on the robot. The `tilt_deg` is therefore set to 0° . The measured height of the LiDAR is $\sim 0.5\text{m}$. The last relevant parameter `height_clip_range` specifies a threshold value for detecting the floor. Only points in the range from `sensor height + height clip range` to `sensor height - height clip range` are taken for ground detection. The setup of the individual nodes is now complete. The recording and evaluation process is described below.

The master node is initialized and started with the help of `roscore`. Subsequently the `rslidar sdk` is started to transfer the point cloud. The next step is to convert it into `velodyne` format and publish the point cloud under the name `velodyne points`. The `roslaunch` record `/velodyne points` command records the data. For simplified operation, a shell script was created on the robot to start the commands automatically.

```
1 source mapping_catkin_ws / devel / setup . bash &
2 roscore &
3 roslaunch rslidar_sdk start . launch &
4 roslaunch rs_to_velodyne rstovelodyne . launch &
5 rosbag record / velodyne_points
```

Listing 4.6: Start and recording process

During both scenarios, the robot was controlled using the remote control. The `roslaunch` info command can be used to verify the recorded file. The date entries listed here are of minor importance. This is the last date set by the manufacturer for the NVIDIA Xavier NX on the robot.

Once it has been ensured that the file is error-free, the algorithm can be executed on the bag file. The mapping process can be easily visualized with the help of RViz. The framework offers two different launch files depending on whether an indoor or outdoor scenario is being considered. The scan matching parameters are changed depending on the scenario. These can be edited in the launch files.

³⁸German Message

	iisys.bag	outdoorhs.bag
duration	4:09s (249s)	9:45s (585s)
size	1.9 GB	2.8 GB
messages	2497	4660
types	sensor-msgs/PointCloud2	sensor-msgs/PointCloud2
	topics/velodyne	points/velodyne points

Table 4.3: Characteristics of the generated bag files

The generated maps are saved using the command `rosservice call /hdl_graph_slam/save map`

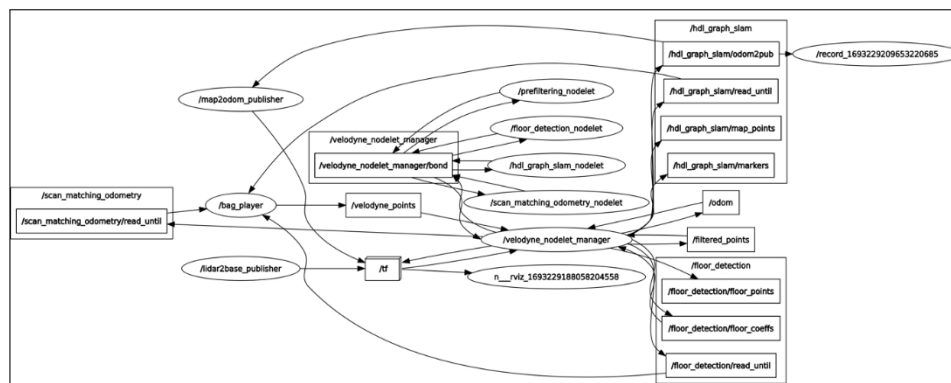


Figure 4.3: Nodes and topics of the rqt graph for the hdl graph slam algorithm

saved. Figures 5.2 and 5.4 show the 3D environments generated by the SLAM algorithm. The waypoints generated by the algorithm can be seen in the bottom right corner. Now that the implementation has been completed, the SLAM procedure is analyzed on the basis of the selected scenarios. Figure

4.3 shows the calculation graph using the `rqt graph` function of ROS. This shows all active nodes and topics during the SLAM process.

```

1  rosparam set use_sim_time true
2  roslaunch hdl_graph_slam hdl_graph_slam_501 . launch
3  roscd hdl_graph_slam / rviz
4  rviz - d hdl_graph_slam . rviz
5  rosbag play -- clock iisys . bag

```

Listing 4.7: Starting the hdl graph slam on the bag file

Videos of the mapping process are available for indoor use at <https://www.youtube>.

com/watch?v=10QQdSywcgI&t=7s and for the outdoor area https://www.youtube.com/watch?v=eKyQpy1LQ&ab_channel=Jonas.

5 Results analysis

The main content of this chapter is the preparation and interpretation of the selected scenarios for the SLAM algorithm. Finally, limitations and obstacles that arose during the test trials are discussed.

5.1 Evaluation of the scenarios

The FAST GICP and FAST VGICP algorithms were considered for the evaluation of the scan matching procedures. The FAST GICP algorithm is recommended by the developers of the hdl graph slam package and is also configured as the default setting. Since the literature on NDT OMP algorithms promises good results with the use of a multi-beam LiDAR, this approach is also considered [Car+21]. The configuration of the standard parameters showed good results. No changes were made in this respect.

Due to the absence of authenticity data, the evaluation is limited to visual impressions and the comparison of the created environment maps with each other. With the help of the evo³⁹ package would allow much more detailed evaluations to be carried out. By comparing with ground truth data, the deviations of the trajectories and statistical characteristics such as absolute pose error (APE) and relative pose error (RPE) could be determined.

³⁹to be found at: <https://github.com/MichaelGrupp/evo>

5.1.1 Consideration of indoor scenario

As already mentioned, the foyer of the Institute for Information Systems at Hof University of Applied Sciences (IISYS) was chosen as the test track for indoor rooms. It is assumed that the floor has a flat surface and the walls are at a 90° angle to the floor. The foyer has many window fronts, so that parts of the inner courtyard and lecture rooms are also covered by the LiDAR (see Figure 5.2). Measurements through window glass can lead to non-reproducible measurements or lower accuracy. This is because glass can either cause reflection or the laser beam is refracted in such a way that it changes its angle or direction [AAP08].

The loop closure procedure also worked, as can be seen in Figure 5.2. More detailed information on the loop closure method can be found in [KMM19]. Figure 5.1 shows the SLAM graph of the indoor test track as well as the generated 3D environment map and a photo of the foyer. The connections of the graph at some nodes indicate a successfully detected loop closure.

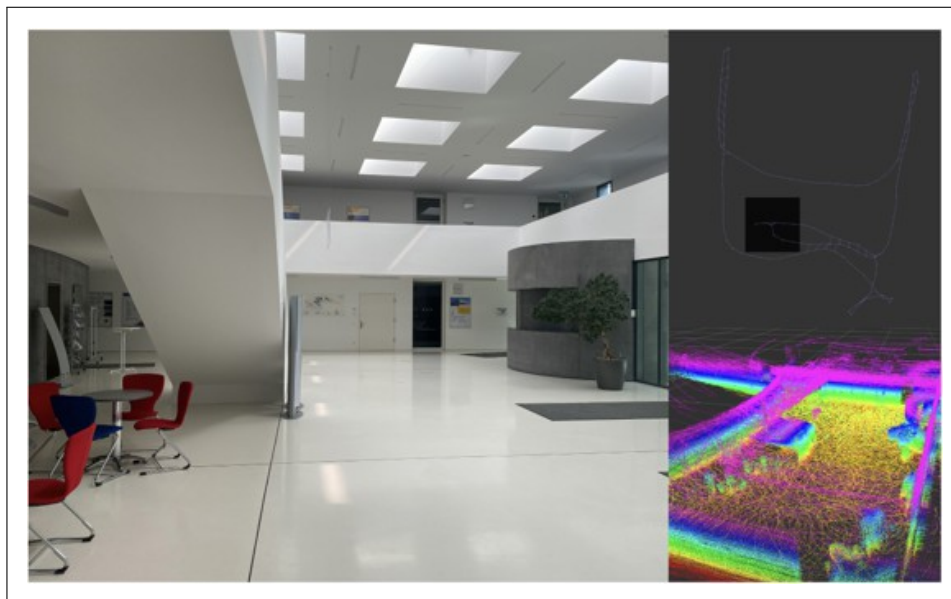


Figure 5.1: Photo with corresponding determined 3D point cloud environment of the Foyer and SLAM graph of the IISYS

The 3D map generated in the interior is quite successful. The best results were achieved using the FAST GICP and FAST VGICP algorithms. Figure 5.2 shows a satellite image together with different views of the environment map created by FAST GICP.

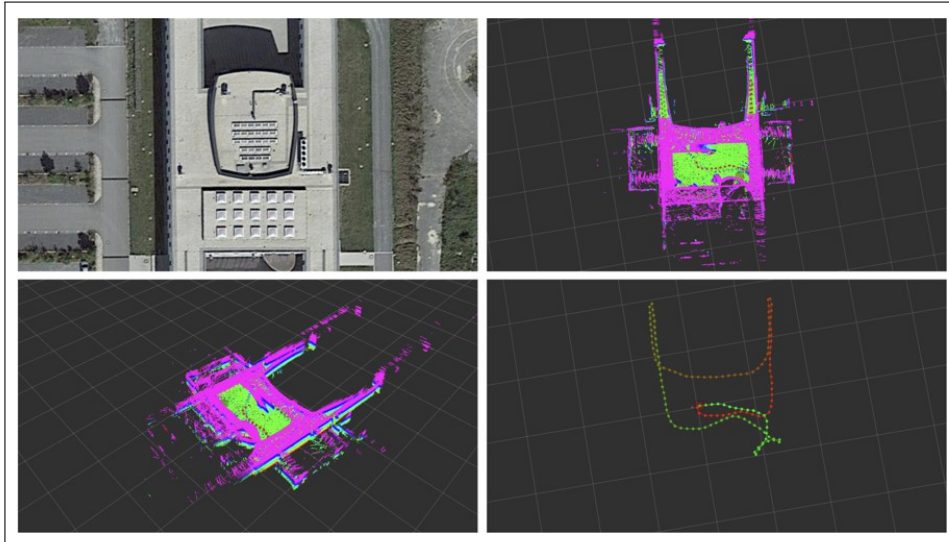


Figure 5.2: Different views of the mapped environment of the indoor scenario (FAST GICP)

While the previously mentioned scan-matching methods provided usable results, as the quality and the representation of the surroundings of the map corresponded to reality, the map generated by HDT OMP was largely unusable. The point cloud was shifted and rotated, which resulted in an incorrect generation of the environment map. Figure 5.3 shows the map created by HDT OMP.

The presence of ground detection also had an effect on the measurements and resulting maps. While no inclines or gradients occurred when ground detection was switched on, this was not the case when ground detection was switched off. In some situations, the measurements resulted in values as if the robot were running up a slope or incline even though the ground was flat.

In order to check the accuracy of the map created, it is also checked for a perpendicularity. For this purpose, the detected walls are compared with the real walls. If it is assumed that the building is rectangular, it can be noted that deviations occur in some places. The map created with hdl graph slam is slightly distorted. The use of other procedures or additional boundary conditions did not lead to significant improvements.

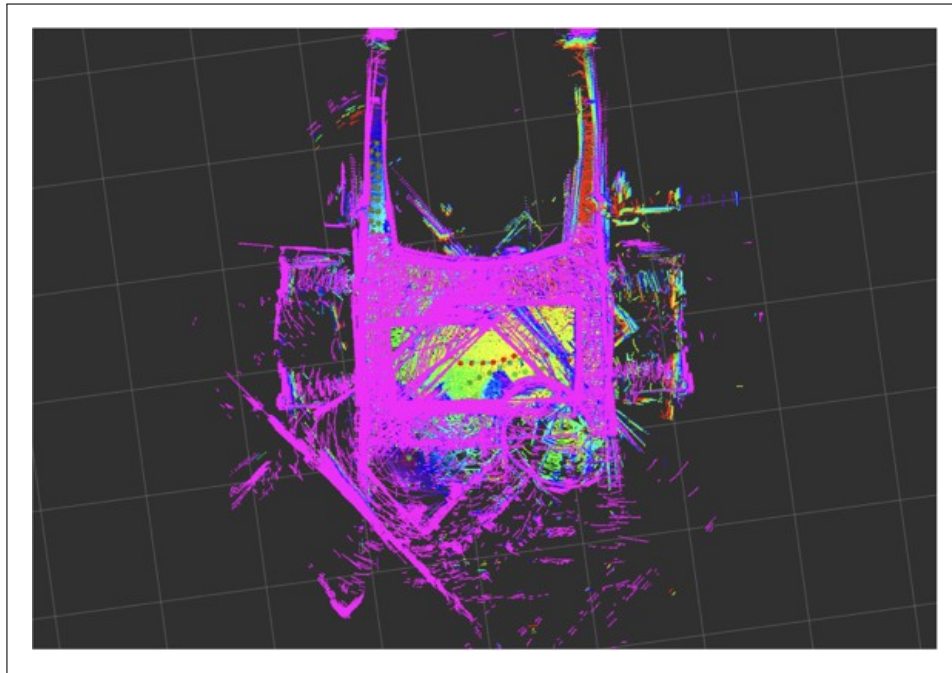


Figure 5.3: Faulty environment map generated by HDT OMP

5.1.2 Consideration of outdoor scenario

A distance of ~ 560 meters is considered for the outdoor area. This stretches from the main entrance of the Institute for Information Systems down a flight of stairs to the parking lot to the main building of Hof University of Applied Sciences. Figure 5.4 can be viewed for this purpose. It shows a satellite view with the route drawn in as well as various views of the surrounding map created using FAST GICP. At the bottom left of the image is the trajectory determined by the algorithm. During the outdoor measurements, the weather was sunny, so there were no major disturbances. However, it is worth noting that it was particularly warm ($\sim 30^\circ \text{C}$), which may allow conclusions about the short battery life (see section 5.2).

The best results were also achieved outdoors with FAST GICP and FAST VGICP. On the other hand, distortions and rotations of the map occurred when using HDT OMP, as was the case indoors. Tests were also carried out in the outdoor area with regard to floor detection. Figure 5.5 shows the slope immediately to the left of the Institute for Information Systems with an adjoining parking lot. Above it on the left is the appropriate section of the generated map of the surroundings without activated floor detection. At the top right is the same section

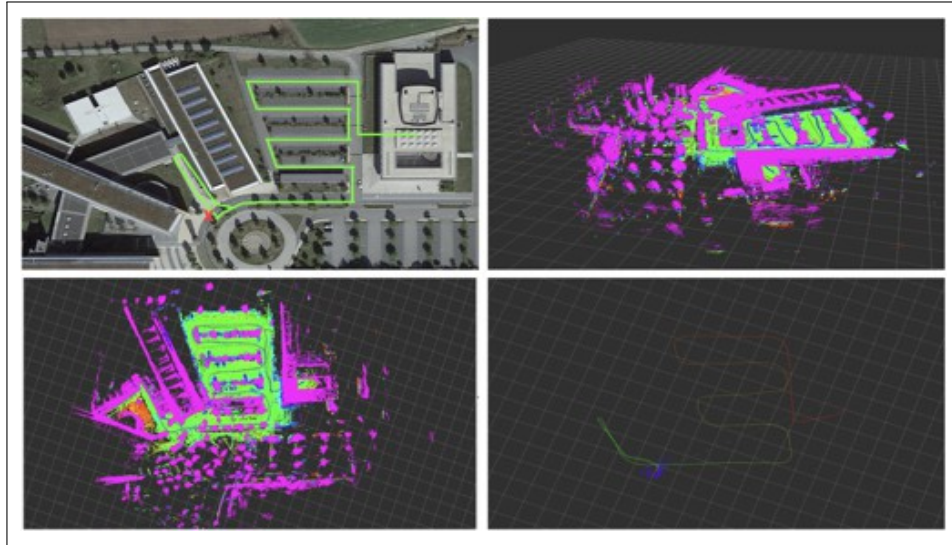


Figure 5.4: Different views of the mapped environment of the outdoor scenario (FAST-GICP)

section with activated floor detection. It is clear that deactivating the floor detection node ensures quite solid detection of the stairs and slope. In contrast, an incorrect map was created when floor detection was activated.

However, the activated ground detection causes problems in other places. Figure 5.6 shows a section of the map of the outdoor area with activated ground detection. A figurative inclination of the surroundings in *the Z* direction can be seen.

It is assumed here that while the robot was descending the stairs and thus the LiDAR was also briefly inclined, an incorrect reference measurement of the environment was made. This led to the fact that after the robot stood on a straight plane again and started the measurement, an inclination was falsely generated. The use of other scan-matching methods did not lead to any improvements here either.

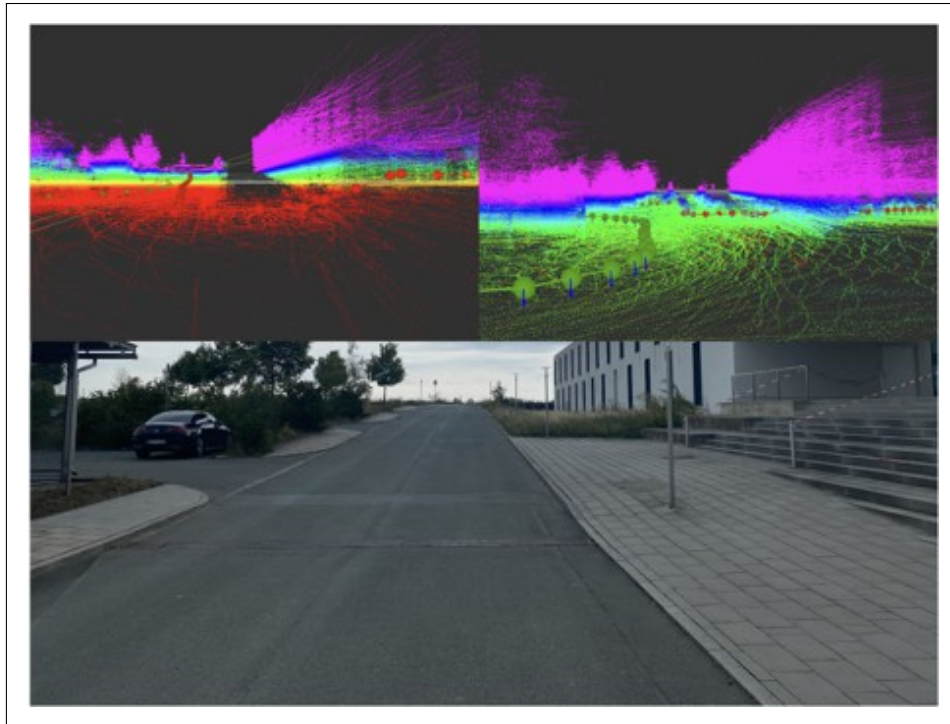


Figure 5.5: Comparison of the 3D map for slopes and stairs without floor detection (left) and with (right)

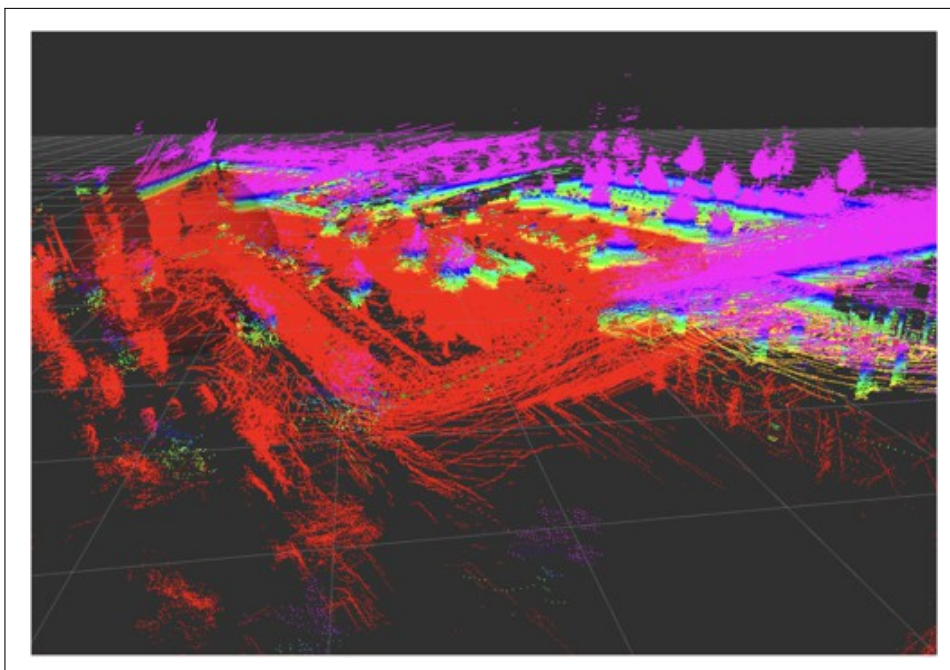


Figure 5.6: Illustration of the faulty 3D map due to tilted reference measurements

5.1.3 Main findings

The experiments carried out showed that the use of mobile four-legged robots is feasible with the aid of modern LiDAR and SLAM technology. Some important findings need to be considered separately.

The system demonstrates high performance and reliability for indoor environments, provided that the floor level is not as a boundary. Otherwise, the results deviate from the actual environment. The FAST GICP algorithm with activated floor boundary turned out to be a particularly promising approach in indoor environments. During the tests, this configuration delivered the best results. Good performance was also achieved using FAST VGICP. Only the NDT OMP-method turned out to be mostly unusable in terms of the final state of the generated 3D map.

However, the inclusion of floor detection can also be relevant for outdoor scenarios, for example when navigating in urban environments or concreted pavements where level ground conditions are to be assumed. In larger scenarios, however, in which navigation takes place over several levels, this limitation must be deactivated. FAST GICP and FAST VGICP also proved to be suitable in this context and delivered usable results. It should be noted, however, that using only the odometry data resulting from SLAM in external areas usually provides inadequate results. The inclusion of additional data such as from cameras, IMU or GPS data is therefore essential for the success of such scenarios.

5.2 Limitations of the system

The battery runtime proved to be a problem during the outdoor scenario recording. The battery was fully charged beforehand. During the recording, the stair mode of the robot was activated in some places to overcome stairs. Furthermore, the robot ran at a significantly higher average speed, as shown in Table 4.1. The mapping had to be terminated in the outdoor area after ~ 15 minutes, as the robot collapsed due to the low battery level.

Another problem was connectivity. As the NVIDIA Xavier NX does not provide a direct Ethernet connection and the RJ-45 port on the back is not available.

of the robot was connected to the LiDAR, the only option was to start the recording process via the Raspberry Pi, but this led to extremely high ping times and was therefore inconvenient to use.

The fact that no ground truth data is available is a hindrance to the evaluation.⁴⁰ data is available. The Go1 does not offer any direct options for accessing data from the IMU. However, this would be useful as an additional framework for the hdl_graph_slam algorithm to improve accuracy. It can be assumed that the more constraints (IMU, GPS, floor detection) are present, the better the output of the calculated environment map will be. GPS solutions or motion-capture recordings for recording real data would be advisable in this respect.

For future real-time applications, a constant connection to the Hof University network must be available. During the tests, it turned out that the connection was unstable or that sometimes only very weak signals with high latency times were received. In the long term, a solution must be found for stable communication between the robot and the Hof University of Applied Sciences network.

⁴⁰German Authenticity data

6 Conclusion

Finally, the most important findings of this work are summarized. This is followed by a consideration of future research opportunities with regard to the use of four-legged robotic systems.

6.1 Summary

The aim of this work was to develop an intelligent lidar-based navigation system for a four-legged robot. The focus was on investigating the implementation of a SLAM procedure on the Unitree Go1. The application area is **limited** to the environment of the Hof University of Applied Sciences and the Institute for Information Systems at Hof University of Applied Sciences. Independent development of a SLAM algorithm is not planned. Instead, a number of well-known libraries were used.

The structure is divided into six chapters. At the beginning, the basics in the area of Robotics, AI and SLAM were presented. A brief historical outline introduced the topic of robotics. This was followed by a look at common robot components such as actuators and end effectors as well as ways to program robots. ROS was examined in more detail as a widely used tool. ROS facilitates the development of complex robot systems through a kind of divide and conquer principle, in which individual nodes are responsible for an explicit task. A combination of these nodes enables the development of complex overall systems.

The terms and differences between AI, ML and DL were then explained. ML and DL are subcategories of AI. Especially in the area of ML, there are approaches that are relevant for the development of navigation systems. The use of LiDAR sensors and the combination of intelligent ML approaches ensures the

Ensuring robust localization and navigation systems. Concepts such as Graph- SLAM, in which the robot's trajectory and relevant keyframes can be represented using a graph, support this project. As a scan-matching methodology, the ICP algorithm was examined in more detail as the basis for many other methods based on it. A consideration of the challenges and related work on four-legged robots and their possible applications rounded off the chapter.

A closer look at the components used in the Go1 and LiDAR sensor followed. The main components of the Go1 are the four limbs, the body and the head. Three motor-joint systems at each of four positions on the body provide a total of 12 DOF. The hardware installed in the robot can be accessed with the help of connections attached to the back. This includes a Raspberry Pi, two NVIDIA Jetson Nanos and an NVIDIA Xavier NX. The robot is also equipped with a range of other sensors, such as ultrasonic sensors and stereo camera systems. The sensor data of the RS-Helios-16P can be visualized with the help of RSView. The LiDAR has different operating settings that can be changed using a web interface. A simulation environment for the Go1 is possible with the help of Gazebo, RViz and the unitree ros SDK.

In chapter 4, the mounting of the LiDAR on the robot and the setup as well as the Configuration of the hdl graph slam framework used. The algorithm offers a number of different scan matching methods (ICP, NDT, VGICP). A floor detection node, which improves the mapping in buildings by assuming a planar plane as floor, can be activated and deactivated as an additional boundary condition. Finally, the tests were analyzed. This was done by considering different setting options of the hdl graph slam package. The SLAM graph was analyzed taking into account various scan matching methods. The FAST GICP approach proved to be particularly suitable, as it showed the best results in indoor areas in combination with the floor detection condition. In outdoor areas, stairs and slopes in particular had a negative impact on the measurement process due to limited sensor technology.

Using SLAM with the help of four-legged robots is very promising and offers LiDAR offers robust options for surrounding and localizing the robot without the specific use of IMU or odometry data. The disadvantage is that the battery life of a LiDAR runs out quickly. Here, long-term

other solutions can be considered.

6.2 Future research approaches

The possibilities for the further development of Go1 are almost unlimited. Initially, however, the focus should be on fully exploring the existing possibilities of Go1. The focus should be on expanding **connectivity**, investigating the SDKs provided and approaches to further integrating the robot into the university environment. This also includes a consideration of the error messages and complications mentioned in this thesis (UnitreeCameraSDK, reading ultrasonic sensors using LCM). They should be resolved in order to enable smooth integration in the future. In order to implement real-time applications with monitoring components on the overall system, for example, stable **connectivity** must be ensured. The tests showed that the connection to the robot only worked with great difficulty. In addition, the connection to the university network was unstable and delivered high latency times. Ensuring a stable connection of the respective components is therefore essential for further developments and should be a high priority for future extensions of the system.

In the long term, the Go1 should be able to navigate autonomously on the university campus and offer interaction possibilities. Possible work could be dedicated to implementing an interface for the robot controller and the AI software. This can be implemented with the help of ROS, for example. An in-depth study of the robot's movement options based on the Unitree SDKs is advisable for this.

In order to enable the evaluation of future SLAM and AI algorithms on the robot, the Simple data should be collected for the comparison. This includes the creation of ground truth data for future analyses of further tests. This could be implemented using motion capture methods to determine the authenticity of the robot's path data. Reviewing and modifying the measurement process and parameters could also provide new insights into the use of SLAM methods and four-legged robots. A comparison with other mapping and SLAM frameworks such as LIO-LOAM would be exciting.

In the same respect, solutions for the recognition of persons must be developed.

the. Collisions can be avoided using a wide variety of sensors. As the AI software provided did not deliver sufficient results after a few attempts, alternatives had to be researched. With the help of OpenCV or YOLO algorithms, it is possible to quickly develop efficient systems. In the long term, however, the various sensors should be merged. The interaction of LiDAR, ultrasonic sensors and cameras **increases** the robustness and resilience of autonomous robots.

The Go1 can be **equipped** with almost any number of extensions. For example, an arm can be placed on the back as an end effector. This creates new possibilities in terms of interaction with its environment.

It would therefore be conceivable for the robot to be able to open doors independently or to operate locking devices. It is also conceivable that it could be used as part of the university's **building** management for control purposes.

The use of AI applications **enables** further conceivable scenarios. Cloud technologies open up the **possibility** of reducing the computing load of the hardware on the robot. Computing-intensive ML and DL processes in particular ensure high utilization of the robot systems. Moving these processes to the cloud is therefore a logical step. Especially since the use of multiple sensor fusion (camera systems, extended sensor functionality) should be aimed for in the later stages. This will push the limited computing power of the robot hardware to its limits. This migration to cloud-based services brings its own challenges and many exciting approaches to the long-term use of the robot on the university campus.

Since a second Go1 robot is available, cooperation scenarios can be developed in the near future. By having several robots working in coordination, tasks can be accomplished together. These could include exploring large areas, efficiently collecting data or collectively solving challenges such as transporting heavy loads.

Finally, ethical and legal issues should also be considered. Especially with regard to privacy, data protection and security in the context of human-robot interaction, further research should be considered to ensure responsible use.




```

pi@raspberrypi: ~/ros_catkin_ws/src
pi@raspberrypi:~/ros_catkin_ws/src $ ls -la
total 220
drwxr-xr-x 52 pi pi 4096 Jun 22 04:32 .
drwxr-xr-x 6 pi pi 4096 Jul 30 2021 ..
drwxr-xr-x 8 pi pi 4096 Aug 7 2019 actionlib
drwxr-xr-x 5 pi pi 4096 Nov 6 2018 actionlib_msgs
drwxr-xr-x 5 pi pi 4096 Jan 8 2020 angles
drwxr-xr-x 3 pi pi 4096 Oct 2 2018 bond
drwxr-xr-x 4 pi pi 4096 Jul 19 2021 bondcpp
drwxr-xr-x 7 pi pi 4096 Jul 11 2021 catkin
drwxr-xr-x 8 pi pi 4096 Jul 11 2021 class_loader
drwxr-xr-x 4 pi pi 4096 Jul 11 2021 cmake_modules
drwxr-xr-x 12 pi pi 4096 Jul 30 2021 david
drwxr-xr-x 11 pi pi 4096 Mar 19 2020 dynamic_reconfigure
drwxr-xr-x 5 pi pi 4096 Jul 11 2021 gencpp
drwxr-xr-x 5 pi pi 4096 Jul 11 2021 geneus
drwxr-xr-x 5 pi pi 4096 Jul 11 2021 genlisp
drwxr-xr-x 7 pi pi 4096 Jul 11 2021 genmsg
drwxr-xr-x 5 pi pi 4096 Jul 11 2021 gennodejs
drwxr-xr-x 7 pi pi 4096 Jul 11 2021 genpy
drwxr-xr-x 3 pi pi 4096 Nov 6 2018 geometry_msgs
drwxr-xr-x 4 pi pi 4096 Mar 10 2020 kdl_conversions
drwxr-xr-x 2 pi pi 4096 Jul 11 2021 message_generation
drwxr-xr-x 2 pi pi 4096 Jul 11 2021 message_runtime
drwxr-xr-x 5 pi pi 4096 Nov 6 2018 nav_msgs
drwxr-xr-x 6 pi pi 4096 Apr 27 2018 nodelet
drwxr-xr-x 4 pi pi 4096 Apr 27 2018 nodelet_topic_tools
drwxr-xr-x 9 pi pi 4096 Mar 21 2018 orocos_kdl
drwxr-xr-x 4 pi pi 4096 Apr 2 2020 pcl_conversions
drwxr-xr-x 3 pi pi 4096 Mar 21 2018 pcl_msgs
drwxr-xr-x 9 pi pi 4096 Apr 2 2020 pcl_ros
drwxr-xr-x 2 pi pi 4096 Apr 2 2020 perception_pcl
drwxr-xr-x 5 pi pi 4096 Jul 11 2021 pluginlib
drwxr-xr-x 13 pi pi 4096 Jul 11 2021 ros
drwxr-xr-x 22 pi pi 4096 Jul 11 2021 ros_comm
drwxr-xr-x 4 pi pi 4096 Jul 11 2021 ros_comm_msgs
drwxr-xr-x 10 pi pi 4096 Jul 11 2021 rosconsole
drwxr-xr-x 6 pi pi 4096 Jul 11 2021 roscpp_core
drwxr-xr-x 4 pi pi 4096 Jul 11 2021 ros_environment
drwxr-xr-x 10 pi pi 4096 Jun 22 04:32 ros-foxxglove-bridge
-rw-r--r-- 1 pi pi 11398 Jul 11 2021 .rosinstall
drwxr-xr-x 6 pi pi 4096 Mar 21 2018 roslint
drwxr-xr-x 12 pi pi 4096 Jul 11 2021 roslisp
drwxr-xr-x 5 pi pi 4096 Jul 11 2021 rospack
drwxr-xr-x 8 pi pi 4096 Nov 6 2018 sensor_msgs
drwxr-xr-x 4 pi pi 4096 Oct 2 2018 smclib
drwxr-xr-x 4 pi pi 4096 Jul 11 2021 std_msgs
drwxr-xr-x 9 pi pi 4096 Mar 10 2020 tf
drwxr-xr-x 5 pi pi 4096 Nov 16 2018 tf2
drwxr-xr-x 4 pi pi 4096 Nov 16 2018 tf2_eigen
drwxr-xr-x 6 pi pi 4096 Nov 16 2018 tf2_kdl
drwxr-xr-x 6 pi pi 4096 Nov 16 2018 tf2_msgs
drwxr-xr-x 3 pi pi 4096 Nov 16 2018 tf2_py
drwxr-xr-x 6 pi pi 4096 Nov 16 2018 tf2_ros
drwxr-xr-x 5 pi pi 4096 Mar 10 2020 tf_conversions
pi@raspberrypi:~/ros_catkin_ws/src $

```

Figure A.2: Output of the installed ROS packages on the Raspberry Pi



Figure A.3: Excerpts from the web application and display of the person recognition software

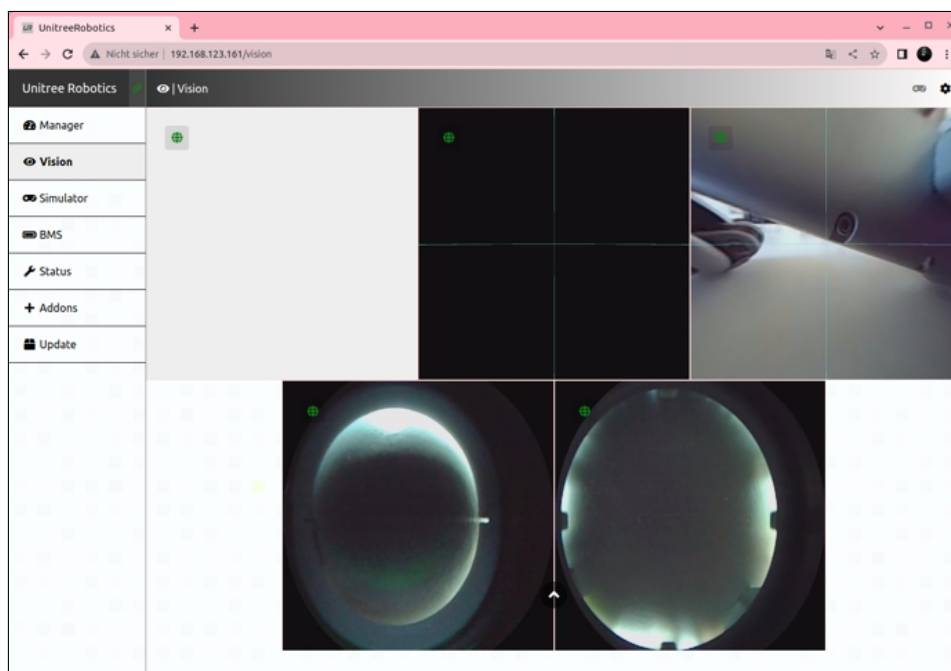


Figure A.4: Excerpts from the web application and display of the various cameras

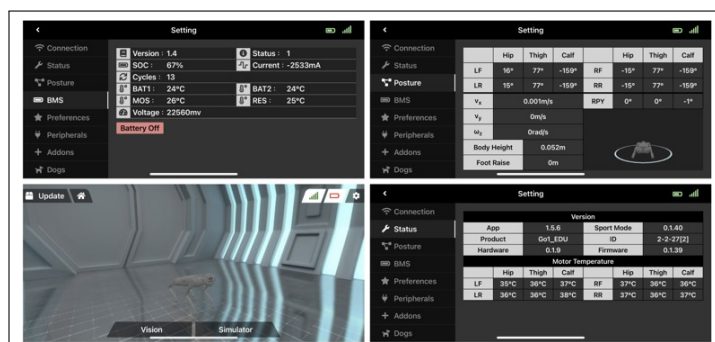
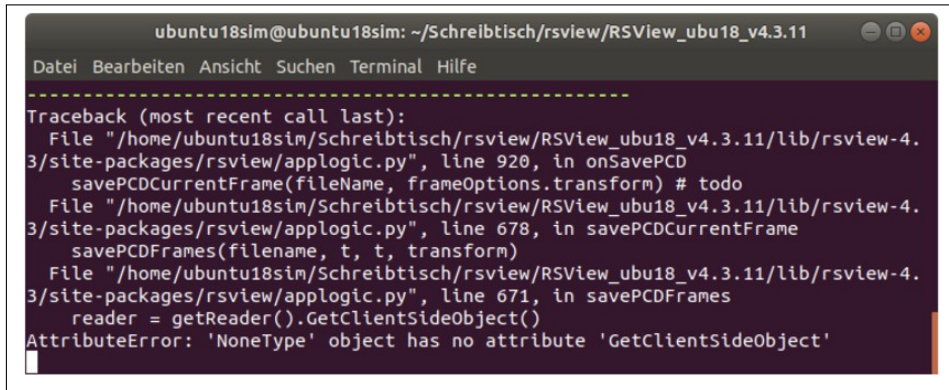


Figure A.5: Excerpts from the iOS app

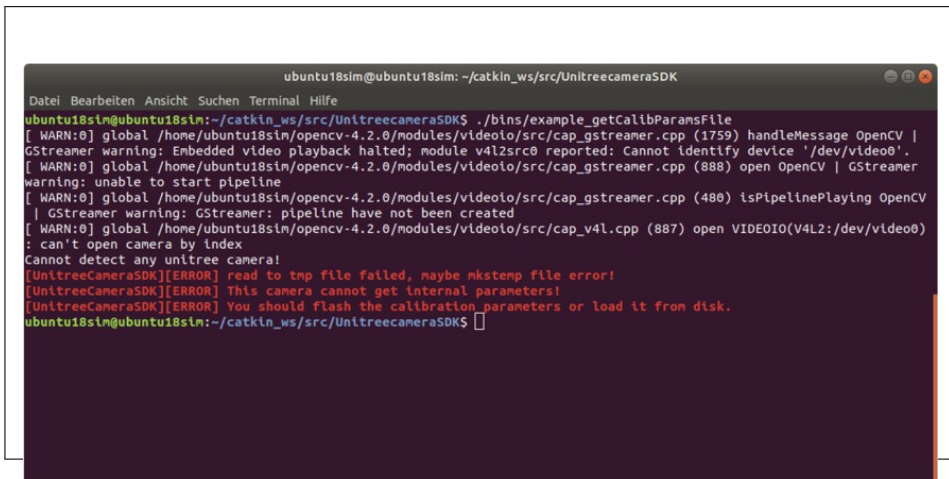


```

ubuntu18sim@ubuntu18sim: ~/Schreibtisch/rsview/RSView_ubu18_v4.3.11
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
-----
Traceback (most recent call last):
  File "/home/ubuntu18sim/Schreibtisch/rsview/RSView_ubu18_v4.3.11/lib/rsview-4.3/site-packages/rsview/applogic.py", line 920, in onSavePCD
    savePCDCurrentFrame(fileName, frameOptions.transform) # todo
  File "/home/ubuntu18sim/Schreibtisch/rsview/RSView_ubu18_v4.3.11/lib/rsview-4.3/site-packages/rsview/applogic.py", line 678, in savePCDCurrentFrame
    savePCDFrames(filename, t, t, transform)
  File "/home/ubuntu18sim/Schreibtisch/rsview/RSView_ubu18_v4.3.11/lib/rsview-4.3/site-packages/rsview/applogic.py", line 671, in savePCDFrames
    reader = getReader().GetClientSideObject()
AttributeError: 'NoneType' object has no attribute 'GetClientSideObject'

```

Figure A.6: Error message output when trying to save the sensor data.

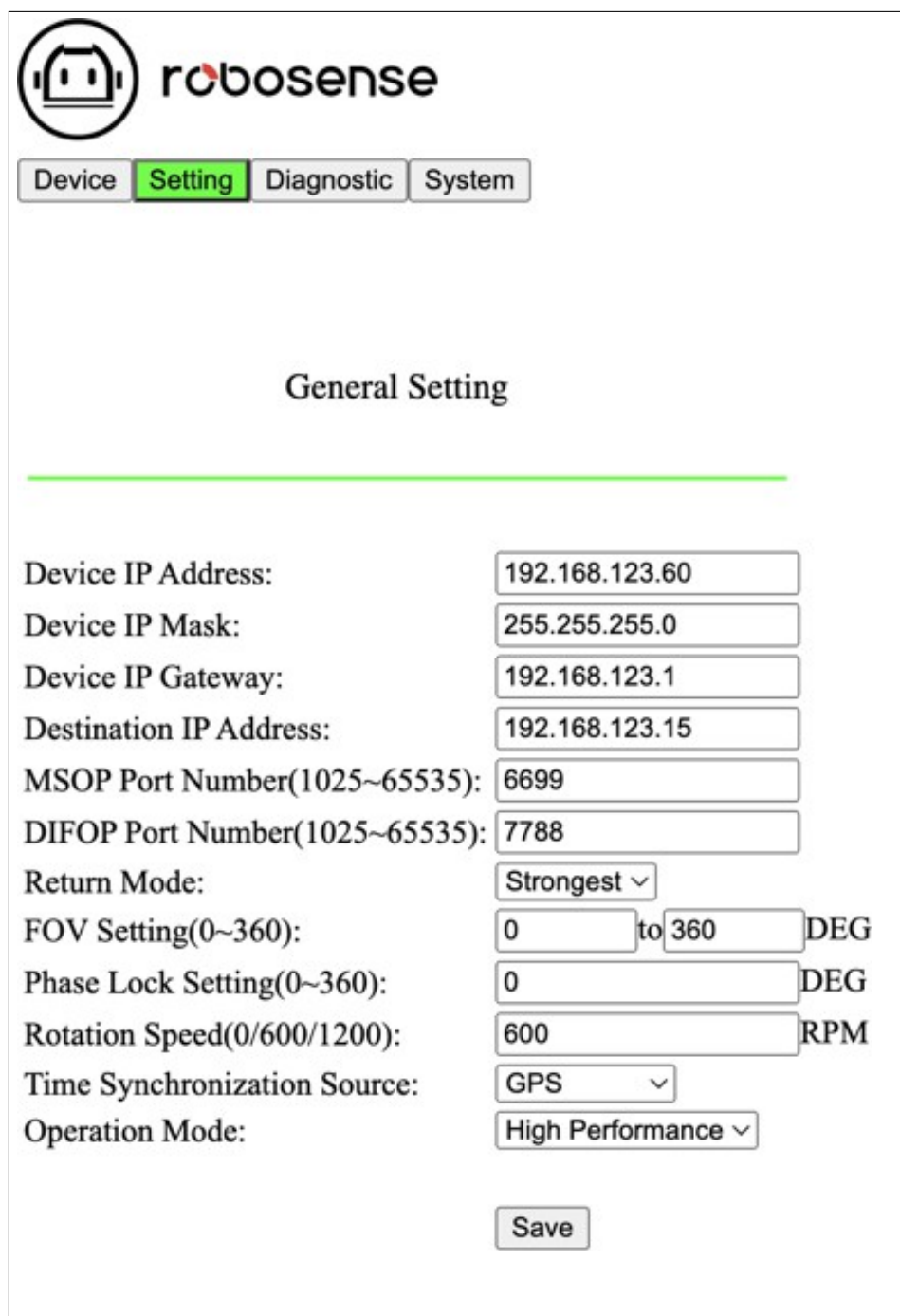


```

ubuntu18sim@ubuntu18sim: ~/catkin_ws/src/UnitreecameraSDK
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
ubuntu18sim@ubuntu18sim:~/catkin_ws/src/UnitreecameraSDK$ ./bins/example_getCalibParamsFile
[ WARN:0] global /home/ubuntu18sim/opencv-4.2.0/modules/videoio/src/cap_gstreamer.cpp (1759) handleMessage OpenCV |
GStreamer warning: Embedded video playback halted; module v4l2src0 reported: Cannot identify device '/dev/video0'.
[ WARN:0] global /home/ubuntu18sim/opencv-4.2.0/modules/videoio/src/cap_gstreamer.cpp (888) open OpenCV | GStreamer
warning: unable to start pipeline
[ WARN:0] global /home/ubuntu18sim/opencv-4.2.0/modules/videoio/src/cap_gstreamer.cpp (480) isPipelinePlaying OpenCV
| GStreamer warning: GStreamer: pipeline have not been created
[ WARN:0] global /home/ubuntu18sim/opencv-4.2.0/modules/videoio/src/cap_v4l.cpp (887) open VIDE0IO(V4L2:/dev/video0)
: can't open camera by index
Cannot detect any unitree camera!
[UnitreeCameraSDK][ERROR] read to tmp file failed, maybe mkstemp file error!
[UnitreeCameraSDK][ERROR] This camera cannot get internal parameters!
[UnitreeCameraSDK][ERROR] You should flash the calibration parameters or load it from disk.
ubuntu18sim@ubuntu18sim:~/catkin_ws/src/UnitreecameraSDK$

```

Figure A.7: Output of the error message when using the CameraSDK



The screenshot shows the Robosense configuration web interface. At the top, there is a logo and four tabs: 'Device', 'Setting' (which is highlighted in green), 'Diagnostic', and 'System'. Below the tabs, the title 'General Setting' is centered. A horizontal green line separates the title from the configuration fields. The fields are arranged in two columns. The left column contains labels for various settings, and the right column contains input boxes or dropdown menus for those settings. The settings include IP addresses, port numbers, return mode, FOV, phase lock, rotation speed, time synchronization source, and operation mode. A 'Save' button is located at the bottom right of the configuration area.

Device IP Address:	<input type="text" value="192.168.123.60"/>
Device IP Mask:	<input type="text" value="255.255.255.0"/>
Device IP Gateway:	<input type="text" value="192.168.123.1"/>
Destination IP Address:	<input type="text" value="192.168.123.15"/>
MSOP Port Number(1025~65535):	<input type="text" value="6699"/>
DIFOP Port Number(1025~65535):	<input type="text" value="7788"/>
Return Mode:	<input type="button" value="Strongest"/> ▾
FOV Setting(0~360):	<input type="text" value="0"/> to <input type="text" value="360"/> DEG
Phase Lock Setting(0~360):	<input type="text" value="0"/> DEG
Rotation Speed(0/600/1200):	<input type="text" value="600"/> RPM
Time Synchronization Source:	<input type="button" value="GPS"/> ▾
Operation Mode:	<input type="button" value="High Performance"/> ▾
<input type="button" value="Save"/>	

Figure A.8: Screenshot from the configuration settings of the LiDAR

```
unitree@nx:~/mapping_catkin_ws$ roslaunch rs_to_velodyne rs_to_velodyne XYZIRT XYZIR
[ INFO] [1639976295.399523272]: Listening to /rslidar_points .....
Failed to find match for field 'intensity'.
```

Figure A.9: Error message output during the conversion of LiDAR formats

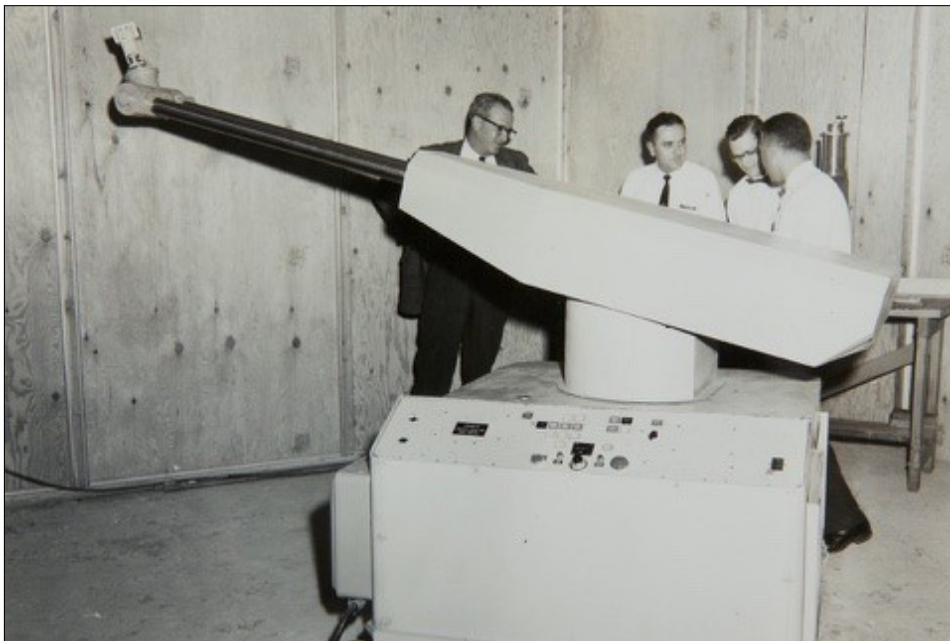


Figure A.10: Illustration of the Unimate robot arm from [Rob18]

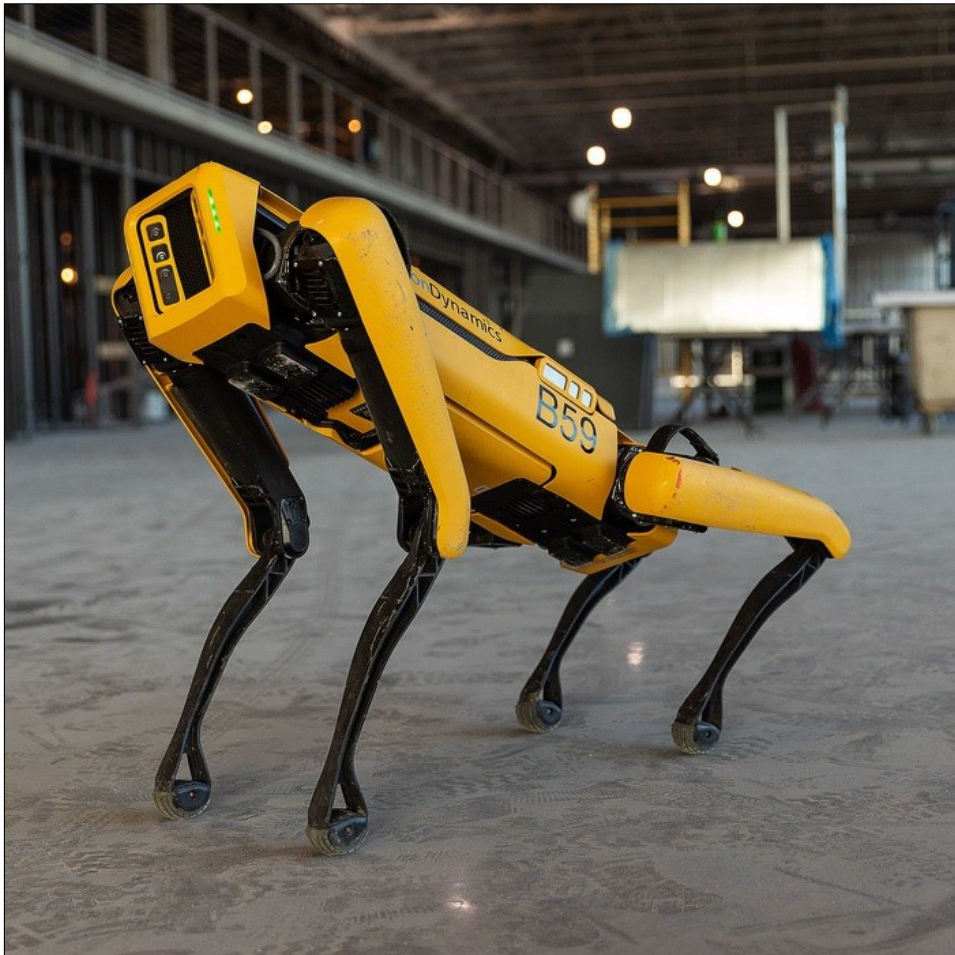


Figure A.11: Illustration of the spot robot from [Ver20]

B Appendix: Code

```

1 # include " UnitreeCamera . hpp " // Assuming this is the header file for
   the UnitreeCamera SDK
2 # include <opencv 2 / opencv . hpp >
3
4 int main ( int argc , char * argv []) {
5     // Initialize a UnitreeCamera object using a configuration file named "
   trans_rect_config. yaml"
6     UnitreeCamera cam ( " trans_rect_config . yaml " );
7
8     // Check if the camera is opened successfully
9     if ( ! cam . is Opened () )
10         exit ( EXIT_FAILURE
11             );
12
13     // Start capturing rectified stereo frames with image H.264 encoding
   disabled and share memory sharing enabled cam . startCapture (
14     true , false );
15
16     usleep ( 500000 ) ; // Sleep for 500 milliseconds
17
18     // Main loop for capturing and potentially displaying rectified stereo
   frames
19     while ( cam . is Opened () ) {
20         cv :: Mat left , right , feim ;
21
22         // Attempt to get rectified stereo frames
23         if ( ! cam . get Rect Stereo Frame ( left , right ) ) {
24             // If getting the stereo frames fails , wait for a short time and
   continue
25             usleep ( 1000 ) ;
26             continue ;
27         }
28
29         // Uncomment the following line to display the frames

```



```

29     // cv:: imshow ( " UnitreeCamera - RectifiedStereo ", left);
30
31     // Wait for a short period (10 milliseconds) and check for user input
32     char key = cv :: wait Key (10) ;
33
34     // If the ESC key ( ASCII code 27) is pressed , exit the loop
35     if ( key == 27)
36         break ;
37
38     // Stop camera capturing
39     cam . stopCapture ();
40
41     return 0;
42
43 }

```

Listing B.1: putImageTrans.cpp with comments

```

1  # include <opencv 2 / opencv . hpp >
2  # include <iostream >
3
4  int main ( int argc , char ** argv )
5  {
6      std :: string Ip Last Segment = " 13 "; // Last segment of the
7      camera 's IP address - 13 for Head
8      int cam = 1; // Camera view index , default is 1
9
10     // Check if a camera view index is provided as a command - line argument
11     if ( argc >= 2)
12         cam = std :: atoi ( argv [1]) ;
13
14     // GStreamer pipeline string components
15     std :: string udpstrPrevData " udpsrc address =192 .168 .123
16     = Ip Last Segment + " port =" ; . " +
17     std :: array <int , 5 > udpPORT std :: array <int , 5 > {9201 , 9202 ,
18     = 9204 , 9205 } ; 9203 ,
19
20     std :: string
21     udpstrBehindData = " ! application / x- rtp , media =
22     encoding = name = H264 ! rtph264depay ! h264 parse ! avdec_h
23     264 !
24     videoconvert ! appsink " ;

```

```

18 // Construct the complete GStreamer pipeline string
19 std :: string udp Send Integrated Pipe = udpstrPrevData + std ::
to_string ( udpPORT [ cam - 1]) + udpstrBehindData ;
20
21 std :: cout << " udp Send Integrated Pipe : " <<
udpSendIntegratedPipe << std :: endl ;
22
23 // Open the video capture using the constructed GStreamer pipeline
24 cv :: VideoCapture cap ( udpSendIntegratedPipe );
25
26 // Check if the video capture was successfully opened
27 if (! cap . is Opened
28     ()) return 0;
29
30 cv :: Mat frame ;
31
32 // Continuous loop to capture and display video frames
33 while (1)
34 {
35     cap >> frame ;
36     if ( frame . empty
37         ()) break ;
38
39     // Display the captured frame
40     imshow (" video ", frame );
41
42     // Wait for a short period (20 milliseconds)
43     cv :: wait Key (20) ;
44 }
45
46 // Release the video capture resources
47 cap . release ();
48
49 return 0;
50 }

```

Listing B.2: getimagetrans.cpp with comments

```

1 # include " unitree_legged_sdk / unitree_legged_sdk
2 . h" # include <math . h>
3 # include <iostream>
4 # include <unistd .
h>

```

```

5 #include <string . h>
6
7 using namespace UNITREE_LEGGED_SDK ;
8
9 class Custom
10 {
11 public :
12     // Custom( uint8_t level): safe( LeggedType :: A1 ), u d p ( level){
13     Custom ( uint8_t level ) : safe ( LeggedType :: A1 ), udp (8090
14     , " 192.168.123.11 " , 8082 , sizeof ( High Cmd ) , sizeof (
15     High State )){
16     // Custom( uint8_t level): safe( LeggedType :: A1 ), udp (8090 ,
17     "192.168.123.161" , 8082 , sizeof( HighCmd ), sizeof(
18     HighState)){
19     udp . Init Cmd Data ( cmd );
20     udp . Set DisconnectTime ( dt , 1);
21     // udp. SetDisconnectTime (0 , 0);
22     }
23     void UDPRecv ();
24     void UDPSend ();
25     void RobotControl ();
26
27     Safety safe ;
28     UDP udp ;
29     High Cmd cmd = {0};
30     High State state = {0};
31     int motiontime = 0;
32     float dt = 0 . 002 ;    // 0.001~ 0.01
33 };
34
35 void Custom :: UDPRecv ()
36 {
37     udp . Recv ();
38 }
39
40 void Custom :: UDPSend ()
41 {
42     udp . Send ();
43 }
44
45 void Custom :: RobotControl ()
46 {
47     motiontime += 2;

```

```

45     udp . GetRecv ( state );
46     cmd . mode = 0;    //0: idle , default stand           1: forced stand
                        2: walk continuously
47     cmd . gaitType = 0;
48     cmd . speedLevel = 0;
49     cmd . footRaise Height = 0;
50     cmd . body Height = 0;
51     cmd . euler [0] = 0;
52     cmd . euler [1] = 0;
53     cmd . euler [2] = 0;
54     cmd . velocity [0] = 0.0 f;
55     cmd . velocity [1] = 0.0 f;
56     cmd . yaw Speed = 0.0 f;
57
58 printf ( " rangeObstacle = [ Head : % f, Left : % f, Right : % f, % f]\ n",
59         state . range Obstacle [0] ,
60         state . range Obstacle [1] ,
61         state . range Obstacle [2] ,
62         state . range Obstacle [3]) ;
63
64     udp . SetSend ( cmd );
65 }
66
67 int main ( void )
68 {
69     std :: cout << " Communication level is set to HIGH - level ." << std
70     :: endl
71         << " WARNING : Make sure the robot is standing on the
72         ground ." << std :: endl
73         << " Press Enter to continue ... " << std :: endl ;
74     std :: cin . ignore ();
75
76     Custom custom ( HIGHLEVEL );
77     Init Environment ();
78     LoopFunc loop_control ( " control_loop ", custom . dt , boost :: bind
79     (& Custom :: RobotControl , & custom ));
80     LoopFunc loop_udpSend ( " udp_send ",    custom . dt , 3 , boost ::
81     bind (& Custom :: UDPSend                custom ));
82     LoopFunc loop_udpRecv ( " udp_recv ",    custom . dt , 3 , boost ::
83     bind (& Custom :: UDPRecv                custom ));
84
85     loop_udpSend . start ();

```

```

81     loop_udpRecv . start
82     (); loop_control .
83     start ();
84
85     while (1) {
86         sleep (10) ;
87     };
88
89     return 0;

```

Listing B.3: Modified example walk file for the output of the ultrasonic sensors

```

1  pi@raspberrypi :~ $ rostopic list
2  / camera1 / point_cloud_face
3  / camera1 / range_visual_face
4  / camera2 / point_cloud_chin
5  / camera3 / range_visual_left
6  / camera4 / point_cloud_right
7  / camera4 / range_visual_right
8  / camera5 / point_cloud_rearDown
9  / cmd_odom
10 / cmd_vel
11 / cmd_vel_ 2
12 / joint_states
13 / lcm_node / obs_env
14 / lcm_node / ultrasonic_env
15 / move_base_simple / goal
16 / pointcloud_process / ground_pointcloud
17 / range_front
18 / range_left
19 / range_right
20 / range_ultrasonic_face
21 / range_ultrasonic_left
22 / range_ultrasonic_right
23 / ros2udp / odom
24 / ros2udp_motion_mode_adv / joystick
25 / rosout
26 / rosout_agg
27 / tf
28 / tf_static
29 / ukd_triple / pose
30 / ukd_triple / state

```

```
31 / ukd_triple_2_goal / path_tag_line  
32 / ukd_triple_2_goal / path_tag_window
```

Listing B.4: Output of the ROS topics running on the Raspberry Pi

C Appendix: Tables

Onboard Raspberry Pi characteristics

	ModelRaspberry Pi Compute Module 4 Rev 1.0
Chipset	Broadcom BCM2711 quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
RAM	1.8 GB
	Memory32 GB SD card hard disk storage
Operating System	Debian 10 (Buster)
Kernel	Linux Kernel 5.4.81 - rt45-v8+

Table C.1: Characteristics of the Raspberry Pi on board the Go1

NVIDIA Jetson Xavier NX characteristics

	ModelNVIDIA Jetson Xavier NX
Chipset	6-Core NVIDIA Carmel ARM v8.2 64-bit CPU
	RAM8 GB
Memory	16 GB SD card hard disk storage chen / 120 GB SSD
Operating system	Ubuntu 18.04.5 LTS
Kernel	4.9.201 -tegra

Table C.2: Characteristics of the NVIDIA Xavier NX on board the Go1

NVIDIA Jetson Nano characteristics

Model	NVIDIA Jetson Nano Developer Kit
Chipset	Quad-Core ARM Cortex-A57 MPCore
	GPUNVIDIA Maxwell-GPU128 cores
	RAM4 GB
	Memory16 GB SD card hard disk spokes
Operating system	Ubuntu 18.04.5 LTS
Kernel	4.9.201 -tegra

Table C.3: Characteristics of the NVIDIA Jetson Nanos on board the Go1

Bibliography

- [AAP08] Haider Ali, Basheer Ahmed and Gerhard Paar. Robust window detection from 3d laser scanner data. Vol. 2. 2008, pp. 115-118.
- [Ble+18] Gerardo Bledt et al. MIT Cheetah 3: Design and Control of a Robust, Dynamic Quadruped Robot. Madrid, Spain: IEEE Press, 2018, 2245-2252. doi: 10.1109/IROS.2018.8593885. URL: <https://doi.org/10.1109/IROS.2018.8593885>.
- [BM92] PJ Besl and ND McKay. A method for registration of 3-D shapes, IEEE T. Pattern Anal., 14, 239-256. 1992.
- [BS03] Peter Biber and Wolfgang Straßer. The normal distributions transform: A new approach to laser scan matching. Vol. 3. 2003, pp. 2743-2748.
- [Bur20] Pawel Burdziakowski. Increasing the geometrical and interpretation quality of unmanned aerial vehicle photogrammetry products using super-resolution algorithms. Vol. 12. 5th MDPI, 2020, p. 810.
- [Cad+16] Cesar Cadena et al. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. Vol. 32. 6. IEEE, 2016, S. 1309-1332.
- [Car+21] Alexander Carballo et al. Characterization of Multiple 3D LiDARs for Localization and Mapping Performance using the NDT Algorithm. 2021, p. 327-334. doi: 10.1109/IVWorkshops54471.2021.9669244.
- [CD22] Jerred Chen and Frank Dellaert. A1 SLAM: Quadruped SLAM using the A1's Onboard Sensors. 2022. arXiv: 2211.14432 [cs.RO].
- [DWB06] H Durrant-Whyte and T Bailey. Simultaneous Localization and Mapping (SLAM): Part i the essential algorithms. Robotics & Automation Magazine, IEEE, 13 (2), 99-110. 2006.

- [Eng+17] Francis Engelmann et al. Exploring Spatial Context for 3D Semantic Segmentation of Point Clouds. IEEE, 2017. doi: 10.1109/iccvw.2017.90. URL: <https://doi.org/10.1109%2Ficcvw.2017.90>.
- [FB81] M. Fischler and R. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. vol. 24. 6. 1981, pp. 381-395. URL: [/brokenurl#http://publication.wilsonwong.me/load.php?id=233282275](#).
- [Gai+16] Adrien Gaidon et al. Virtual Worlds as Proxy for Multi-Object Tracking Analysis. 2016. arXiv: 1605.06457 [cs.CV].
- [GBC16] Ian Goodfellow, Yoshua Bengio and Aaron Courville. Deep Learning. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [Gër19] A. Gëron. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, 2019. isbn: 9781492032595. URL: <https://books.google.en/books?id=HnetDwAAQBAJ>.
- [Hac+17] Timo Hackel et al. Semantic3D.net: A new Large-scale Point Cloud Classification Benchmark. 2017. arXiv: 1704.03847 [cs.CV].
- [HBMO22] Stefan Hensel, Marin B. Marinov and Markus Obert. 3D LiDAR Based SLAM System Evaluation with Low-Cost Real-Time Kinematics GPS Solution. Vol. 10. 9. 2022. doi: 10.3390/computation10090154. URL: <https://www.mdpi.com/2079-3197/10/9/154>.
- [Hen+20] Stefan Hensel et al. Experimental Set-up for Evaluation of Algorithms for Simultaneous Localization and Mapping. 2020, S. 433-444.
- [Hes+16] Wolfgang Hess et al. Real-time loop closure in 2D LIDAR SLAM. 2016, S. 1271-1278.
- [HSL13] C. Hayot, S. Sakka and P. Lacouture. Contribution of the six major gait determinants on the vertical center of mass trajectory and the vertical ground reaction force. Vol. 32. 2. 2013, pp. 279-289. doi: <https://doi.org/10.1016/j.humov.2012.10.003>. URL: <https://www.sciencedirect.com/science/article/pii/S0167945712001212>.
- [IPP21] Malinka Ivanova, Petya Petkova and Nikolay Petkov. Machine Learning and Fuzzy Logic in Electronics: Applying Intelligence in Practice. Vol. 10. 22. MDPI, 2021, P. 2878.

- [JC21] L. Joseph and J. Cacace. Mastering Ros for Robotics Programming: Best Practices and Troubleshooting Solutions When Working with Ros. Packt Publishing, 2021. isbn: 9781801071024. URL: <https://books.google.de/books?id=08GQzgEACAAJ>.
- [Kat18] Benjamin Katz. A low cost modular actuator for dynamic robots. Jan. 2018.
- [Kit+16] Satoshi Kitano et al. TITAN-XIII: sprawling-type quadruped robot with ability of fast and energy-efficient walking. Vol. 3. März 2016. doi: 10.1186/s40648-016-0047-1.
- [KL80] V. Klema and A. Laub. The singular value decomposition: Its computation and some applications. Vol. 25. 2. 1980, pp. 164-176. doi: 10.1109/TAC.1980.1102314.
- [Koi+21] Kenji Koide et al. Voxelized GICP for fast and accurate 3D point cloud registration. 2021, S. 11054-11059.
- [Leh23] Noah Lehmann. Integration of a Unitree Go1 quadruped robot into a university ecosystem. Oct. 2023.
- [LIG20] You Li and Javier Ibanez-Guzman. Lidar for Autonomous Driving: The Principles, Challenges, and Trends for Automotive Lidar and Perception Systems. Vol. 37. 4. 2020, pp. 50-61. doi: 10.1109/MSP.2020.2973615.
- [Ma+20] Qi Ma et al. A Cloud-based Quadruped Service Robot with Multi-Scene Adaptability and various forms of Human-Robot Interaction. Vol. 53. 5th 3rd IFAC Workshop on Cyber-Physical & Human Systems CPHS 2020. 2020, pp. 134-139. doi: <https://doi.org/10.1016/j.ifacol.2021.04.092>. url: <https://www.sciencedirect.com/science/article/pii/S2405896321001786>.
- [May16] Helmut Maier. Fundamentals of robotics. Berlin: VDE-Verlag, 2016. isbn: 978-3-8007-3944-8.
- [MB+22] Andr a Macario Barros et al. A Comprehensive Survey of Visual SLAM Algorithms. Vol. 11. 1. 2022. doi: 10.3390/robotics11010024. URL: <https://www.mdpi.com/2218-6581/11/1/24>.
- [McC+06] John McCarthy et al. A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, August 31, 1955. vol. 27. Jan. 2006, p. 12-14.

- [MF13] Aaron Martinez and Enrique Fernndez. Learning ROS for Robotics Programming. Packt Publishing, 2013. isbn: 1782161449.
- [Mol+13] E. Molinos et al. Comparison of Local Obstacle Avoidance Algorithms. Edited by Roberto Moreno-D'iaz, Franz Pichler and Alexis Quesada-Arencibia. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 39-46. isbn: 978- 3-642-53862-9.
- [MSD18] Ievgeniia Maksymova, Christian Steger and Norbert Druml. Review of LiDAR sensor data acquisition and compression for automotive applications. Vol. 2. 13. 2018, p. 852.
- [OEC19] OECD. Artificial Intelligence in Society. 2019, p. 152. doi: <https://doi.org/https://doi.org/10.1787/eedfee77-en>. URL: <https://www.oecd-ilibrary.org/content/publication/eedfee77-en>.
- [RN21] Stuart Russell and Peter Norvig. Artificial Intelligence, Global Edition A Modern Approach. Pearson Germany, 2021, p. 1168. isbn: 9781292401133. URL: <https://elibrary.pearson.de/book/99.150005/9781292401171>.
- [Sap+21] Azhar Aulia Saputra et al. AQuRo: A Cat-like Adaptive Quadruped Robot With Novel Bio-Inspired Capabilities. Vol. 8. 2021. doi: 10.3389/frobt. 2021.562524. URL: <https://www.frontiersin.org/articles/10.3389/frobt.2021.562524>.
- [SB18] R .S. Sutton and A.G. Barto. Reinforcement Learning, second edition: An Introduction. Adaptive Computation and Machine Learning series. MIT Press, 2018, pp. 1-26. isbn: 9780262039246. URL: <https://books.google.en/books?id=sWV0DwAAQBAJ>.
- [Spa19] Gu"nter Spanner. Robotics and Artificial Intelligence. eBook, PDF. Elektor International Media, 2019. isbn: 978-3-89576-345-8.
- [Spe+21] Mariusz Specht et al. Concept of an innovative autonomous unmanned sys- tem for bathymetric monitoring of shallow waterbodies (INNOBAT system). Vol. 14. 17 MDPI, 2021, p. 5370.
- [Tar12] Sasu Tarkoma. Publish/Subscribe Systems: Design and Principles. English. United States: Wiley, Aug. 2012. isbn: 9781119951544. doi: 10.1002 / 97811118354261.

- [TKDT14] Than Trong Khanh Dat and Phuc Tran. A Study on Locomotions of Quadruped Robot. Vol. 282. Jan. 2014, pp. 595-604. isbn: 978-3-642-41967-6. doi: 10.1007/978-3-642-41968-3_59.
- [Tur09] Alan M. Turing. Computing Machinery and Intelligence. Edited by Robert Epstein, Gary Roberts and Grace Beber. Dordrecht: Springer Netherlands, 2009, pp. 23-65. isbn: 978-1-4020-6710-5. doi: 10.1007/978-1-4020-6710-53. URL: https://doi.org/10.1007/978-1-4020-6710-5_3.
- [Wal09] Richard S. Wallace. The Anatomy of A.L.I.C.E. 2009, p. 181. doi: 10.1007/978-1-4020-6710-513.
- [Wan+18] Guowei Wan et al. Robust and precise vehicle localization based on multi-sensor fusion in diverse city scenes. 2018, S. 4670-4677.
- [Xia22] P. Xiao. Artificial Intelligence Programming with Python: From Zero to Hero. Wiley, 2022. isbn: 9781119820864. URL: <https://books.google.en/books?id=1KR-zgEACAAJ>.
- [Zha+14] Jiaqi Zhang et al. Trot Gait Design and CPG Method for a Quadruped Robot. Vol. 11. 1. 2014, pp. 18-25. doi: [https://doi.org/10.1016/S1672-6529\(14\)60016-0](https://doi.org/10.1016/S1672-6529(14)60016-0). url: <https://www.sciencedirect.com/science/article/pii/S1672652914600160>.
- [Zha+21] Chi Zhang et al. FRL-SLAM: A Fast, Robust and Lightweight SLAM System for Quadruped Robot Navigation. 2021, pp. 1165-1170. doi: 10.1109/ROBIO54168.2021.9739499.
- [ZS14] Ji Zhang and Sanjiv Singh. LOAM: Lidar odometry and mapping in real-time. Vol. 2. 9. 2014, pp. 1-9.

Internet sources

- [cha] champ. ROS Packages for CHAMP Quadruped Controller. URL: <https://github.com/chvmp/champ> (visited on 21. 08. 2023).
- [CSl23] Fraunhofer Institute for Cognitive Systems IKS. Artificial Intelligence. 2023. url: <https://www.iks.fraunhofer.de/en/topics/artificial-intelligence.html> (visited on 06. 06. 2023).
- [DEV23] NVIDIA DEVELOPER. Isaac Gym - Preview Release. 2023. URL: <https://developer.nvidia.com/isaac-gym> (visited on 07/30/2023).
- [Dyn23] Boston Dynamics. Spot. 2023. URL: <https://www.bostondynamics.com/products/spot> (visited on 03. 07. 2023).
- [Ele] Reichelt Elektronik. QR GO1 BATT. URL: <https://www.reichelt.de/de/en/quadruped-go1-akku-4-500-mah-qr-go1-batt-p334403.html?r=1> (visited on 10. 08. 2023).
- [ENG23] A TRIBUTE TO JOSEPH ENGELBERGER. UNIMATE The First Industrial Robot. 2023. url: <https://www.automate.org/a3-content/joseph-engelberger-unimate> (visited on 21. 08. 2023).
- [Enn23] Ennomotive. Industrial IoT Sensor Prices. 2023. URL: <https://www.ennomotive.com/industrial-iot-sensor-prices/#:~:text=The%20cost%20of%20sensors%20is,decisions%20at%20a%20lower%20cost.> (visited on 06. 06. 2023).
- [Fau22] Alberto Paitoni Faustinoni. History of Robots: Origins, Myths, Facts. 2022. url: <https://robotics24.net/blog/history-of-robots-origins-myths-facts/> (visited on 24. 06. 2023).
- [Fra19] Historical Museum Frankfurt. Shakey the robot. 2019. URL: <https://blog.hnf.de/shakey-der-roboter/> (visited on June 26, 2023).

- [IBM23a] IBM. IBM100 - Deep Blue. 2023. url: <https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/> (visited on 06. 06. 2023).
- [IBM23b] IBM. Unsupervised Learning. 2023. url: <https://www.ibm.com/topics/unsupervised-learning#:~:text=Unsupervised%20learning%20C%20also%20known%20as,the%20needed%20for%20human%20intervention.> (visited on 06. 06. 2023).
- [Ind21] The Independent. SpaceX Starship rocket blows up after landing in latest test flight. 2021. URL: <https://www.independent.co.uk/space/spot-robot-dog-spacex-sn10-b1813085.html> (visited 03. 07. 2023).
- [koi23] koide3. hdl graph slam. 2023. URL: https://github.com/koide3/hdl_graph_slam (visited on 21. 08. 2023).
- [Mat19] Kayla Matthews. 5 materials to evaluate for designing, building robust robots. 2019. URL: <https://www.therobotreport.com/materials-rugged-robot-design-building/> (visited on 07/30/2023).
- [MAV] MAVProxyUser. HangZhou Yushu Tech Unitree Go1. url: <https://github.com/MAVProxyUser/YushuTechUnitreeGo1> (visited on 08/10/2023).
- [Nvi.] Nvidia. Quadruped. url: https://docs.omniverse.nvidia.com/isaacsim/latest/ext_omni_isaac_quadruped.html (visited on 21. 08. 2023).
- [Par20] Devin Partida. What Are the Main Components of Robots? 2020. URL: <https://rehack.com/trending/science/what-are-the-main-components-of-robots/> (visited 07. 07. 2023).
- [Ph.20] Michele Baker Ph.D. Hydraulic vs. Pneumatic vs. Electric Actuators. 2020. URL: <https://yorkpmh.com/resources/hydraulic-vs-pneumatic-vs-electric-actuators/#:~:text=Hydraulic%20power%20performance%20is%20also,than%20hydraulic%20and%20electric%20actuators.> (visited on 07. 07. 2023).
- [RL] RoboSense Lidar. RS Lidar SDK. url: https://github.com/RoboSense-LiDAR/rslidar_sdk (visited on 10. 08. 2023).
- [Roba] Robosense. Helios Series. url: <https://www.robosense.ai/en/rslidar/RS-Helios> (visited on 10. 08. 2023).
- [Robb] Unitree Robotics. GO-M8010-6 motor. url: <https://shop.unitree.com/products/go1-motor> (visited on 10. 08. 2023).

- [Robc] Unitree Robotics. Unitree Go1 Battery. URL: <https://shop.unitree.com/products/unitree-go1-battery> (visited on 10. 08. 2023).
- [Robd] Unitree Robotics. UnitreecameraSDK. URL: <https://github.com/unitreerobotics/UnitreecameraSDK> (visited on 18. 08. 2023).
- [Robe] Generation Robots. Go1 Datasheet EN. URL: https://www.generationrobots.com/media/unitree/Go1%20Datasheet_EN%20v3.0.pdf (visited on 10. 08. 2023).
- [Rob16a] Open Robotics. Concepts. 2016. URL: <http://wiki.ros.org/ROS/Concepts> (visited on 18. 07. 2023).
- [Rob16b] Open Robotics. Messages. 2016. url: <http://wiki.ros.org/Services> (visited on 18. 07. 2023).
- [Rob16c] Open Robotics. Services. 2016. url: <http://wiki.ros.org/Services> (visited on 18. 07. 2023).
- [Rob16d] Open Robotics. Topics. 2016. url: <http://wiki.ros.org/Topics> (searched on 18. 07. 2023).
- [Rob18] Kawasaki Robotics. Kawasaki Robotics - History. 2018. URL: https://robotics.kawasaki.com/en/1/anniversary/history/history_01.html (visited on 25. 06. 2023).
- [ROB19] UNIVERSAL ROBOTS. TYPES OF SENSORS IN ROBOTICS - UNIVERSAL ROBOTS. 2019. URL: <https://www.universal-robots.com/in/blog/types-of-sensors-in-robotics-universal-robots/> (searched on 10. 07. 2023).
- [Rob21] Open Robotics. Why ROS? 2021. URL: <https://www.ros.org/blog/why-ros/> (visited on 18. 07. 2023).
- [Rob23a] Open Robotics. Concepts. 2023. URL: <https://docs.ros.org/en/iron/Concepts.html> (visited on 18. 07. 2023).
- [Rob23b] Open Robotics. ROS Index. 2023. URL: <https://index.ros.org/packages/> (visited on 07/18/2023).
- [ROS] micro ROS. micro-ROS for STM32CubeMX/IDE. URL: https://github.com/micro-ROS/micro_ros_stm32cubemx_utils (visited on 10. 08. 2023).

- [Sha21] Shashthrasnehi. Quadruped Robotics: The Evolution of Four-Legged Robots. 2021. url: <https://shashthrasnehi.com/quadruped-robotics-the-evolution-of-four-legged-robots/> (visited on 03. 07. 2023).
- [Sta10] Stanford University. Stanford LittleDog Project. 2010. URL: <https://cs.stanford.edu/groups/littledog/> (visited 03. 07. 2023).
- [Ver20] The Verge. Boston Dynamics' robot dog is patrolling a SpaceX site in latest video. 2020. URL: <https://www.theverge.com/2020/2/19/21144648/boston-dynamics-spot-robot-mass-state-police-trial-issues> (visited on 02. 07. 2023).
- [Web17] ROS-Industrial Website. Using rqt Tools for Analysis. 2017. url: https://industrial-training-master.readthedocs.io/en/melodic/_source/session6/Using-rqt-tools-for-analysis.html (visited on 21. 08. 2023).

Explanation

I hereby declare that I have prepared this thesis independently and without the use of any aids other than those indicated; the ideas taken directly or indirectly from external sources are identified as such. To the best of my knowledge, this work has not yet been submitted to any other examination authority in the same or a similar form and has not yet been published.

Hof, September 13, 2023

JONAS Kemnitzer