# Contents

## List of Abbreviations

| | |
|---|---|
| LED | Light-emitting diode |
| LIDAR | Light detection and ranging |
| ROS | Robot Operating System |
| RTE | Runtime environment |

## Notation

Latin Alphabet

| | | |
|---|---|---|
| $C_1$ | arbitrary constant | 1 |
| $C_r$ | receiver constant | 1 |
| $C_t$ | transmitter constant | 1 |
| $E_r$ | received energy | J |
| $E_t$ | transmitted energy | J |
| $I_r$ | received intensity | $W/m^2$ |
| $a$ | distance between to scanned points | m |
| $b$ | amplitude of a signal | 1 |
| $c$ | speed of light | m/s |
| $d$ | distance | m |
| $d_{min}$ | minimal distance | m |
| $r$ | reflectivity | 1 |
| $t$ | time | s |
| $w$ | width of a pulse | s |
| $x$ | x-coordinate | m |

| | | |
|---|---|---|
| $x_{robot}$ | x-coordinate of robot | m |
| $x_{target}$ | x-coordinate of target | m |
| $y$ | y-coordinate | m |
| $y_{robot}$ | y-coordinate of robot | m |
| $y_{target}$ | y-coordinate of target | m |

Greek Alphabet

| | | |
|---|---|---|
| $\alpha$ | yaw between robot and target | rad |
| $\beta$ | angle between two scanned points | rad |
| $\vartheta$ | inclination angle | rad |

# List of Figures



# List of Tables

# 1 Abstract

This project will consider the development and implementation of a LIDAR-based method for approaching a Charging Station using the TurtleBot 3 platform. The requirement of charging an autonomous system during operation is caused by the limited capacity of storing energy within the system. Therefore, it is necessary to approach a charging station based on information given by embedded sensors. A widely used sensor that should be capable of achieving this is a LIDAR. The capabilities of recognizing materials of different reflectance will be evaluated and compared. Reflective tape is chosen, since it is most efficient. A software solution for localizing and navigation towards a marked charging station will be developed and evaluated.

# 2 Introduction

One of the main problems in mobile robotics is their limited energy capacity. So, it must be concerned increasing the amount of energy which mobile robots are able to carry. Due to the fact that improving the capacity of batteries and enhancing the efficiency of robotics is rather limited. Enlarging the battery increases weight and costs rapidly, that are not always feasible. This leads to the need of charging the battery of a mobile robot during operation to ensure that it can be continuous and autonomous. Furthermore, the ability to charge without interaction with the operator is a first step to create an autonomously action system. According to [CB99] an autonomous system should be capable to behave autonomously and purposeful in the real world. Interpreting this definition leads to the assumption that a purposeful behavior can take longer than the system's runtime with batteries. To perform such an action it has to be charged during operation. To ensure an autonomous operation this must be executed without intervention of the user. So, an autonomous charging solution has to be developed to run the addressed system in autonomous operation.

Addressing this need, a software solution will be developed to charge a Turtlebot 3 of the model Waffle or Burger autonomously during its previously assigned operation. The smaller Burger model is only equipped with a Light Detection and Ranging sensor (LIDAR). So, this is the only feasible data source to recognize the environment, which gives us a set of ranges and intensities depending on the used material. In fact, the mentioned solution will consist of one or multiple ROS nodes. These will evaluate sensor data from the embedded LIDAR of the used system and find marked spots due to intensity variations. Additionally, the capability of the LIDAR to detect different materials based on these variations is evaluated. From these detected spots the software should identify a charging station based on a certain pattern in the intensity variations. Furthermore, it should map the identified charging station/s. During operation it should be able to monitor the battery status of the system and determine the need of charging the robot. If charging is needed the robot should drive towards a charging station until it recognizes contact with the station. After reaching a desired charging state the system should continue with its assigned operation.

# 3 Literature review

As entry point for this project some literature concerning how to process LIDAR scan data is reviewed. The essential step to use a charging station is to detect it with sensors that already are embedded into the system.

In the domestic environment there is the need of autonomous detecting a charging station and charging of a robotic system. Attempts to decrease the inconvenience of charging cleaning robots are done in [Pal04]. Here it is stated that a drawback of cleaning robots is the need of two operations of the user. First, the user has to place the robot to start the cleaning process in the environment to be cleaned. Second, the user has to store the system for recharging it. This creates a significant delay between cleaning operation and charging. The approach is to introduce a flexible recharge cable which is plugged into the robot if needed and is automatically unplugged and collected if enough force is applied to it. The advantage of this attempt is that the robot can disconnect itself from the charging cable by starting its cleaning process. Here the delay between working and charging as well as inconvenience for the user is significantly reduced.

A fully autonomous charging solution would decrease the inconvenience of the operator even more. Therefore, an approach to find a charging station without knowing its position has to be found. It should rely on sensors the system already can offer. So, one of the most capable sensor the robot has is the LIDAR. Therefore, applications of LIDAR are reviewed.

[Rob18] offers a well applicable approach of using the LIDAR of an TurtleBot 3 Waffle to detect a marking that consists of reflective tape and navigation towards it. The goal of this project was to park the robot in an specific orientation and location relative to these marking. Although, the downside of this approach was that the size of the described marking was relatively large in comparison with the parking spot. Therefore, it can be integrated into this project, but needs further improvement.

An interesting approach can be found in research regarding line recognition in autonomous cars. Here it is needed to extract the street surface from obstacles and detect lane marking properly. Unfortunately, LIDARs are often only used as supportive systems in addition to camera-based lane detection. Although the measured intensity of lane lines is rather high compared with surface of roads under standard conditions. So this is probably capable for this project.

In [Oga06] there is the attempt to find lane lines with a LIDAR by building it onto the front bumper of a car. Due to its location the LIDARs sight is rather limited so that it can only detect the actual used lane. Although this is sufficient to keep the vehicle in track. The basic principle to detect lane lines is a threshold attempt and recognizing patterns from the intensity data sets by using the Hough transformation[1]. This is achieved by mapping positive spots to a second order polynomial which should describe the lane line. Finally, the thesis concludes that comparing the detected lines based on LIDAR data is more stable than lines based on camera data. This is achieved due to the measured tracks which were on the motorway where lane lines are rather consistent and have an extraordinary high reflectance.

[1] an approach to identify lines in an image by grouping edge points described by [Hou62]

Another more advanced approach is researched in [Kam08] here the Team AnnieWay designed a lane marker detection for the DARPA Urban Challenge 2007. It is based on a 64 beam LIDAR and allows through its real-time ability a way of getting an offset between a digitally generated map and the real environment while driving. The design is even robust enough to deal with shadows, direct sunlight and changing environmental influences like during sunset. Team AnnieWays approach is able to detect painted lane marking and curbs. Painted markings are detected from higher intensity readings of the LIDAR and curbs by their height change in the range readings. Additionally these elements can be mapped to line segments to increase their detection rate. The advantage of LIDAR readings is that range and intensity readings are rather independent from background light and occurring shadows where camera-based readings are sensitive to these factors.

To summarize research regarding locating a charging station and using LIDARs for a variety of applications was already done. Especially the approach performed by [Rob18] by using sensor data of a LIDAR with the TurtleBot 3 finds well integration into this project.

# 4 Basics

In the following chapter basics required for the aim of the project are explained. Therefore, the TurtleBot 3 as the system platform is introduced. The functionality of the main sensor, a LIDAR, of the platform is shown. Furthermore, ROS as operating system will be considered. The aim of this project is stated as follows. The system should detect a marked charging station during its operation and save the charging station's position. If the system has a low battery level, it should pause its task and drive to the station to recharge. Later it should continue with its operation.

## 4.1 TurtleBot 3

Due to [Rob18] the TurtleBot series is a standard platform robots running ROS. More information about ROS can be found in subsection 4.3. It originates from the Turtle robot driven by the educational programming language Logo in 1967. According to [Pea83] Logo is a general purpose language and known for turtle graphics which are a type of vector graphics. Figure 4.1 shows the third generation of this series. TurtleBot 3 is designed as a small, affordable, programmable and ROS-based robot platform for educational, research, hobby and product prototyping uses. Its goal is, according to [ROS18], to decrease the size of the robot platform and costs without sacrificing functionality and quality by using a modular concept and established components. For this project a TurtleBot 3 Waffle is used. Due to the fact that only sensors are used that are available on all TurtleBot 3 models, the Burger and Waffle Pi could be used with slight parameter adjustment.

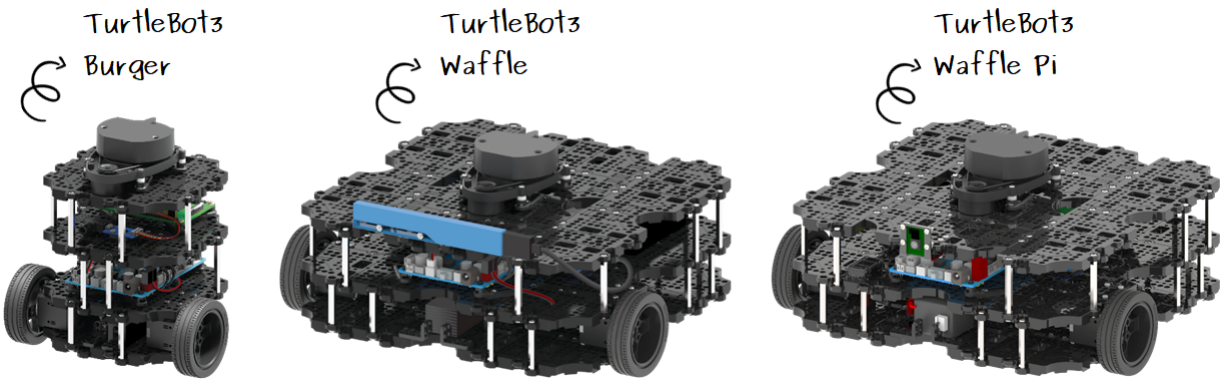


Figure 4.1: TurtleBot 3 models [Rob18]

The TurtleBot 3 Waffle is a three level platform robot with two electric drives directly connected to solid rubber tires. To stabilize it while driving it is equipped with two omni-directional steel balls on the downside of the lowest platform at the back of the robot which can be seen in Figure 4.2. Furthermore, the Waffle uses Intel Joule 570x as central processing board and an OpenCR board for lower level tasks. The Intel Joule 570x is a discontinued single chip board, that runs a Linux distribution. Additionally, the robot is equipped with a LIDAR as main sensor and a custom solution for charging and detecting contact to the charging station which is further described in subsection 4.1.1. Since the camera module is not used in this project, it is disconnected. This should ensure using only sensors that are available on all TurtleBot 3 models.

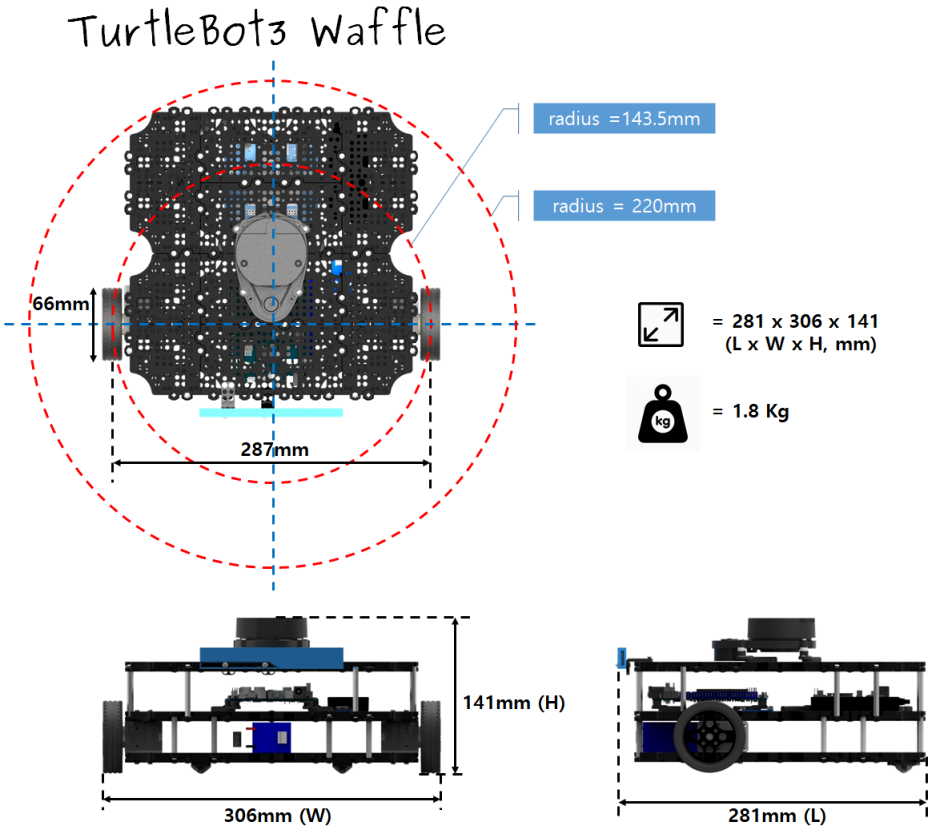


Figure 4.2: Dimensions of the TurtleBot 3 Waffle [Rob18]

### 4.1.1 Custom charging solution

The custom charging solution which is used here consists of two contacts. One at the bottom of the robot with ground contact and another one at the front face. Both have to be in physical contact with the charging station during recharging. The charging circuit is connected to the main battery circuit. Additionally, it is connected to a LED as visual charging indicator and to the OpenCR board to inject the charging state to ROS. The charging station is shown in Figure 4.3. It basically consists of two contact plates and a marker.

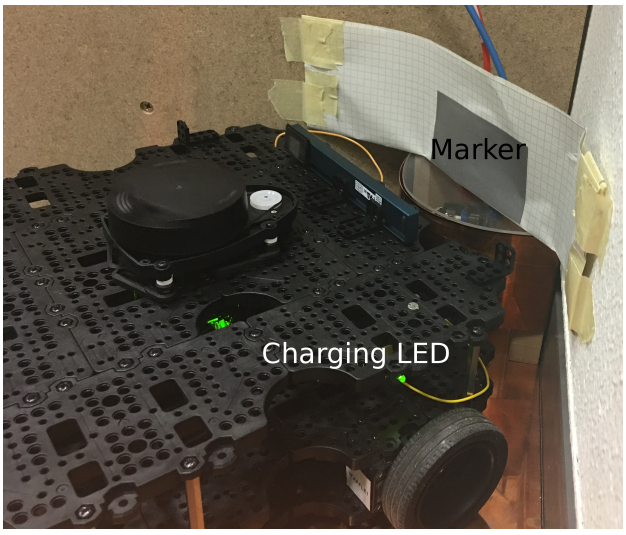


Figure 4.3: TurtleBot 3 Waffle connected to the charging station

## 4.2 LIDAR

According to [AA16] a LIDAR is an active optical remote sensor that has a broad range of applications. These include precise distance measurement over a specific range depending on the system's specifications, measuring the speed of an object from remote position and detecting differences in the ability of a material to reflect incoming light. Therefore, it is often used for robotic applications like the TurtleBot 3. Figure 4.4 shows the model used in the TurtleBot 3 family.

Figure 4.4: 2D Laser Distance Sensor LDS-01 which is used on all TurtleBot 3 models [Rob18]

A basic 2D LIDAR is composed of a light transmitter, e.g. an infrared light-emitting diode, receiver, often a photo diode and a lens in front of them, shown in Figure 4.5. These components placed on a horizontal turnable platform to achieve full 360° scanning angle which is also valid for the LDS-01. Other arrangements are possible for achieving different scanning angles and increasing scan rate and accuracy.

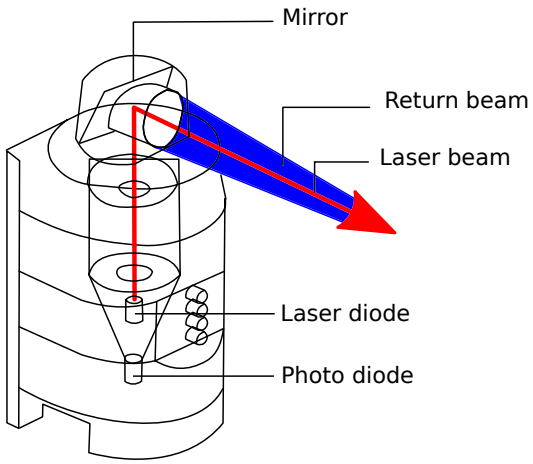


Figure 4.5: Typical scheme of a LIDAR based on [Ric06]

To measure the distance $d$ to an object a short laser pulse is sent to the object and the time $t$ the reflected pulse needs to return is measured. Regarding [AA16] the pulse travels a round-trip distance $2d$, the distance $d$ to the object is determined by equation 4.1. Here $c$ is the speed of light. To eliminate inaccuracies not only one pulse is sent, but a series of pulses is used and the average

result is taken into account to determine the distance $d$. This principle is known as *transit time measurement*[2].

$$d = \frac{c\,t}{2} \tag{4.1}$$

Measuring a distance as described in equation 4.1 is rather practicable and has developed a wide range of applications, like in autonomous vehicles, archeology, geology, military and spaceflight. Depending on the application the following parameters of the LIDAR have to be taken into account: the minimum detection range, the maximum detection range and the tolerance in detecting objects. Additionally the scan rate, angular range and angular resolution have to be considered and the LIDAR has to be chosen according to the applications demands. In this project the LIDAR model was already given due to the sensors which are embedded in the TurtleBot 3 series. The minimum detection range describes the smallest distance the system can be detect, which in example of the used LIDAR Model HLS-LFCD2 is according to [Sto14] 120mm. In contrast the maximum detection range is the greatest distance the system is able to detect obstacles, here 3,500mm. Tolerances describe the variation in measured ranges due to misreadings and noise, here 15mm in range to 499mm and up to 5% in greater ranges. Scan rate is defined by number of full scan cycles per second. Furthermore, a scan cycle is described by the angular range which gives scannable angles, here a full range of 360°. Angular resolution means the angle between two scanned points in degree, here 1°.

According to [Jut09] the energy of a laser scanner receives can be determined with equation 4.2 by the amplitude of the signal $b$ and the duration of the pulse $w$, where $c$ is constant.

$$E_r = c \cdot b \cdot w \tag{4.2}$$

Building an energy balance between transmitted and received energy like $E_r = E_t \cdot (1 - \text{losses})$ yields equation 4.3, where $E_r$ is the received energy, $E_t$ is the transmitted energy, $C_t$ and $C_r$ are constant terms of transmitter and receiver, $d$ is the distance to the object, $e^{-2\,b\,d}$ is the atmospheric attenuation and $f(c_s)$ contains all other influencing factors like surface material and geometry. With given amplitude and range the intensity of received light can be determined with equation 4.4 where all constants are built in the arbitrary constant $C_1$.

$$E_r = E_t \cdot C_t \cdot C_r \cdot d^{-2} \cdot e^{-2\,b\,d} \cdot \cos\vartheta \cdot f(c_s) \tag{4.3}$$

$$I_r = C_1 \cdot b \cdot w \cdot d^2 \cdot e^{2\,b\,d} \tag{4.4}$$

[2] a method to obtain a distance of an object from the round trip time of a light pulse [AA16]

## 4.3 ROS

According to [Qui09] ROS is an open-source, meta-operating system for robots. The ROS runtime environment is structured like a peer-to-peer network of processes. In the context of ROS these processes are referred to as nodes. These have multiple options to communicate for instance synchronous message sending over services, asynchronous data streaming over topics and storing data on parameter servers[3]. Furthermore, it is not a realtime framework, but offers possibilities for realtime operation. Objectives of the ROS project are as follows. First and foremost it should be thin and easy to integrate into other robot software frameworks. Additionally, it should be easily accessible with clean interfaces and language independent. Implementations for Python, C++ and Lisp are already added to the framework and experimental libraries for Java and Lua are available. Furthermore, unit and integration testing should be easy. Finally, ROS should be scalable and able to handle large runtime systems and development processes.

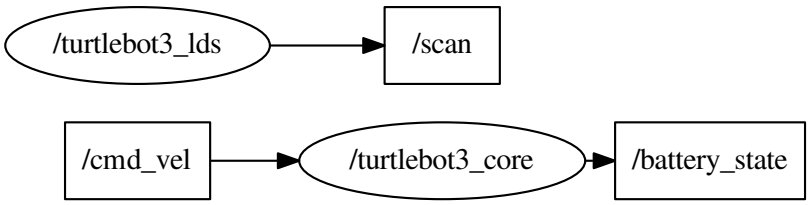


Figure 4.6: Simplified RQT Graph after startup

Figure 4.6 shows nodes and topics running after a default startup recorded with the tool *rqt_graph*. Important for this project are the */turtlebot3_core* and */turtlebot3_lds* nodes which provide the */cmd_vel* topic that enables controlling movements of the system. Additionally, the */battery_state* topic provides information of the actual battery voltage and whether the system is charged or not. Furthermore, the */scan* topic provides all scan data measured by the LIDAR.

[3] shared dictionaries which are accessible via API calls [ROS18]

# 5 Implementation of the charging station localization algorithm

In the following chapter the capability of the LIDAR to detect different materials is evaluated and the implementation of localizing and navigating to the charging station is described. Additionally, a method of reading the actual charge state of the battery and how to interrupt already running operations is described.

## 5.1 Capabilities of the embedded LIDAR

At first the capabilities of LIDAR with different materials were tested regarding their reflecting intensity. In fact the LIDAR should be able to differentiate between material with a remarkably high reflectance like metals, e.g. polished copper or steel, and materials with low reflectance like black electrician tape. Reflectance means here the ability of a material to reflect with a high intensity. So specimens with a bar code of electrician tape and copper or steel are prepared. The bar code consists of three 3cm wide stripes with a spacing of 3cm to each other. Additionally, bare copper and steel plates are recorded and compared with the bar code records. In figure 5.1 a bar code of electrician tape on a polished copper plate is recorded with a distance of 40cm. In comparison a copper plate with the same distance is recorded. The reflected intensity of the recorded copper plate is lower without a bar code than with one. In theory it should be higher caused by the electrician tape reducing the total reflectance of the plate. The lower intensities might be caused by the overall quite high variance of the values. Furthermore, in the intensity values no pattern caused by the three striped bar code is noticeable. This is probably caused by the quality and noise resistance of the LIDAR. In conclusion a polished copper plate combined with electrician tape is not usable in this project.

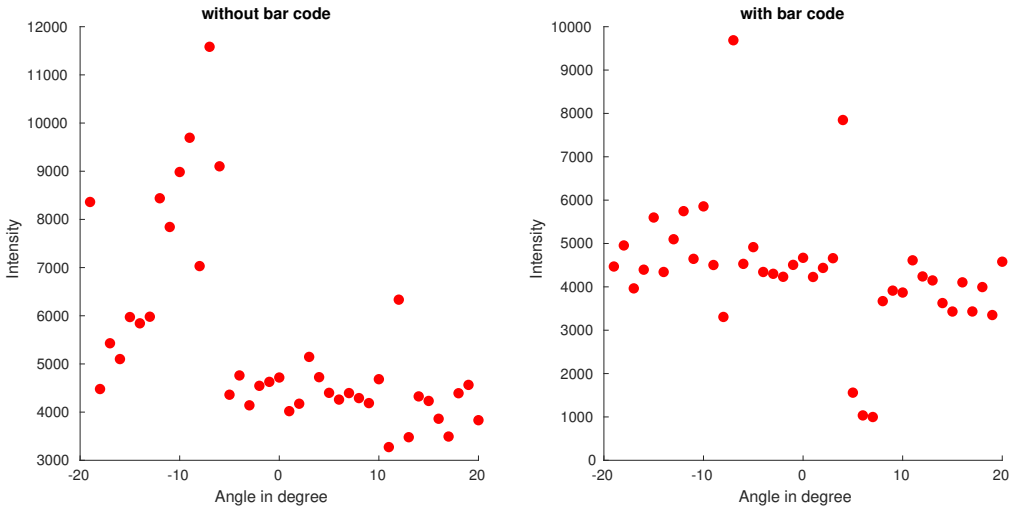


Figure 5.1: Recorded LIDAR data in front of a copper plate

Changing to another material, steel, shows a similar result comparing in Figure 5.2. But the steel plate shows the estimated result of copper. Here the averaged intensity is lower with a bar code. According to recorded LIDAR data no significant irregularities are detectable.

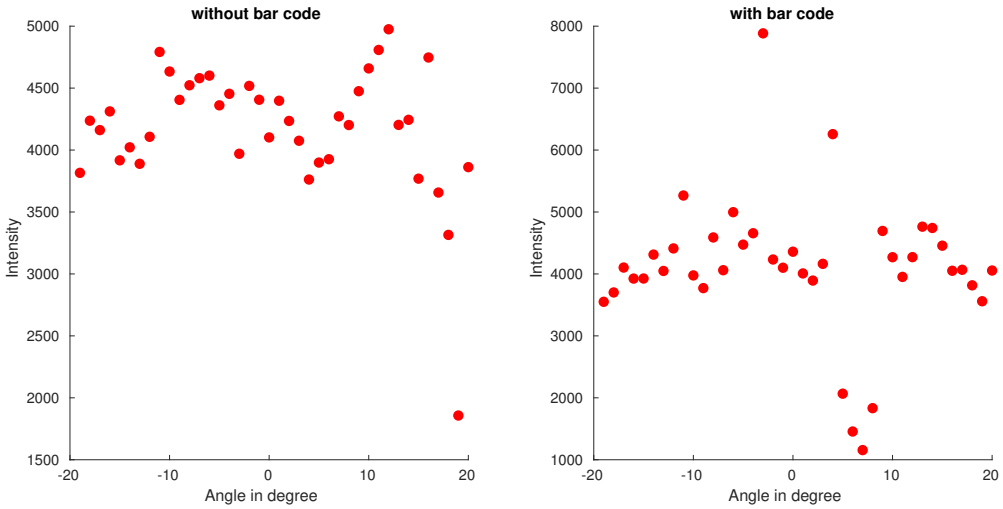


Figure 5.2: LIDAR data in front of a steel plate

In conclusion a material with even higher reflectance is chosen. Reflective tape is considered to be the best option as marking material due to its extraordinary high reflectance in comparison with wood and its availability. Alternatively, reflective paint could be used. But due to easier appliance on surfaces reflective tape is favored. With reflective tape differences in intensity are observable, shown in Figure 5.3. The approach of this project should be based on the *automatic_parking* package provided by [Rob18]. To increase the visibility of marked spots over higher distances a value called reflectivity is introduced with equation 5.1 according to [Rob18], where $r$ is the reflectivity, $d$ the measured distance and $i$ the intensity. In Figure 5.3 the three bar code stripes are clearly detectable due to differences in intensity. Furthermore, the reflectivity differences are even more significant.

$$r = \frac{d \cdot {I_r}^2}{10000} \tag{5.1}$$

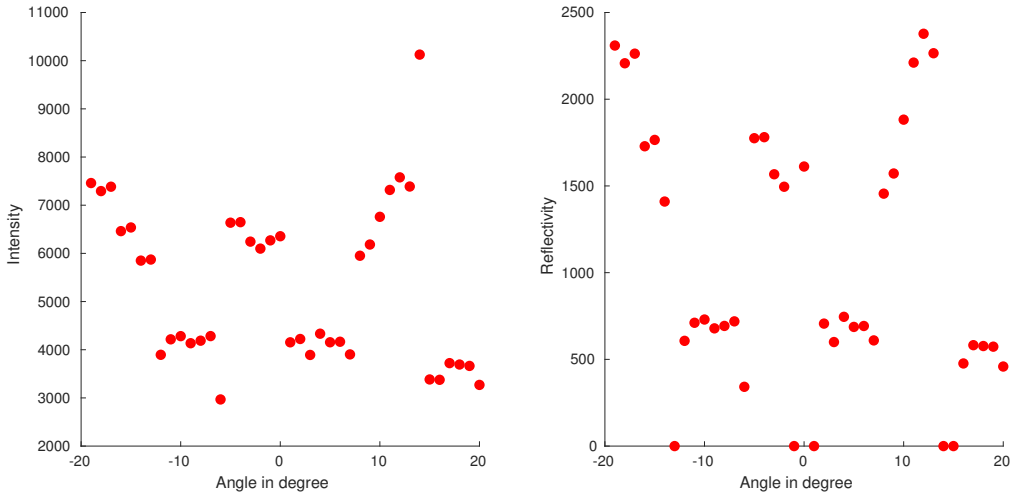


Figure 5.3: Recorded LIDAR data in front of a paper surface with 3x 3cm reflective tape markings spaced with 3cm

Now for each scanning angle its reflectivity is compared to an adaptive reflectivity threshold with an underlying hard coded fall back threshold. If the calculated reflectivity is high enough the spot is considered to be marked. Furthermore, the scan data from one sensor turn is simplified to a true/false array, where true is a marked spot and false the opposite. Due to the fact that the resolution of the LIDAR is 1 °, its capability is rather limited in detecting small surfaces with high reflectance in greater distances. This limitation is shown in Figure 5.4, where $P1$ and $P2$ are two neighbored scanned points, $\beta$ is the angle between these, $a$ is the distance between the points and $d$ is the distance between scanner and plate. The relation can be described with Equation 5.2 and solved for $a$ in Equation 5.3. Therefore, the minimal detectable distance between two points increases with the distance to the object.

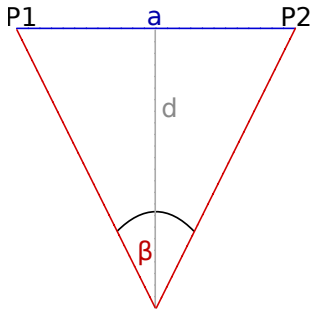


Figure 5.4: Limitation of angular resolution

$$\tan\left(\frac{1}{2}\cdot\beta\right)=\frac{0.5\cdot a}{d} \tag{5.2}$$

$$a=2\cdot d\cdot\tan\left(\frac{1}{2}\cdot\beta\right) \tag{5.3}$$

So for a higher detection rate a 9cm wide marking with reflective tape is chosen. The calculation of the adaptive reflectivity threshold is based on the following algorithm in Listing 5.1 derived by [Anj08]. This enables the possibility to equalize or even weighten specifically the distribution of marked and not marked values. Therefore, the threshold is flexible to ambient light conditions and different coatings of walls and marked spots in the environment of the system. The total weight has to be derived experimentally. Using such an algorithm decreases the detection rate significantly. The algorithm in Listing 5.1 averages the smallest and highest value to receive an medium one. Then it moves this medium value as long to the right until the weight of values on both sides of the medium value is described by weight on left side multiplied with an arbitrary constant is the same as the weight on the left side.

Listing 5.1: Adaptive threshold implementation in python

```
n, e = np.histogram(data)
l_s = e[0]
l_e = e[-1]
l_m = (l_s + l_e) / 2.0
W_l = getWeight(n, getIndex(e, l_s), getIndex(e, l_m))
W_r = getWeight(n, getIndex(e, l_m) + 1, getIndex(e, l_e))

while W_r > TOTAL_WEIGHT * W_l:
        W_r -= n[getIndex(e, l_e) + 1]
        l_e = e[getIndex(e, l_e) - 1]

        if (l_s + l_e) / 2.0 < l_m:
                W_l -= n[getIndex(n, l_m)]
                W_r += n[getIndex(n, l_m)]
                l_m -= 1.0
```

## 5.2 Implementation

In this chapter the implementation of the charging behavior is described. It will consider locating and navigating the charging station. Furthermore, it will describe reading the battery state and controlling ROS nodes. It is assumed that only one charging station exists in the environment of the robot and this station is marked with reflective tape. Furthermore, all obstacles can be detected by the LIDAR, e.g. appropriate height.

### 5.2.1 Localizing the charging station

From all detected spots of one scan an average is calculated to approximate the position of the marked area relative to the system's frame. This position is published. Furthermore, a node is listening to the position publishing topic and to the tf topic of the robot. The tf topic is a helper to

transform coordinates between multiple frames known by the system. It transforms the position from the robots frame to a global map and publishes the transformed position with a specific lifetime again. The lifetime is needed to ensure that old spots are removed because of an increasing inaccuracy due to the reliance on odometry. Another node is listening to the transformed spot positions and calculates the average of spots from a specified number of latest scan data sets. This position is considered to be the approximated location of the charging station. Averaging the spots has to be done due to the fact that the location is not determinable from one scan data set. Single spots are not accurate enough and often have a huge offset because of incorrect reflections, low quality of the LIDAR, general misreadings and noise. The estimated position of the charging station is now with constant frequency published not regarding the event of new incoming spots. If the node */goto_charging_station* is started by the monitoring node because of low battery charge the second position and additionally an orientation is calculated from the estimated location of the charging station to determine a target for the *turtlebot3_navigation* package. This package provides scripts for navigating around obstacles to a given target. First the yaw between the robots and charging stations position is determined by their Cartesian coordinates in the map frame with equation 5.4.

$$\alpha = \arctan\left(\frac{y_{target} - y_{robot}}{x_{target} - x_{robot}}\right) \tag{5.4}$$

From $\alpha$ a quaternion[4] can be derived with the build-in Euler-to-quaternion-transformation in the tf. This gives the robot orientation for the goal that should be published to the navigation topic. In fact this will result in the robots orientation always being in front of the target. The position for the goal is determined by equation 5.5.

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_{robot} + d \cdot \cos\alpha \\ y_{robot} + d \cdot \sin\alpha \end{pmatrix} \tag{5.5}$$

Where $d$ is the distance between robot and target with an offset defined by equation 5.6 with a constant $d_{min}$ which is needed to avoid having a target position in a not reachable location. That means the robot can drive to this position.

$$d = \sqrt{(x_{target} - x_{robot})^2 + (y_{target} - y_{robot})^2} - d_{min} \tag{5.6}$$

### 5.2.2 Navigating to the charging station

The general approach of navigation to the charging station is shown in Figure 5.5. Here is can be seen that the navigation is started after the battery voltage drops under a given reference voltage

[4] describes an orientation in an three-dimensional space [Kui99]

and is split into navigating near to the station with the navigation package and connecting to the station based on latest intensity reading from the LIDAR.

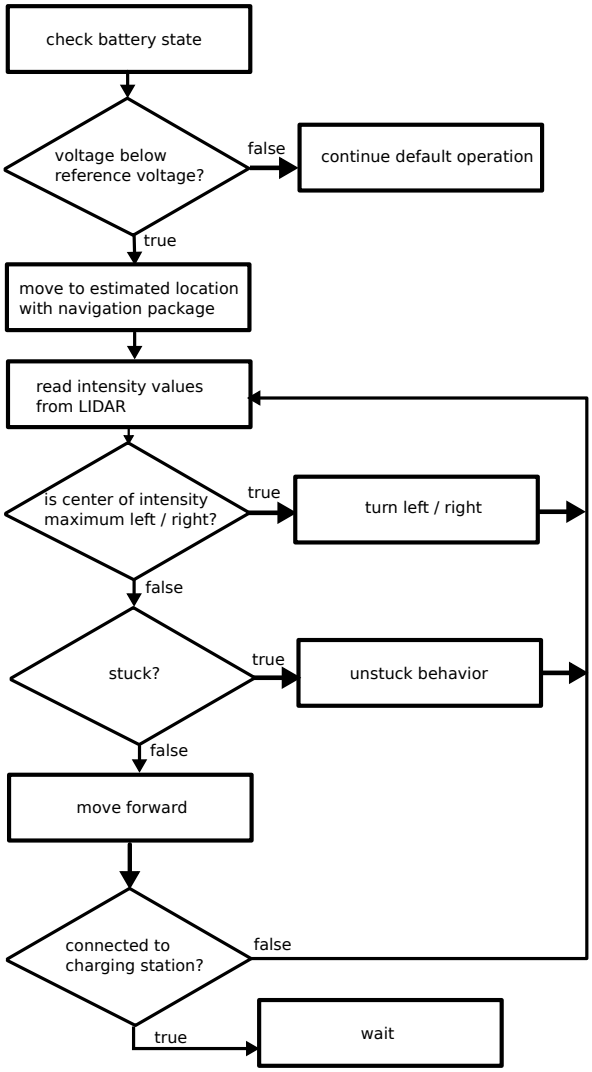


Figure 5.5: Flow chart of the navigation algorithm

Now the navigation package will publish movement commands to the topic */cmd_vel* according to a previous by the navigation package planned most cost efficient route. This results in the robot avoiding obstacles on its way to the charging station and stopping in a distance $d_{min}$ in front of it. The next phase of moving to the station starts due to the circumstance that it is not feasible to go directly to the station with the navigation package. This is caused by the needed tolerances of the navigation package to work probably. To reach the target position that allows charging the robot it will determine a section of high intensities and move to an averaged location of these until it detects that the robot is being charged. These movements rely only on the latest LIDAR scan to minimize the influence of incorrect scan data which increases with decreasing distance to the station. Furthermore, movement is simplified to driving forwards or backwards, turning left or right and unstuck behavior left or right with linear and rather low velocities. Here unstuck behavior means moving slowly backwards with slight left or right turning to free a stuck tire. If the system detects contact with the charging station all movements are stopped and is set to a waiting state.

### 5.2.3 Monitoring the battery status

The status should be monitored by reading the voltage from the topics *battery_states* published *sensor_msgs/BatteryState* Message. These were recorded under normal conditions while driving in figure 5.6. The discharging behavior was observed as parabola with slight variation. Therefore, a reference voltage is introduced that invokes when reached the charging behavior. Additionally, a target voltage is introduced that leads to the assigned behavior of the robot. Reference and target voltage are experimentally acquired.

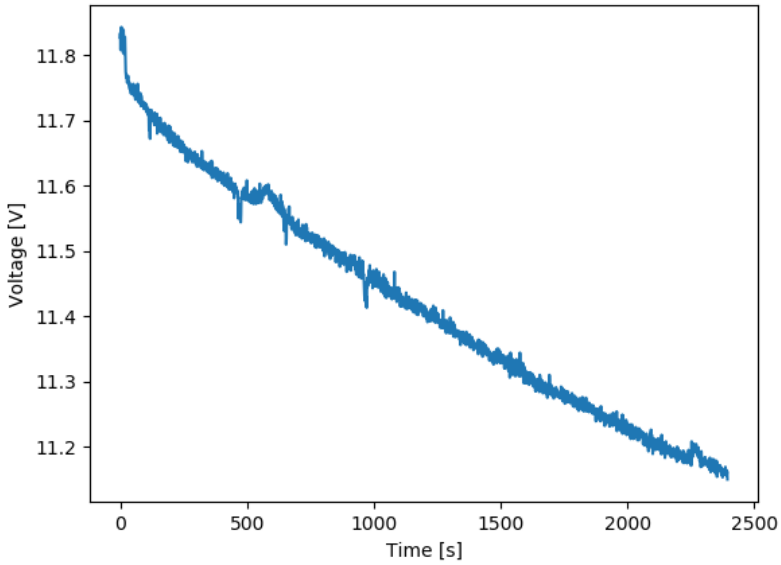


Figure 5.6: Recorded battery voltage of the TurtleBot 3 Waffle

Additionally, a button at the front of the robot was introduced to enable charging and later detect contact with the charging station. This button is connected to one digital input pin of the OpenCR board and the state is published in the *BatteryState*. So other nodes can receive and process this state. Although, it is only possible to detect the actual battery voltage, not the current and whether the robot is in contact with the charging station. Therefore, it is not possible to detect if the battery is completely charged.

### 5.2.4 Interrupting operating ROS nodes

An essential feature of the charging solution is monitoring the battery state during an operation and pausing the ongoing task to invoke a routine moving towards the charging station. This can be accomplished by in the *rospy* library included functions. Here *launch*-files can be started and stopped individually. *launch*-files are predefined scripts to start individual or multiple nodes.

# 6 Evaluation

In the following chapter the ability of the software to detect the marking on the charging station and connect to it will be evaluated. This will be done by setting up different environments and letting the robot look for the charging station. Four environments are set up and named after letters that can be recognized in the obstacle orientation. The charging station is always located in the upper right corner. Its position is determined by publishing at least three points manually where it is to be estimated based on the LIDAR mapping at the end of the test and later averaged out of these points. As factor to evaluate the accuracy of location the station the distance between the manually estimated position and the by the robot estimated position shall be considered. Positions of the charging station are published with 10Hz. The movement of the robot is partially random and manual controlled. Additionally, it will be tested if the robot can navigate to the charging station and connect to it. It is to note that the final approximated location of the charging station is marked with a green cycle and the initial position of the robot is marked by the turtle symbol.

The first environment shall be without any obstacles and therefore named *plain*. The environment is shown in Figure 6.1. The initial distance between real and estimated based on LIDAR data is quite high due to the fact that the robot starts at the lower left corner and is barely able to detect the station, shown in Figure 6.2. After 1000 measurements drops down significantly because the system has moved towards the station. Later it moves constantly lower if the system is in the range of 1m around the station and reaches it local minimum. The distance increases slightly after the robot moving away from the station but stay at a local maximum after leaving a detectable distance.

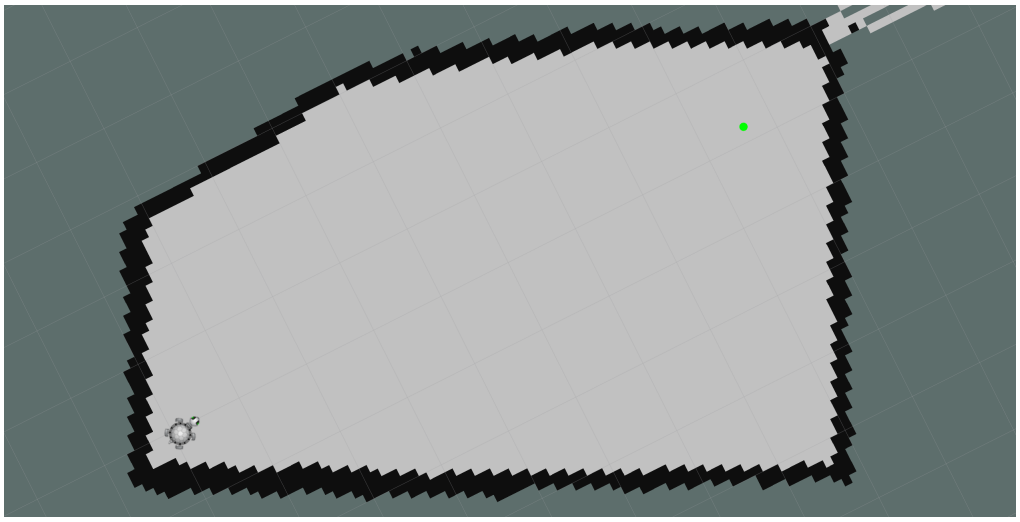

Figure 6.1: Recorded map of test run *plain*

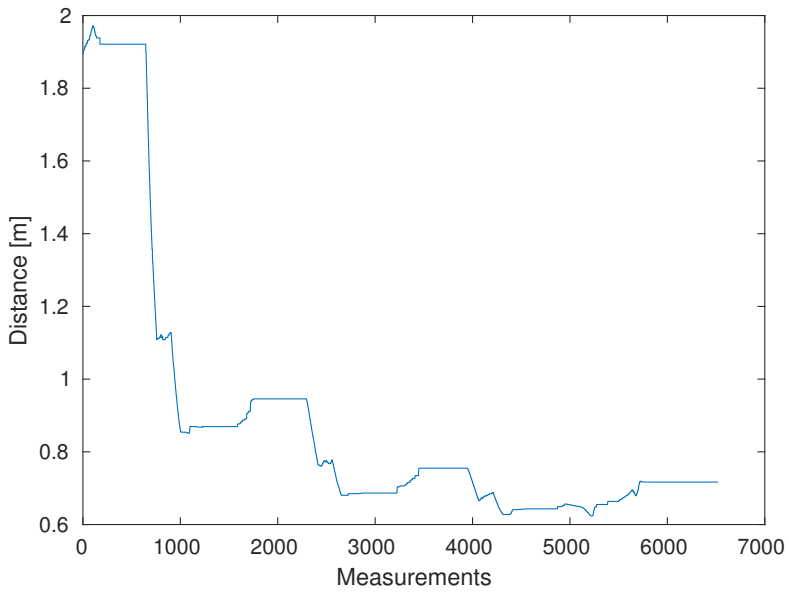


Figure 6.2: Distance between approximated position of charging station to real position in test run *plain*

In the second environment are two L-formed obstacles placed, shown in Figure 6.3. Therefore, it is named *two L*. The robot's starting position is again the lower left corner. Here the distance in Figure 6.4 between real and estimated position of the charging station reduced significantly because before the marked station is clearly out of sight of the LIDAR. The local minimum is lower than in environment *plan* because the robot stays during operation closer to the station.

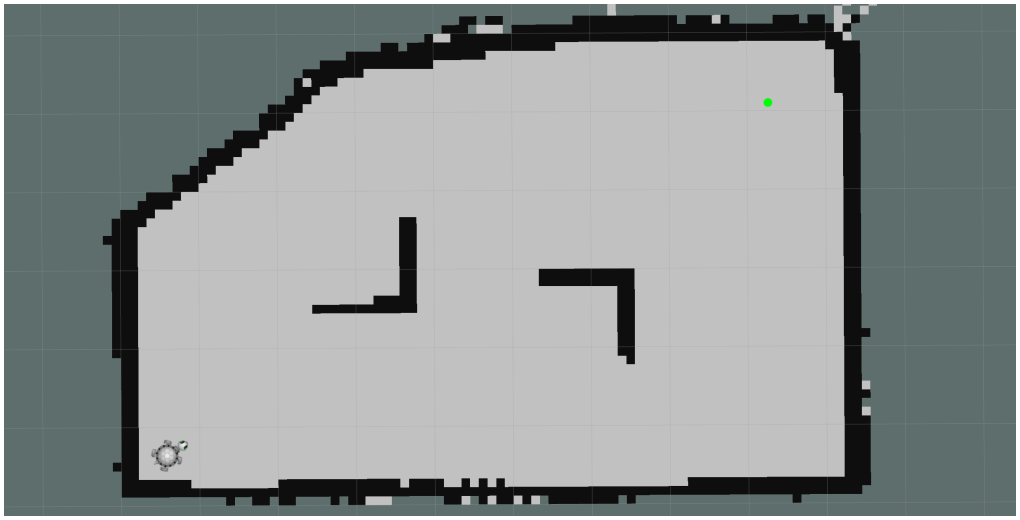

Figure 6.3: Recorded map of test run *two L*

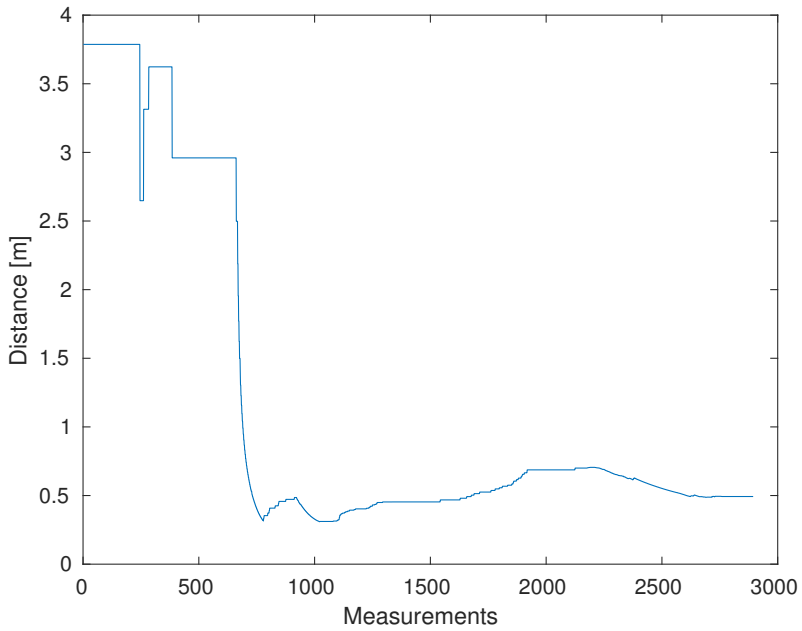


Figure 6.4: Distances between approximated position of charging station to real position in test run *two L*

The third environment consists of one M-formed obstacle and one L-formed, shown in Figure 6.5. Therefore, it is named *ML*. The robot's starting position is again the lower left corner. And the initial distance in Figure 6.6 stay rather long quite high due to the limited sight towards the station caused especially by the L-formed obstacle in front of the station. After coming relatively close to the station the distance stays small.

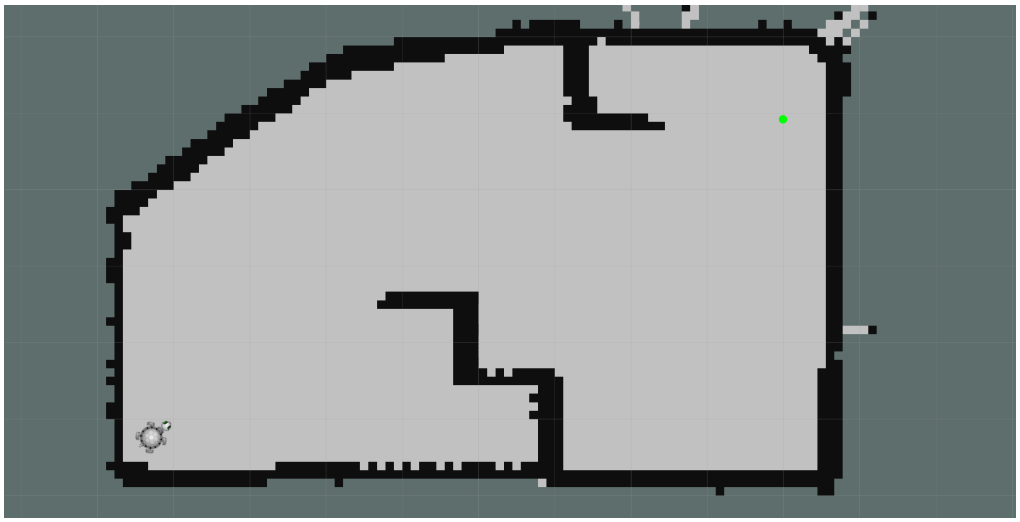

Figure 6.5: Recorded map of test run *ML*

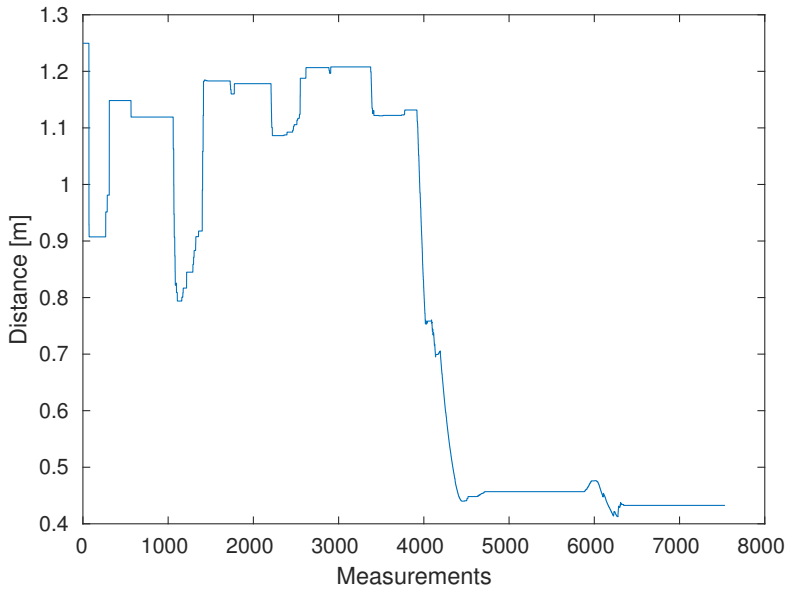


Figure 6.6: Distances between approximated position of charging station to real position in test run *ML*

In the last environment one P-formed obstacle and one L-formed are placed, shown in Figure 6.7. Therefore, it is named *PL*. The robot's starting position is in the lower right corner behind the P-obstacle. Here the distance, shown in Figure 6.8 drops significantly early after moving out of the P-obstacle because the LIDAR is line of sight with the station.

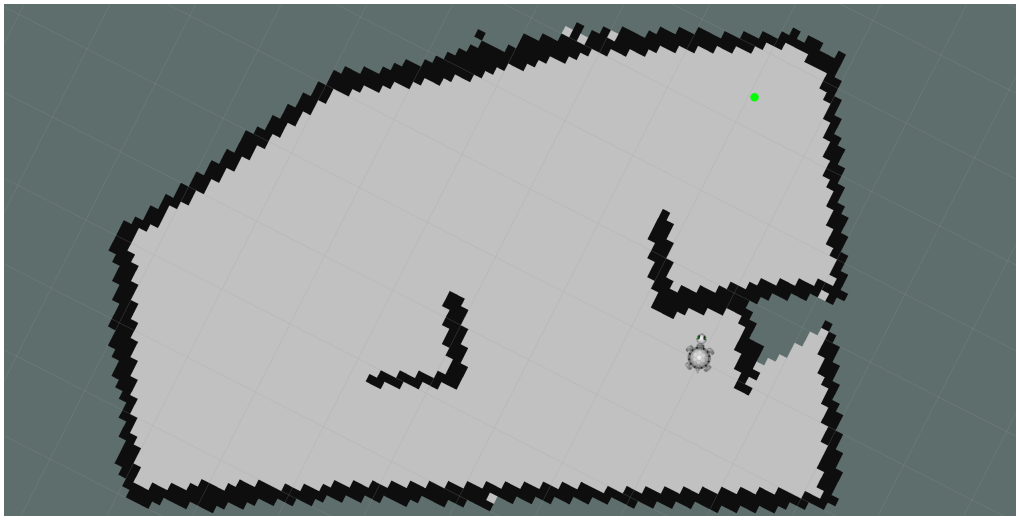

Figure 6.7: Recorded map of test run *PL*

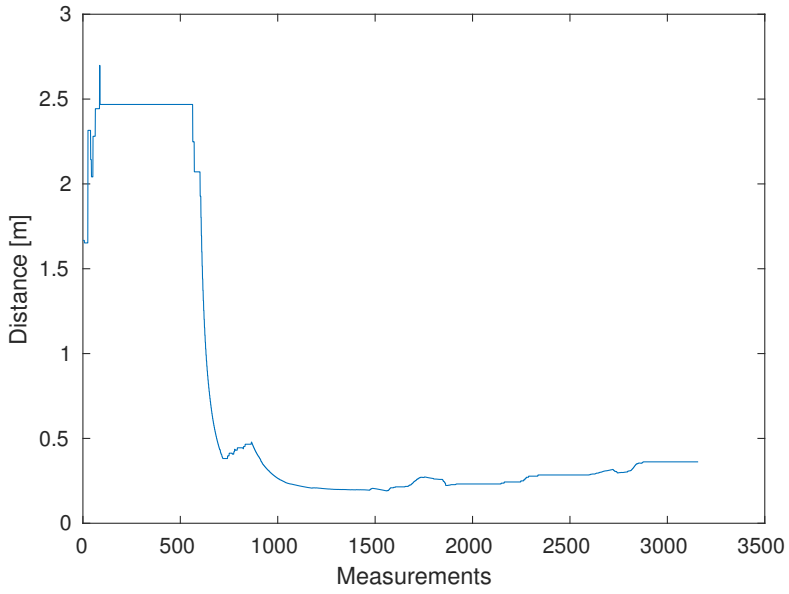


Figure 6.8: Distances between approximated position of charging station to real position in test run *PL*

In general it can be stated that the initial distances are rather high but decrease with the system moving towards the station. The line of sight is an essential factor for this behavior. In all four environments the robot was able to connect to the charging station indicated by a LED. Even with such a high error it is possible to navigate to the station because the estimated location is only relevant for the first phase of movement. Connecting to the station is done by processing the latest intensity readings. It is hardly possible to estimate a more accurate position of the charging station due to high misreadings in range data from the LIDAR probably caused by the extra ordinary high reflectance of the reflective tape. These misreadings displace the measured distance of an scanning angle closer to the sensor than they are in reality. So it is necessary to process the latest scan of the LIDAR regarding its intensity values. But even those consist of some misreadings.

# 7 Conclusion

In conclusion the capabilities of the used LIDAR are rather limited regarding localization of a with high reflectance material marked point. Recognizing patters is only possible in close distances and therefore, it is not feasible to detect even simple patterns in an practical environment. This may could be compensated with a more sensitive LIDAR. The essential factor for localizing the charging station is the necessity of a line of sight between the LIDAR and the marked object.

The available navigation package is a practical way to approach a desired location, but it leeks the ability to target a position precisely. This is basically due its reliance on odometry. Furthermore, its routes are sometimes unnecessary complicated, e.g. driving a curve if the target can be reached by driving a straight line regarding the target orientation. In the second navigation phase, locating the exact position with the latest LIDAR data is also rather error-loaded due to misreadings of high reflectance materials in close ranges. Although constantly processing these data can lead to reaching the desired location. Nevertheless, the robot has to continuously try to connect to the charging station based on the intensity readings. This leads to wall-bumping and depending on the incidence angle of the robot to the station to a connecting time in the range of up to several minutes. Overall it is to state that several sources of errors could be identified and have to be eliminated.

The TurtleBot 3 series as hardware platform offers a lot of possibilities. Though its modular design combined with ROS as software platform it is rather straight forward to achieve certain tasks. Therefore, it is rather capable for this project. Nevertheless, a long term testing should be performed to obtain sophisticated data regarding the success rate of this project. In the previously described environments all four tries were successful, but the number of testing is too small to obtain a reasonable statement. Although, problems with connecting remotely to the robot prevented long term evaluations. Additionally, some connection losses with the OpenCR board occured. So a further approach would be to stabilize especially the connection between the system and a remote operator and investigating in the overall stability of the system.

# A Data sheets

Table A.1: Hardware Specifications TurtleBot 3 Waffle [Rob18]

| | |
|---|---|
| Maximum translational velocity | 0.26 m/s |
| Maximum rotational velocity | 1.82 rad/s (104.27 deg/s) |
| Maximum payload | 30kg |
| Size (L x W x H) | 281mm x 306mm x 141mm |
| Weight (+ SBC + Battery + Sensors) | 1.8kg |
| Threshold of climbing | 10 mm or lower |
| Expected operating time | 2h |
| Expected charging time | 2h 30m |
| SBC (Single Board Computers) | Intel® Joule™570x |
| MCU | 32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS) |
| Actuator | Dynamixel XM430-W210 |
| LDS(Laser Distance Sensor) | 360 Laser Distance Sensor LDS-01 |
| Camera | Intel® Realsense™R200 |
| IMU | Gyroscope 3 Axis<br>Accelerometer 3 Axis<br>Magnetometer 3 Axis |
| Power connectors | 3.3V / 800mA<br>5V / 4A<br>12V / 1A |
| Expansion pins | GPIO 18 pins<br>Arduino 32 pin |
| Peripheral | UART x3, CAN x1, SPI x1, I2C x1, ADC x5, 5pin OLLO x4 |
| Dynamixel ports | RS485 x 3, TTL x 3 |
| Audio | Several programmable beep sequences |
| Programmable LEDs | User LED x 4 |
| Status LEDs | Board status LED x 1<br>Arduino LED x 1<br>Power LED x 1 |
| Buttons and Switches | Push buttons x 2, Reset button x 1, Dip switch x 2 |
| Battery | Lithium polymer 11.1V 1800mAh / 19.98Wh 5C |
| PC connection | USB |
| Firmware upgrade | via USB / via JTAG |
| Power adapter (SMPS) | Input : 100-240V, AC 50/60Hz, 1.5A @max<br>Output : 12V DC, 5A |

# References


[AA16] Mohammad D. Al-Amri, Mohamed El-Gomati, and M. S. Zubairy. *Optics in Our Time*. English. Springer, 2016.

[Anj08] António Anjos and Hamid Shahbazkia. *Bi-Level Image Thresholding - A Fast Method*. Jan. 2008.

[CB99] Jose C. Brustoloni. *Autonomous Agents: Characterization and Requirements*. Oct. 1999.

[Hou62] Paul VC Hough. *Method and means for recognizing complex patterns*. US Patent 3,069,654. 1962.

[Jut09] Boris Jutzi and H Gross. *Normalization Of Lidar Intensity Data Based On Range And Surface Incidence Angle*. Jan. 2009.

[Kam08] Soren Kammel and Benjamin Pitzer. "Lidar-based lane marker detection and mapping." In*Intelligent Vehicles Symposium*. IEEE. 2008, pp. 1137–1142.

[Kui99] Jack B Kuipers et al. *Quaternions and rotation sequences*. Vol. 66. Princeton university press Princeton, 1999.

[Oga06] Takashi Ogawa and Kiyokazu Takagi. *Lane recognition using on-vehicle lidar*. IEEE, 2006.

[Pal04] Jordi Palacin, José Antonio Salse, Ignasi Valgañón, and Xavi Clua. *Building a mobile robot for a floor-cleaning operation in domestic environments*. 2004.

[Pea83] Roy D Pea. *Logo Programming and Problem Solving.[Technical Report No. 12.]*. 1983.

[Qui09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. *ROS: an open-source Robot Operating System*. Kobe, Japan, 2009.

[Ric06] Patrick Richter. "Entwicklung eines kamerabasierten Evaluierungsverfahren für Fahrerassistenzsysteme mit autonomen Notbremseingriff." MA thesis. Dresden: Hochschule für Technik und Wirtschaft, 2006.

[Rob18] Robotis. *Turtebot 3 e-Manual*. [Online; accessed 11-June-2018]. 2018. URL: `http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/#overview`.

[ROS18] ROS.org. *ROS.org Documentation*. [Online; accessed 11-June-2018]. 2018. URL: `http://wiki.ros.org`.

[Sto14] HL Data Storage. *LDS1.5 Specifications*. 2014.