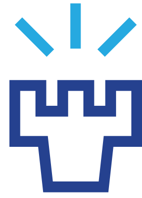


FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Ville Kivikko
Markus Kyllönen
Lassi Perälä
Katri Säily

TRANSFER LEARNING FOR MOBILE ROBOTS

Bachelor's Thesis
Degree Programme in Computer Science and Engineering
June 2021

Kivikko V., Kyllönen M., Perälä L, Säily K. (2021) Transfer Learning for Mobile Robots. University of Oulu, Degree Programme in Computer Science and Engineering, 50 p.

ABSTRACT

In this project transfer learning can be defined as transferring previously learned knowledge to a new environment and making use of it to avoid obstacles. The feasibility of transfer learning was studied in a situation where a robot is given a task to navigate to a user-defined location in a virtual environment without hitting walls and utilizing reinforcement learning to teach the robot, which means that the robot will receive rewards according to the way it moves in the environment and how close it is to the goal location. In this project everything is done and tested in simulation. First the robot is trained in a standard environment, which is a simple hallway. It requires around 4000 iterations for the robot to learn better practices and reach the goal more frequently. When the training is done, the robot is moved to a test environment, which is otherwise similar to the standard one with the exception of a slanted floor, a ramp, in the beginning of the hallway. This proved to be an obstacle that the robot could not overcome without the help of sensor spoofing. Sensor spoofing in this case means inputting fake values to the robot's laser sensor, which is responsible for detecting obstacles around the robot.

The major target in this research was to transfer the previously learned data from the standard environment to the test environment and utilize sensor spoofing to help the robot overcome the slanted floor and eventually analyze if transfer learning helped the robot perform better. The performance can be compared by looking at the rewards received by the robot, since the robot receives highest rewards when reaching the goal location in the environment and negative rewards when crashing into walls. If transfer learning is beneficial for the robot, the robot should reach the goal point more frequently when making use of previously trained data from the standard environment and sensor spoofing in the test environment, compared to how it performs without them. This was also the result achieved. Even though the performance was not as good as it was without the ramp since without the ramp the robot reached the goal point every time after training around 200 episodes, the performance was better than it was without the trained model and sensor spoofing being used. As a result, transfer learning can be applied in virtual environments for mobile robots under certain restrictions. It can also be utilized in many other cases, this project is just one example.

The codes and files used for this project are available on GitHub at https://github.com/lperala/Transfer_learning_for_mobile_robots.

Keywords: machine learning, ROS, OpenAI, reinforcement learning, LiDAR, sensor spoofing

Kivikko V., Kyllönen M., Perälä L, Säily K. (2021) Transfer learning for mobile robots. Oulun yliopisto, Tietotekniikan tutkinto-ohjelma, 50 s.

TIIVISTELMÄ

Tässä projektissa oppimisen siirtäminen voidaan määrittää aiemmin opitun tiedon siirtämisenä uuteen ympäristöön ja sen hyödyntämisenä esteiden välttelyyn. Oppimisen siirtämisen toteutettavuutta tutkittiin tilanteessa, jossa robotille on annettu tehtävä navigoida käyttäjän määrittämään sijaintiin virtuaalisessa ympäristössä osumatta seiniin hyödyntäen vahvistavaa oppimista robotin opettamiseksi, joka tarkoittaa että robotti saa positiivisia palkkioita sen mukaan miten se liikkuu ympäristössä ja kuinka lähellä se on tavoitesijaintia. Tässä projektissa kaikki on tehty ja testattu simulaatiossa. Ensin robotti koulutetaan standardiympäristössä, joka on yksinkertainen käytävä. Robotti tarvitsee noin 4000 toistoa koulutusta, jotta se oppisi liikkumaan paremmin ja saavuttamaan tavoitesijaintinsa useammin. Kun koulutus on tehty, robotti siirretään testiympäristöön, joka on muuten samanlainen kuin standardiympäristö, mutta sisältää kaltevan rampin käytävän alussa. Tämä osottautui esteeksi, jonka yli robotti ei kyennyt liikkumaan ilman sensorin huijaamista. Sensorin huijaaminen tarkoitti tässä tapauksessa tekaistujen arvojen syöttämistä robotin lasersensorille, joka vastaa esteiden havaitsemisesta robotin ympärillä.

Suurin tavoite projektissa oli siirtää aiemmin opittu data standardiympäristöstä testiympäristöön ja hyödyntää sensorin huijaamista auttaakseen robottia ylittämään ramppi ja lopulta analysoida oliko oppimisen siirtämisestä hyötyä robotin suoriutumisen kannalta. Suoriutumista voitiin tarkastella vertaamalla robotin keräämiä palkkioita, koska robotti saa isoimmat palkkionsa saavuttaessaan tavoitesijaintinsa ympäristössä ja taas negatiivisia palkkioita, mikäli se törmää seinään. Jos oppimisen siirtäminen on hyödyllistä, se tarkottaisi että robotti saavuttaisi tavoitesijainnin useammin kun se hyödyntää aiemmin opittua dataa kuin jos se suoriutuisi ilman opittua dataa. Tämä oli myös tulos johon päädyttiin. Vaikka suoriutuminen ei ollut yhtä hyvää kuin ilman ramppia, koska ilman ramppia robotti saavutti tavoitteensa jo 200 koulutusepisodin jälkeen, suoriutuminen oli parempaa kuin se oli täysin ilman koulutusta ja sensorin huijaamista. Tuloksena, oppimisen siirtämistä voidaan hyödyntää virtuaalisissa ympäristöissä mobiileille roboteille tiettyjen rajoituksin. Sitä voidaan myös hyödyntää monissa muissakin tapauksissa, tämä projekti on vain yksi esimerkki.

Projektissa käytetyt tiedostot ovat saatavilla GitHubissa osoitteesta https://github.com/lperala/Transfer_learning_for_mobile_robots.

Avainsanat: koneoppiminen, ROS, vahvistava oppiminen, valotutka, sensorin hämääminen

TABLE OF CONTENTS

ABSTRACT
TIIVISTELMÄ
TABLE OF CONTENTS
FOREWORD
LIST OF ABBREVIATIONS AND SYMBOLS
1. INTRODUCTION 8
2. RELATED WORK 10
2.1. Transfer Learning 10
2.1.1. Transfer Reinforcement Learning 10
2.2. Mobile Robots 11
2.3. Robot Spoofing 11
2.3.1. Sensors 12
2.3.2. Spoofing Real Robots 12
2.4. Robot Operating System 12
3. DESIGN 14
3.1. First Steps and Goal 14
3.1.1. Tutorials and Projects 14
3.1.2. Initial Plans 14
3.2. Testing Environments 17
3.2.1. ROS Subscribers and Publishers 19
4. IMPLEMENTATION 20
4.1. Autonomous Navigation and Mapping 20
4.2. ROS Computation Graph 23
4.3. Reinforcement Learning 23
4.3.1. Reward Functions 24
4.4. Transferring the Learned Model 27
4.5. Detection of Obstacles 27
4.5.1. Examine the Actual Situation 27
4.5.2. Parameter Altering 28
4.6. Spoofing the LiDAR 28
5. EVALUATION 31
5.1. Evaluation Plan 31
5.2. Learning Parameters 32
5.3. Spoofing 32
5.4. Trained Vs. Untrained Models 33
5.4.1. Reward Function 2 33
5.5. Data-Analysis 38
6. DISCUSSION 40
6.1. Limitations 40
6.2. Simulation Vs. Real Robots 40
6.2.1. Spoofing the LiDAR 40
6.2.2. Differences in Robot Models 41
6.2.3. Different Approach 42

6.3. Transfer Learning 42
6.4. Future Work 43
7. CONCLUSIONS 44
8. REFERENCES 45
9. APPENDICES 48
9.1. Contributions 48
9.2. Terminal Commands 50

FOREWORD

This Bachelor's thesis and the project was done for a course Applied Computing Project I (ACP1), 521041A from the University of Oulu.

Oulu, June 7th, 2021

Ville Kivikko
Markus Kyllönen
Lassi Perälä
Katri Säily

LIST OF ABBREVIATIONS AND SYMBOLS

ROS	Robot Operating System
LiDAR	Light Detection And Ranging
SLAM	Simultaneous Localization and Mapping
Sarsa	State–action–reward–state–action
GUI	Graphical User Interface
VR	Virtual Reality

1. INTRODUCTION

Machine learning demands a lot of time and data. In standard case, every time the machine needs to be taught into a completely new task, the training needs to be started from the scratch. The feasibility of transfer learning has been under research for a while now to reduce the time and data needed when teaching a new, similar task for a machine which uses machine learning. This research is done to evaluate the effectiveness of transfer learning by trying to overcome an existing problem with mobile robots, when they are facing a slanting on the floor. The initial hypothesis is that transfer learning is not feasible as is, but sensor spoofing can have an effect on that.

To understand the concept of transfer learning, one must have a basic knowledge about what machine learning is. Even though the technology of machine learning can be applied into various number of subjects, the common factor in these subjects is that the machines which use machine learning need to be taught the task they are meant to perform. The machine learns from the data it process. One could describe machine learning as computational methods based on experience — and in this case experience means the data processed earlier, for example as a form of training sets labeled by human — to make accurate predictions or to improve performance [1].

In order to obtain a sufficient level of functionality, which means that the machine can do the task it has been taught into with an appropriate success rate for the task in question, the machine must have enough experience. In other words, it must have processed a sufficient amount of good-quality training data. Acquiring, gathering and labeling data, which is specific enough for the task in question but at the same time diverse enough to be able to perform the task in real-life scenarios where the input can vary, is very expensive. If the expenses and the amount of data, or even one of these can be brought down, the learning process can become cheaper or the required time can be shorter. Less time and money used is obviously desired, and this can also lead to better performance of the robot, if the otherwise saved time or money is still being used on the training process.

If the machine's task is changed significantly, it needs to be trained all over again with new data sets which are suitable for the new task. This is the problem transfer learning is trying to overcome by using the already existing knowledge to learn new, related tasks without the necessary routine of teaching the machine from the beginning every time a new task or a big change in the task is introduced [2].

What that means is that if the experience from that previously learned, related task can be used to help the learning of new tasks, the learning process gets much easier, and more complex tasks could be taught. Training in simulation rather than real robots is usually safer because of the need for multiple iterations and therefore risk of the real robot receiving damage when crashing.

Considering mobile robots, transfer learning methods would offer huge benefits in the training process. If the robot gets its experience in basic functions in a standardized environment, for example moving in a regulated, flat area without anything to interrupt the robot, what happens when the environment changes, for example, a blizzard starts? Most likely, the robot has not got much — if at all — experience from this kind of situations. If transfer learning methods would be successfully applied, the robot could have been taught for this kind of situation with relatively little effort by for example

sensor spoofing. This would save a huge amount of time and it would allow the robots to be taught how to react if something unforeseen occurs.

Facing a slanted floor is a widely-known problem occurring with basic mobile robots, as they think that they are facing a wall instead of an elevated plain which they actually could overcome. This is due to the LiDAR (Light Detection and Ranging) sensor, which is attached on the robot, receiving low values in the back of the robot when the robot is at an angle. In this project, the focus is on testing the feasibility of transfer learning combined with sensor spoofing, which in this case happens by feeding modified values to the LiDAR sensor, to make it more effective.

The testing is first done by seeing if the model learned when training in the standard environment, which in this paper means the flat area built for the robot, helps the robot get over the slanted floor in the test environment, which in this paper means the area with the slanted floor built for the robot. If not, overcoming this problem would require spoofing the LiDAR sensor to make the robot ignore the slanting and continue exploring. If the spoofing helps the robot to overcome the slanting, which it considers as a wall without spoofing, the spoofing would be beneficial for the robot, and the transfer learning could also be feasible. The initial hypothesis for this is that the transfer learning is not feasible as in the training from the standard environment as such is not enough for the robot to overcome the slanted floor, but with sensor spoofing implemented it can be beneficial.

2. RELATED WORK

2.1. Transfer Learning

Transfer learning can be considered as an improved version of machine learning in the way that whereas machine learning methods can be used to learn isolated tasks from square one [3, 2], transfer learning methods draw information from previously learned source tasks to learn the new target task [4, 5], which is presented in Fig. 1. This way previous knowledge can be used to learn new tasks faster [2, 4]. It must be noted that transfer can also occur as negative transfer if it does not enhance performance but instead decreases it [2], which is something that needs to be avoided. Recent studies have showed that in the field of robotics transfer learning can provide for example increased performance in collision localization [6] and even enable the transfer of human-like skillsets to robots with skill transfer learning [7].

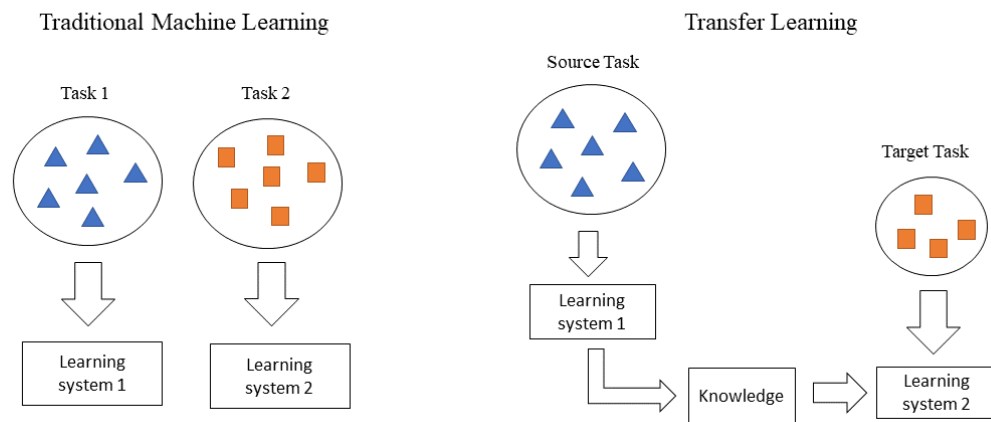


Figure 1. The differences between machine and transfer learning methods.

In this project the role of transfer learning is to take the knowledge that has been teached to the robot in one standard environment and utilize this learned information in another test environment. In this case, transfer learning is combined with reinforcement learning.

2.1.1. Transfer Reinforcement Learning

The basis of reinforcement learning lies in Markov's decision process, which means that the learning agent tries to find optimal actions that produces the best cumulative reward [2, 4]. Transfer reinforcement learning method has been applied to for example autonomous driving, where training is done in simulation and then the model is transferred to autonomous cars [8], which in theory is quite similar to the goal of this project with the exception that in this project everything is done in simulation. The same method has also been applied to robotics, for example in a study where reinforcement learning was used to train a robot to hit a ball as far as possible while

being seated [4]. The studies mentioned previously also state that reinforcement learning can take up a lot of time because of the large amount of training iterations needed for sufficient learning to occur, which is something that transfer learning could help with. This project focuses on transferring reinforcement learning in ROS (Robot Operating System) framework. In a study by Zamora et al. [9] reinforcement learning is used in ROS framework to train robots to avoid obstacles but that study along with some others [10] focuses more on the reinforcement learning aspect and does not implement transfer learning, which is one of the focus points in this project.

2.2. Mobile Robots

Thanks to modern technology, there are several methods for machines to detect different objects. Mobile robots can recognize objects, move around safely even in challenging surroundings, locate and plan paths for themselves, and they can understand natural speech.[11]. One example of object detection technologies is the 3D LiDAR method. This type of technology is typically used in advanced driver assistance systems and in autonomous driving vehicles [12]. As a practical example, mobile robots can be taught to detect humans using the 3D LiDAR method [13]. Taking human detection as an example, laser sensors and cameras are the most common technologies for tracking humans [14].

Multiple programming languages have been used in mobile robot software. A few well-known examples are Python, C, and C++. C and C++ are the most used when addressing the problem of satisfying real-time capability, Python language is used massively for mobile robots and artificial intelligence because of its simplicity [15]. There is a lot of information available to learn more about designing, simulating, programming, and prototyping an interactive autonomous mobile robot from scratch using Python [16].

2.3. Robot Spoofing

Robot Spoofing is the act of trying to distract or force a robot to malfunction. One of the simplest ways to spoof a robot is robot kidnapping [17]. If a mobile robot knows it's surroundings and it works based on that, the robot could be picked up and moved to an entirely different location. As the surroundings are now different, the robot might not be able to locate itself which may cause malfunctions or even crashes. Robot Spoofing does not necessarily need physical contact. The spoofing could be done with the robot's GPS [18] or GNSS [19] components, sensors [20] or even VRR (Virtual Reality for robots) [21].

As robots become more common with the digitalization of the world, the spoofing should be taken more seriously. Especially on robots that will be used in safety critical roles. For example a security robot could be spoofed to the state where it is not able to report to other surveillance units [21]. These states of malfunction are obviously huge security threats and could be abused by criminal activity for example.

2.3.1. Sensors

The sensor that is spoofed in this project is the 3D LiDAR sensor, which is responsible for detecting the surroundings of the robot. The LiDAR can sense a full 360 degrees range around the robot. The use of LiDAR in a simulation can be seen well in the study by Zamora et al. [9]. Another sensor that could very easily be spoofed is the wheel odometry sensor, that fetches values of the robot's position as well as orientation. Also the robot's camera is one sensor that could be spoofed. An example of this can be found in the study by Davidson et al. [22] where they mimic a spoofing attack to the camera by a laser to see how much the camera can be affected by an outside source. Other additional sensors can be implemented to a TurtleBot as well, for example an ultrasonic sensor. When spoofing, the sensors need to be affected, in this project this happens by running a script that publishes fake values to the sensor, spoofing it to act as the user wants it to.

2.3.2. Spoofing Real Robots

The study by Ingabire et al. [23] shows a great example of how a TurtleBot can be programmed to avoid obstacles in laboratory settings instead of simulation. In a real setting some type of additional device, for example Raspberry Pi, would have to be used to establish a connection between the computer, the robot and the sensors. The spoofing itself would function the same as mentioned in subsection 2.3.1 Sensors.

2.4. Robot Operating System

ROS (Robot Operating System) is a framework that is used to make software for robots. It is an open source collection that aims to make robotic programming simpler by enabling collaboration between programmers [24]. This way people can work together and develop their own project by building on top of some existing framework. ROS was invented to ease the nearly impossible task of developing software for robots from the scratch. Even the tasks that seem so simple to humans, are usually very complicated when trying to program a robot to perform them. For example, if a robot has a task to get the newspaper, it would first have to understand what it is asked to do, then have some kind of navigation to the door, identify the newspaper, pick it up and navigate its way back. All of these actions would have many scenarios where they can go wrong. This task can be much easier to achieve, if someone has already come up with a program to perform one of these steps, for example the navigation. This is where an open source software benefits everyone. [25]

ROS is composed of many different elements, which include drivers to control motors and collect data from the sensors, constantly growing collection of algorithms for various tasks such as building maps, the computational frameworks needed, tools for visualizing the robot and to help with debugging and also resources such as a wiki for documentation of ROS's functionalities [25]. To be able to work with ROS it is recommended that one has an Ubuntu Linux environment and some programming experience, preferably with Python, but other languages such as C++ and Lisp are also

used. ROS has three levels of concepts. Filesystem level consists of packages and their manifests, metapackages, repositories, messages and services. Computational Graph is "the peer-to-peer network of ROS processes that are processing data together" [26]. The final level, Community level consist of resources such as repositories and distributions that make collaboration between separate groups possible [26].

3. DESIGN

3.1. First Steps and Goal

The purpose of the project is to train a TurtleBot to find its way through a map to the desired point which gets defined beforehand by x and y coordinates, avoiding walls at the same time. Initially the training will be done in a standard environment with a flat floor. After that, the robot will be tested in a test environment that provides more challenge to the robot, which in this case is a slanted floor. The goal is to make the robot learn to move successfully in the test environment by training it only in the standard environment and by modulating the sensor input. The LiDAR sensor is used for this task, which is installed on TurtleBot as well as most other mobile robots. The project is done in simulation, however, one focus of the project is that there should be nothing that would block this from working with real robots as well.

3.1.1. Tutorials and Projects

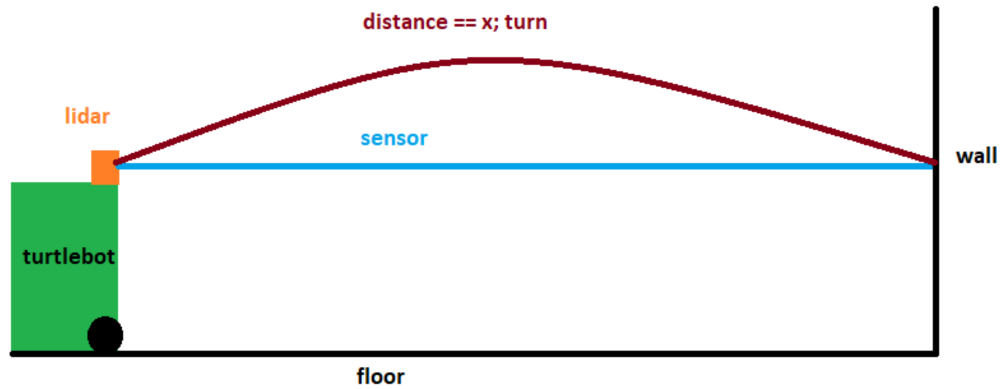
As the first phase of the project, the simulation environment is built using ROS according to project instructors' tutorial [27]. Also, other tutorials have been looked up [28] that helped with the set up and learning the basics of ROS.

Research has been done regarding setting up test environments and programming mobile robots. Since ROS is an open source framework, many tutorials are available for others to build on top of. An existing Reinforcement Learning implementation of OpenAI ROS [29] was the first basis of the project. This tutorial also had a further example on how to make a TurtleBot robot learn to move through a maze without colliding [30]. The one part of the work is to utilize the tutorials found online, build testing environments to be used with them, take a look on the sensor inputs and how they work, and then to test the robot in the test environments and see if spoofing the LiDAR helps the robot to overcome the problem.

Some other papers regarding the subject of this thesis are introducing simulation environments for mobile robots, where ROS and Gazebo are used to set up a test environment for mobile robots to implement the simulation process in the real robot without modifying the code [31]. Also, this study presents different ROS tasks, such as autonomous navigation and 3D mapping.

3.1.2. Initial Plans

The first thoughts were to use the 3D LiDAR sensor found in TurtleBot to ensure that the robot would change its direction when detecting a wall. The problem with this was that if the floors are slanted in the test environment, the robot might see them as obstacles to avoid instead of moving through them. This is illustrated in the Fig. 2, where the range detecting from LiDAR is presented in low fidelity drawings. The height of where the LiDAR is located on the robot has a big factor with the distance it measures when facing a situation like this.



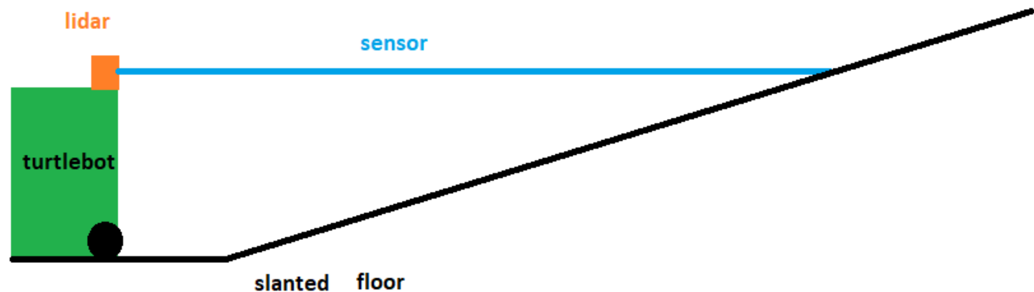
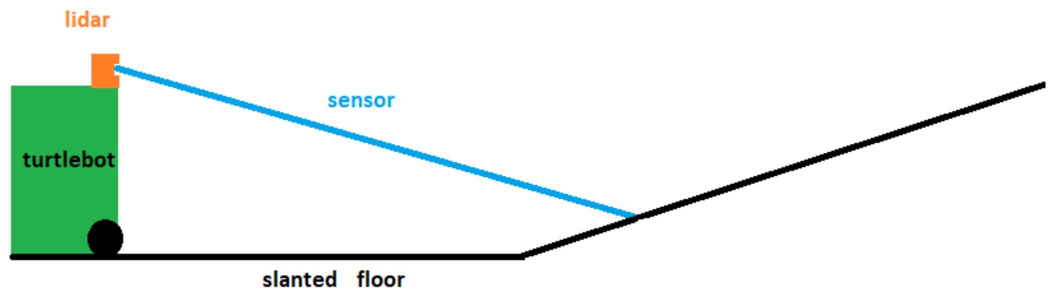


Figure 2. The problem faced when using the LiDAR sensor to detect obstacles.

To solve the TurtleBot's problem of detecting a slanted floor as a wall, the idea of tilting the sensor so that it would not be in a horizontal position and it would face downwards instead was introduced. This way the robot could be programmed to detect slanted floors but not consider them as walls to avoid. This could also be used to help the robot to detect sudden drops in the floor, as pictured in Fig. 3, where the range detection is presented when facing a slanting and a drop on the floor, and the LiDAR is not in line with the floor.

When doing more research, this plan was dropped because while it could have been use to solve the recognition problem of a slanted floor or a sudden drops in the floor, it would have not been anything else than that, and it would have not even use transfer learning. The focus was changed on observing the LiDAR readings when the robot faces a slanted floor, and what could be done to them to overcome the problem. With this as the target of the project, transfer learning and sensor spoofing could be applied while trying to overcome the problem. This way the evaluating of the feasibility of



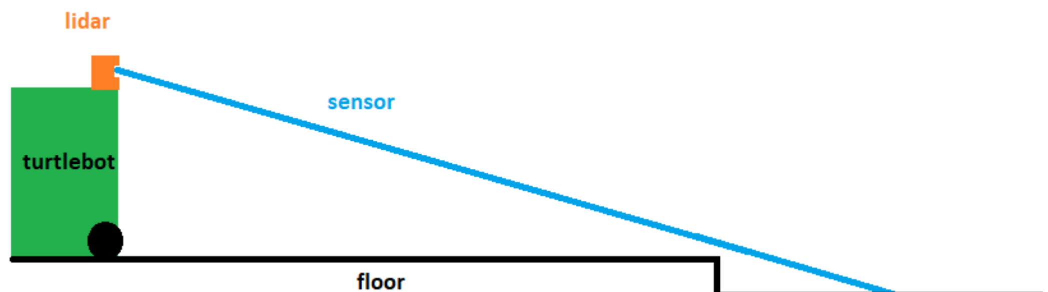


Figure 3. Detecting changes in elevation.

transfer learning with sensor spoofing could really be done, and perhaps some kind of methodology which could be used on various kind of issues could be found.

The first idea about spoofing the LiDAR was to feed such values to the LiDAR sensor that mimic the ones it would receive when the robot faces a ramp. However, it was figured that a simpler way the LiDAR could be spoofed was to make the robot believe that the floor would be flat when it actually is slanted. The data from the robot's LiDAR would have to be captured, and then these readings would have to be modified by inputting that kind of data back to the LiDAR what it reads when there are no slanting at all. This way the robot could think that it is in an environment without a ramp, even though it has a ramp in front of it.

This could be very efficient solution to the problem, especially when previously learned model is applied for the robot. The model would come from a similar environment, but without any ramps. And when the robot would actually face a ramp, the LiDAR would get spoofed to show similar values it reads when there are no ramp.

Spoofing the sensor this way, the existing problem with the slanting might be defeated. This solution could also be applied to different problems existing with mobile robots, and perhaps other machines as well, if they are using machine learning.

3.2. Testing Environments

To be able to witness the unwanted behaviour of the robot when it comes across with a slanted floor, two slightly different testing environments had to be created. The environments needed to be otherwise similar, but one of the environments has a ramp or a slanted floor included, which messes with the robot's sensors causing this unwanted behaviour. These two simple areas imitate a hallway in a house. The difference in these environments is the ramp existing in one of the environments, while the other one is completely flat. With these environments, it could actually be seen what the robot sees in each of these cases, and also training the robot could be done in these areas, and then compare the results afterward.

Simple hallways are created by using Blender, which is a well-known, open-source product for 3D design. Multiple versions of the environment with a slanted floor had to be created because there was a problem where the robot was not able to travel the ramp-up due to a lack of friction. However, the problem is beaten by changing the angle of the ramp and altering the TurtleBot's parameters regarding the friction on its wheels. Also, a very important step was to merge the objects in Blender, so that the ramp and the floor did not have a gap between them, where the robot could get stuck. The outcome of the simple hallways with a tight turn is presented in Fig. 4.

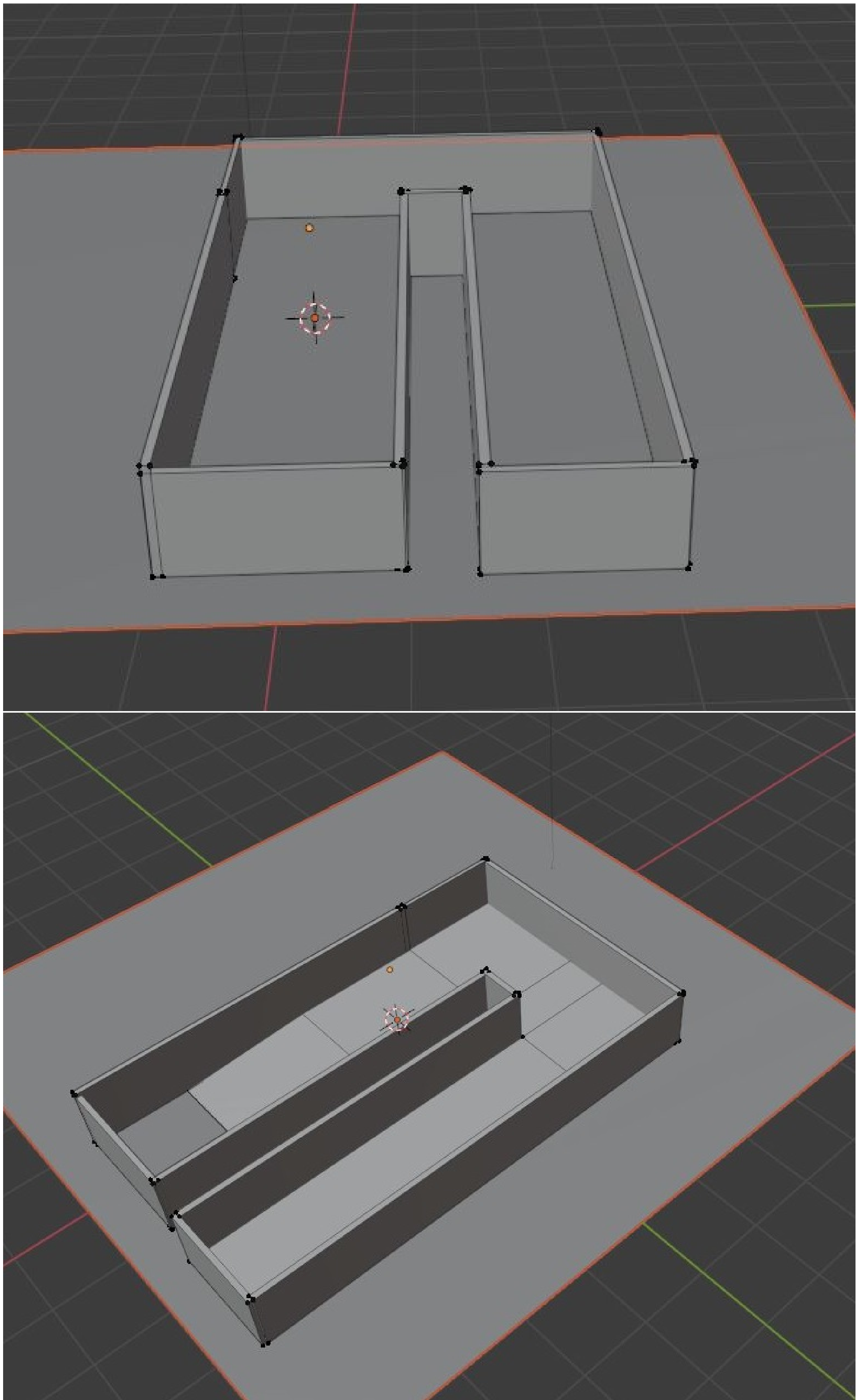


Figure 4. Two environments created in Blender. One with a flat floor and the other with a slanted one.

3.2.1. ROS Subscribers and Publishers

In ROS the nodes communicate by publishing and subscribing to topics. To receive the information published by the topics, such as /scan, which publishes the LiDAR values that the TurtleBot gets, setting up a subscriber is needed. Then again, to spoof the sensors and input the modified data, publishing information into a topic and replacing the current data it is receiving had to be done.

To get an impression of what type of readings the sensors get when facing a problem like slanted floors, sensor data needs to be plotted in real-time. With the help of an tutorial on LaserScanner data [32] and slight modifications to the code to implement Python Matplotlib's real-time plotting, LiDAR data can be presented in constantly updating plot. The results from this can be seen in Fig. 5. This implementation can also be easily applied to other sensors such as the odometer, which controls the wheels of the robot.

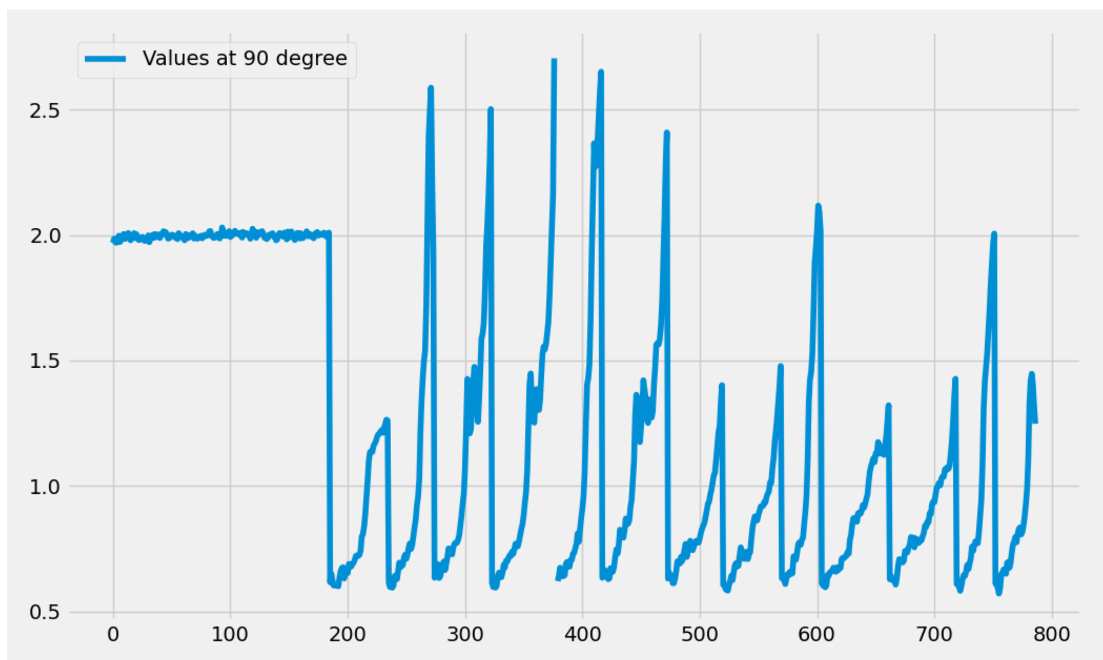


Figure 5. Data from /scan topic in training session. X-axis represents the iterations from beginning the real-time plotting and y-axis portrays the /scan topic's ranges value [180], which is 90 degrees.

4. IMPLEMENTATION

4.1. Autonomous Navigation and Mapping

Once the standard and test environments got created, the next step was to make the robot navigate autonomously around the environment. While navigating around the environment, the robot should be able to detect obstacles by using its 3D LiDAR sensor. Illustrating the functionality of TurtleBot in the environment, mapping is used from a technique called SLAM (Simultaneous Localization and Mapping) [33]. With the mapping feature, TurtleBot builds a map for the robot from the environment it is being used in [28]. By building this map, the robot's limits can be seen as to where it can move around safely and where the LiDAR sensor detects obstacles. This map can be visualized with a program named Rviz.

After spawning the TurtleBot to the desired location and starting the autonomous navigation in a flat floor environment, the robot starts to generate the map based on the LiDAR data. The map the TurtleBot created in standard environment is shown in Fig. 6. As seen in the picture, the generated map is a fairly accurate estimate of the actual environment.

When performing the same job in the test environment with the slanted floor, a new issue regarding mapping appears, which is shown in Fig. 7. The drawn map image is not as identical to the original map as it was in the test executed in the flat floor environment. When the robot is not yet on the slanted floor, it detects the ramp in front of it as a wall. If the robot is forced to move up the ramp, the laser data from the LiDAR detects the floor behind it as a wall. This leads to a very messy map, which causes issues for the robot, when determining where it is safe to move without colliding to the walls.

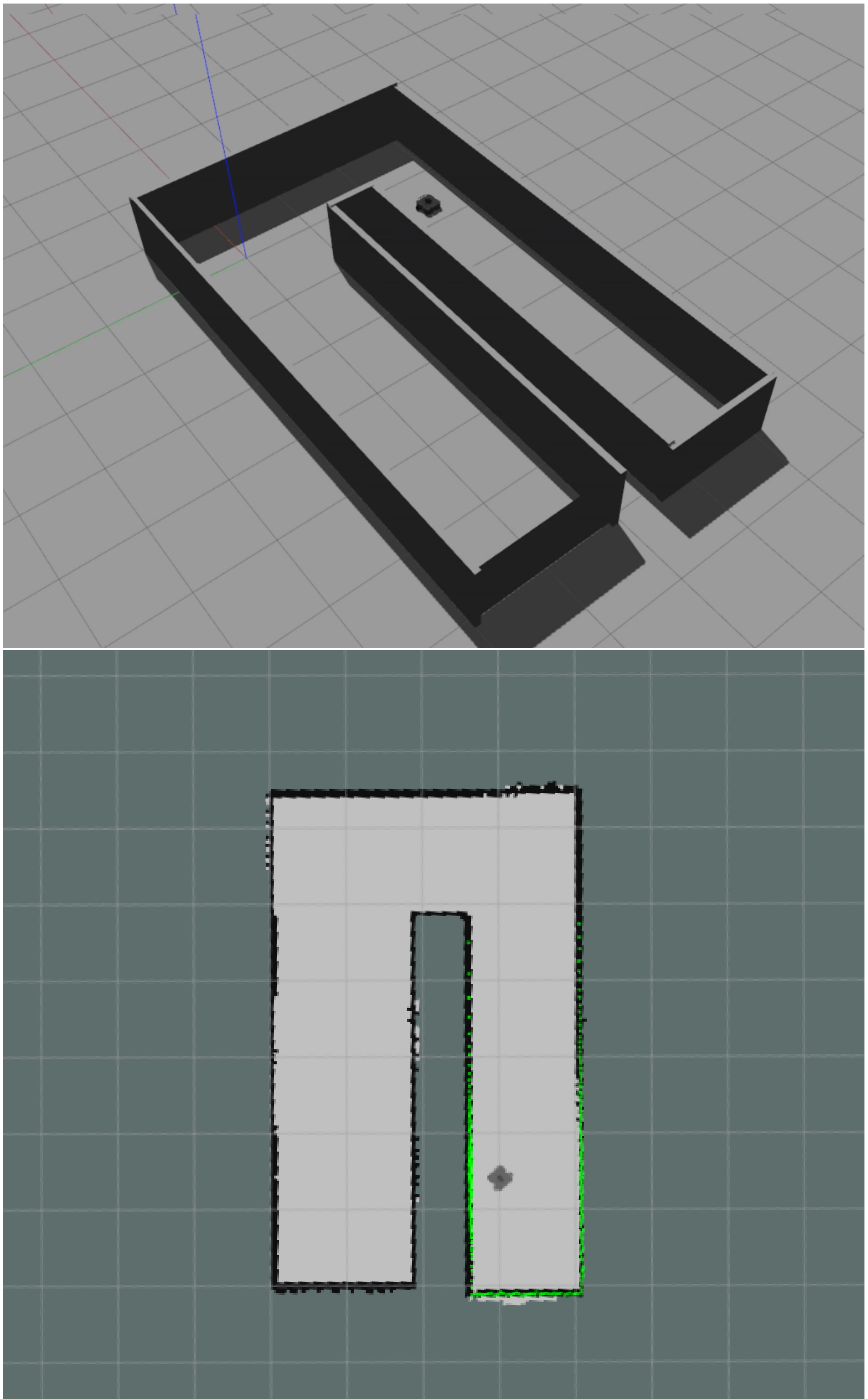


Figure 6. TurtleBot3 performing an area sweeping in an environment with a flat floor. The second picture presents the map it created.

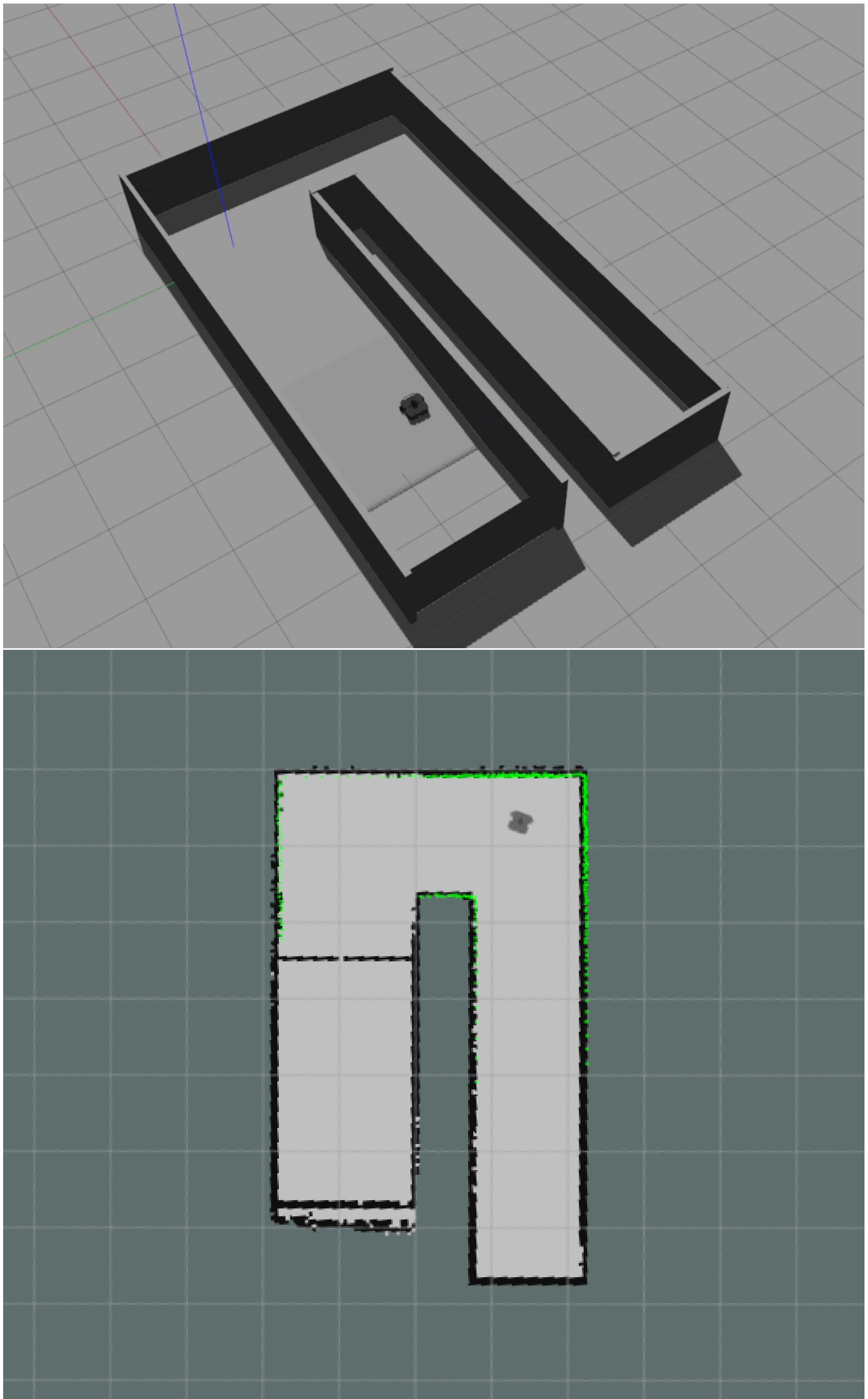


Figure 7. TurtleBot3 performing an area sweeping in an environment with a slanted floor. The second picture presents the map it created.

4.2. ROS Computation Graph

In order to get a better understanding of ROS functioning, the ROS computation graph can be visualized by using a GUI plugin rqt_graph [34]. Fig. 8 represents a graph when performing TurtleBot3 training process in a Gazebo environment, while at the same time visualizing robot's sensor data. The environment, in which the training is performed, is marked as "turtlebot3_world". Controlling the translational and rotational speed of the robot unit can be seen from the node "/cmd_vel". The speed can be marked either in m/s or rad/s. When the robot has a certain speed, it can be visualized in a node "/gazebo", which has multiple outputs.

The attitude of the robot based on the acceleration and gyro sensor is included in a node "/imu". Odometry information of the robot based on the encoder and IMU is located in a node "/odom". The scan values of the LiDAR, which is mounted on the robot, are in a node "/scan". "/joint_states" -node is the state of a set of torque-controlled joints, which send information through "/robot_state_publisher" to node "/tf", which contains the coordinate transformation. Continuing from there, the data is sent to node "n__rviz", where the data from the LiDAR can be visualized. This is one example of ROS Computation Graph, how a TurtleBot3 can perform learning in a gazebo environment while visualizing the LiDAR data in RViz [35].

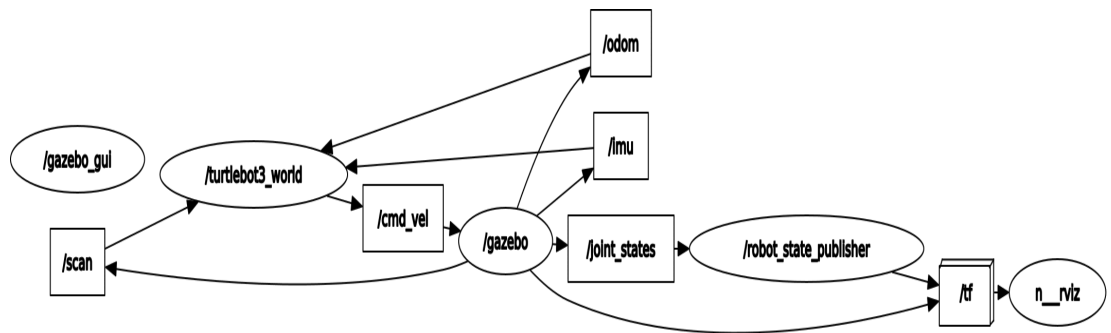


Figure 8. ROS computation graph, when performing TurtleBot3 training in Gazebo and RViz.

4.3. Reinforcement Learning

For this work to include transfer learning, a need of applying some type of reinforcement learning method was present. A decision was made to use OpenAI [29], which implements a Q-learning algorithm. The Q-learning algorithm [36] uses the following formula:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + 1 + \gamma \max Q(s+1,a) - Q(s,a) \right] \tag{1}$$

where s is the state, a is the action and r is the reward.

While training, the robot receives positive rewards for succeeding in tasks and going forwards without crashing and negative rewards for crashing into the walls. This way the robot is encouraged and taught to move without crashing into anything. Learning

can be affected by changing the parameters such as α, γ and ϵ in the yaml configuration file (located in my_turtlebot3_openai_qlearn_params.yaml).

α is used to define the learning rate of the algorithm. γ values are used to define whether the robot is looking for higher rewards immediately or in the long run. ϵ parameter is used to tell the robot whether it should exploit the learnt knowledge or to explore for new knowledge. Usually, and in this case as well, the ϵ parameter starts from a number close to 1, and then is discounted by a discount factor when the iterations go by.

For testing purposes, the Sarsa (State–action–reward–state–action)-algorithm [9] was implemented, which follows the formula of:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + 1 + \gamma Q(s+1, a+1) - Q(s,a)\right]] \tag{2}$$

The difference being that the actions with Sarsa follow the originally selected action whereas the Q-learning method tries to find the action that would give the maximum reward. Both of these algorithms fall under reinforcement learning, but Q-learning algorithm is considered off-policy and Sarsa on-policy. This means that Sarsa tries to improve the original policy that it used to choose its first action whereas Q-learning aims to improve the policy that is not necessarily original but the most optimal [9].

4.3.1. Reward Functions

The original reward functions were highly based on the actions that the TurtleBot makes. For example turning or going forwards were the only factors that gave rewards for the robot. This is definitely not optimal for this case, so slightly more complex reward functions needed to be created, which are more suitable for this project.

The task environment's reward function was modified into two different tailor-made functions that work on the environments built for this project. Both of the rewarding systems are highly based around reaching a desired position in the map. The closer the robot gets to the desired point, the higher rewards it receives. The robot will no longer receive the same points for it's actions as it used to, but some reward gets given for correct actions.

In the first reward function the rewarding is highly based around the comparison of the robot's distance to the desired point in the current step and in the previous step. If the robot's distance to the desired point is smaller in the current step than it was in the previous step, the robot gets rewarded. The robot also gets higher rewards based on how efficiently it moves towards the desired point. For example moving straight towards the desired position gives the robot higher rewards than moving slightly towards the desired point. If the robot reaches the desired point, it receives additional 50 points and the current episode ends. If the robot crashes without reaching the desired point, it receives negative 50 points. Fig. 9 represents the algorithm in 4000 episodes.

First reward function:

```
distance_difference = distance_from_des_point - previous_distance_from_des_point

if not done:

    if distance_difference < 0:
        reward += 5

    else:
        if distance_difference < -0.06:
            reward += 3

        elif distance_difference > -0.06 and distance_difference < -0.03:
            reward += 2

        elif distance_difference > -0.03 and distance_difference < -0.01:
            reward += 1

else:
    if is_in_desired_position(current_position):
        reward = 50
    else:
        reward = -1*50
```

The main purpose of the second reward function was to make the robot travel to the desired point as directly as possible. This function is custom for this project's environments, but it can be applied to other environments as well, if slight changes on the coordinates are made. If the robot is moving forwards, the reward function gives a reward of 5, and turning gives a small negative reward with a value of -1. This was done to force the robot into avoiding unnecessary turnings of the robot.

To avoid crashing to the walls, the robot gets small negative rewards when it is moving at locations which are too close to the walls. Also, if the robot goes in the wrong direction directly from the start, it gets punished with negative rewards. When the episode ends, the robot will get rewards, based on the distance between the desired point and the current location of the robot. The closer the robot is to the desired point, the bigger the reward is. Figure 10 represents the second reward function in 4000 episodes. As can be seen from that figure, a clear learning for the robot has been acquired with this reward function.

Second reward function:

```
if not done:
    if current_position.y >= -0.2 and current_position.y <= 0.3 and current_position.x > 0.0:
        if self.last_action == "FORWARDS":
            reward = 5
        else:
            reward = -1

    elif current_position.y < -0.2 or current_position.y > 0.3:
        reward = -2

    elif current_position.x < 0.0:
        reward = -20

else:
    if current_position.y > -0.9 and current_position.x < 1.5:
        reward = -200

    elif current_position.y > -0.9 and current_position.x < 4.0:
        reward = int(current_position.x * 40) - 100

    elif current_position.y > -0.9 and current_position.x >= 4.0:
        reward = 200

    elif current_position.y <= -0.9:
        reward = 200
```

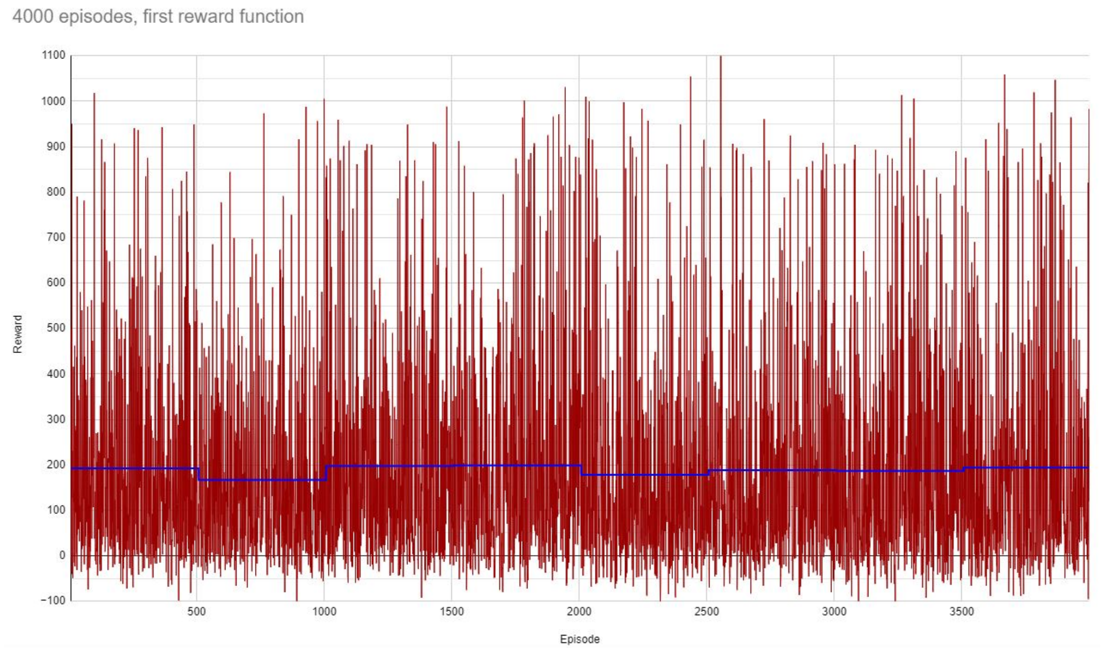


Figure 9. The first tailor-made reward function used in training with a length of 4000 episodes.

After comparing the results of the reward functions used in these tests, the second reward function got chosen to be used in this project, as it produced better results. The improving can be visually seen from the graph, since the means from the second reward function are clearly rising, as the mean rises from the negative side on to the positive side.

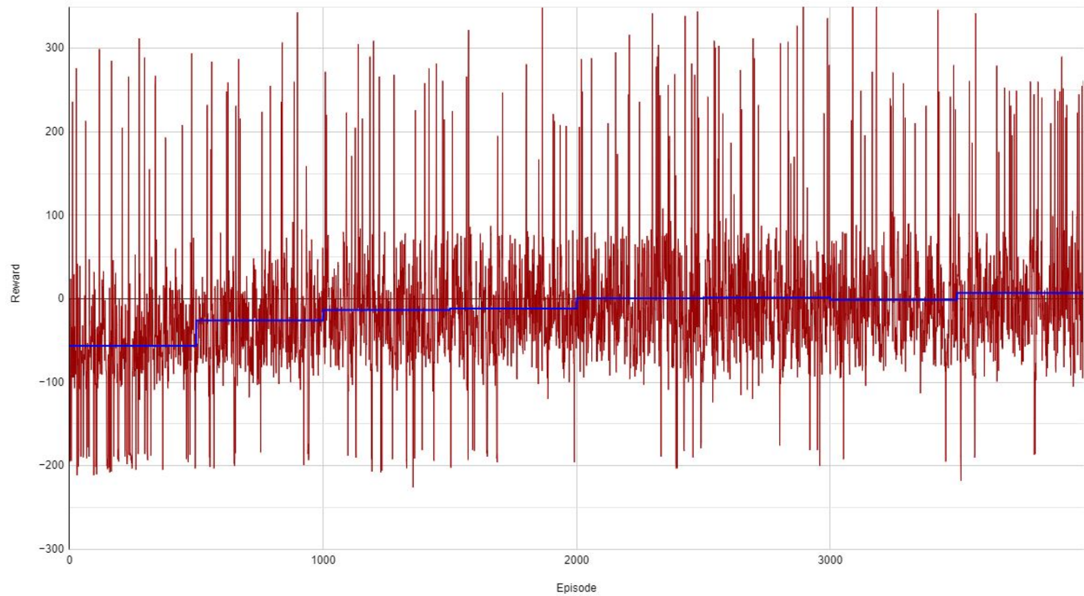


Figure 10. The second tailor-made reward function used in training with a length of 4000 episodes.

4.4. Transferring the Learned Model

When the robot has reached enough training iterations to learn the desired task, reaching the user-defined goal location in this case, the learned model should be transferred to test it in the environment with a slanted floor. Python package called Numpy was used to save and the load the states of the Q-learning.

If the robot has a learned model behind it, it already has some kind of experience it can use when trying to perform its task in the current environment. If there are some similarity between the tasks, and the robot understands it, it should, in theory, take less time when learning the new task. This is due to the previous experience the robot has, so the robot does not have to start the training from the scratch. This is a huge part of the transfer learning, as the name suggests.

4.5. Detection of Obstacles

4.5.1. Examine the Actual Situation

To get an impression of what the robot sees when facing a slanted floor, the situation when the robot crashes when it is entering the ramp needed to be visualized somehow. Only knowledge so far was that the LiDAR announced that the robot has crashed to an obstacle, because the LiDAR got readings where the range was less than 0.20 meters. The problem with rising up the ramp was confirmed by looking at the state that the LiDAR sees when it is entering the ramp. And it is indeed thinking that it is crashing.

As you can see in the Fig. 11, where the data from the robot's LiDAR is visualized, the robot thinks that it is facing a wall behind it while it actually is not, it is just trying to rise the ramp up.

At this point, also the observation of the learning algorithm was necessary. It thinks the same; the robot is too close to a wall, as seen in the Fig. 12 which represents the distance from the robot on to non-existing walls, when it actually is not. The crashing results tells that the non-existing wall is appearing to be 0.146 meters away from the robot, which is under the initial value of 0.2 meters. If this was done in the standard environment, which does not have the slanted floor on it, the robot would not be detecting any non-existing walls.

4.5.2. Parameter Altering

To mimic that the floor is flat even though there is the slanted floor, the parameters affiliated with the learning algorithm were altered by changing the threshold value of wall detection from 0.2 meters to 0.05 meters. This value defined what distance to the wall is concidered as crashing. With this change the robot still sees the non-existing wall, but it does not think that it is too close to it. It does not care about the non-existing wall anymore. This mimics the situation where the LiDAR has been spoofed into thinking that there is no ramp or anything that would affect the robot's movements.

The results were successful, as the robot did not think that there were anything that would stop it when rising up the ramp, so it got over it successfully. Even though, in this tentative test, the threshold value for detecting the walls was dropped down, the robot still recognized the actual walls when it crashed to them due to the fact that the robot can detect when it gets stuck, but it ignored the non-existing walls. Even though this change in the parameter affects little for the robots behaviour in this test, which is by going closer to the walls before it thinks its too close to it, it would not be the case if the spoofing would be done only when the robot enters and is on the slanted floor. The robot would go over the ramp and then keep doing its task normally. There is a chance that the robot would not cover the area fully, because of the change in the odometer while rising up the ramp, compared to the situation that the ramp does not exist. But that could possibly be tweaked out by using more sophisticated learning algorithms.

Even though that parameter altering is not actually sensor spoofing because the data from the sensor is not being modified in any way, and it would only be a solution for this specific problem instead of a general solution which could be applied for different issues existing with mobile robots, it provided valuable information for this project. This pointed the focus to the direction where the LiDAR values could be accurately imitated from the standard environment to be used in the environment with the slanted floor, the robot should not, in theory, have any problems on doing its task.

4.6. Spoofing the LiDAR

To be able to actually spoof the sensor, there was a need to capture the data from LiDAR, then modify it, and then let it pass to the robot. To reach this, a Python script was created which has subscriber for the LiDAR data. After the subscription, which

provides the access to the data to be used basically any way wanted, the data gets modified to show bigger distance on the places where the robot thinks the non-existing walls exists. In this case, the bigger distances for the data was added when the robot was on the ramp. The fake walls which the robot sees without sensor spoofing can be visualized in Figure 11.

To get the information when to start the spoofing, the spoofing script needed also a subscriber for the TurtleBot's odometer sensors. With that, an access was acquired to the coordinates of the TurtleBot, as well as the rotational information from its X-, Y- and Z-axes.

The script's triggering was implemented in two different ways. A more general implementation was to look at the tilting of the TurtleBot by checking the odometry sensor data. This means that when the TurtleBot is not in line with the floor, the spoofing starts. However, when testing this implementation, the TurtleBot still crashed multiple times in the beginning of the ramp, perhaps due to the latency stemming from the execution of the script. To overcome this problem, the implementation where the start of the spoofing gets triggered based on the X and Y coordinates of the TurtleBot was taken in use. This solution is more tailored to the case in this project due to the coordinates obviously depending on the location of the map and the location of the ramp in the map. With this implementation, better performance was reached, as the TurtleBot now had fewer crashes, due to the fact that the start of the spoofing could be started a little before when the robot actually hits the ramp, so the delays occurring with the execution of the script did not have that big of an effect anymore on the robot's behaviour.

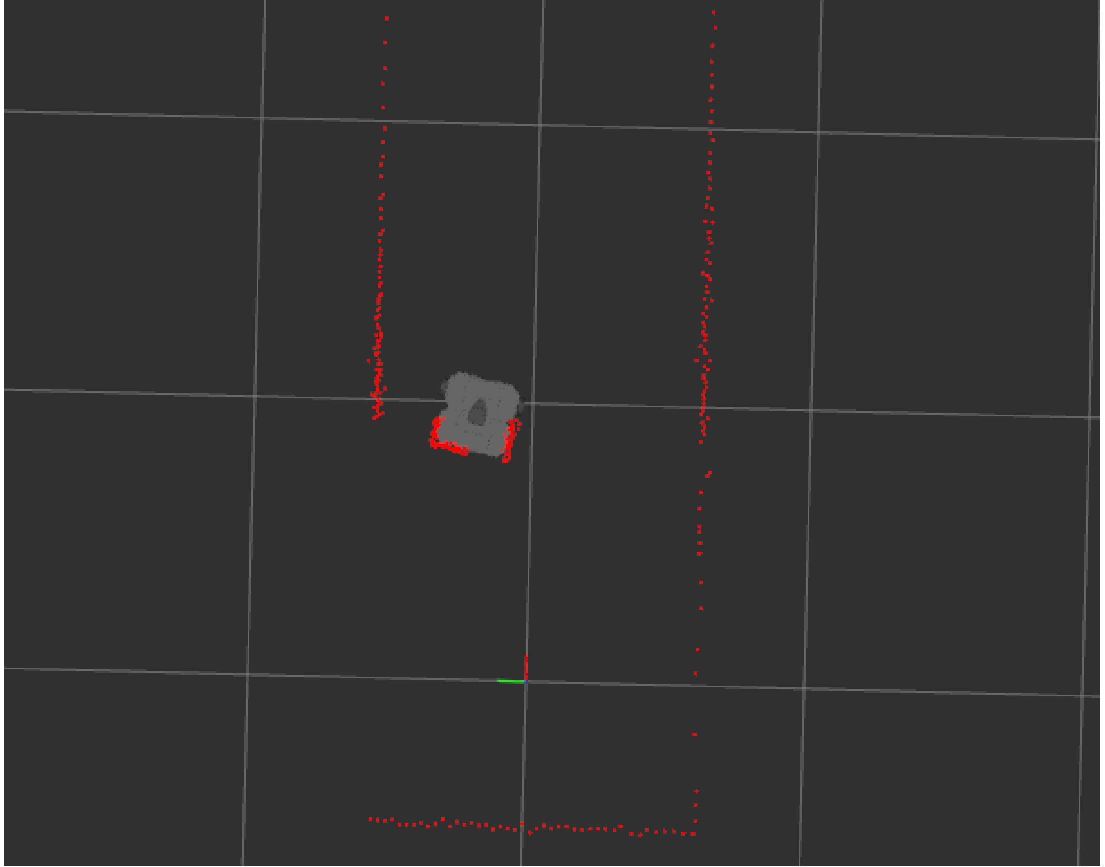


Figure 11. What LiDAR sees when the robot is starting to rise up the ramp. The markings on red are what the LiDAR sees. The red patterns touching to the robot are the non-existing walls.

```
WARN] [1616931285.077591, 8.701000]: ############### END Step=>41
WARN] [1616931285.078612, 8.701000]: ############### Start Step=>42
ERROR] [1616931285.284780, 8.903000]: done Validation >>> item=0.14639836549758
91< 0.2
ERROR] [1616931285.286144, 8.903000]: TurtleBot2 is Too Close to wall==>
```

Figure 12. The distance for the non-existing wall according to the learning algorithm.

5. EVALUATION

5.1. Evaluation Plan

The evaluation would have the most realistic and accurate outcome if it could be done with a real robot in a real environment. That way it could have been actually possible to observe how the robot behaves in the real life scenario. The plan was to test the transferring of the previously learned model. In the case of simulation, the evaluation of the learning algorithms can be done by comparing the graph constructed from the rewards given to the robot. The better the robot performs in its task, the higher the reward. By calculating the reward averages for example for 400 episodes at a time from the learning process, it can be seen if the robot is learning anything.

The best way to notice the improvement in rewards was to collect a batch of episodes in the beginning and in the end of the training sessions and then comparing the averages of the rewards on the batch. For example, training the robot for 4000 episodes and comparing the average rewards of the first 400 episodes and the last 400 episodes. If the average of the rewards is higher in the batch of the last 400 episodes, the robot has learned something. Of course, when there still are some randomness in the process, because epsilon value never goes to zero, there will also be some divergence in the averages. But when comparing the first episodes to the last ones, especially when as large batch as 400 episodes is being used, the average should definitely be higher regardless of the little randomness still existing in the learning process.

Evaluation plan is concentrated on comparing the robot's performance in the environment which has the slanted floor between these two cases:

1. The robot has not been taught anything before entering the test environment.
2. The robot has been trained in the environment with a flat floor.

If the robot would have a better performance in case number two, it would be beneficial to teach the robot in a standard environment. However, if the robot does not manage to overcome the ramp even with the training done in the standard environment, then comparing this situation to the case without training at all would be pointless for this project.

If this is the case, the evaluation will be done by comparing the following cases in the environment with the slanted floor:

1. The robot has been trained in the environment with a flat floor, and then moved to the environment with the ramp.
2. The robot has been trained in the environment with a flat floor, then moved to the environment with the ramp, and the LiDAR is spoofed to discard the ramp.

On these evaluation runs, the learning parameters of the robot are tweaked, so that the robot values instant, big rewards, and it has low chance to explore new things in the environment it is located.

α, which is the learning rate, remains unchanged, but the γ value is reduced to 0.1, so the robot tries to reach big rewards instantly. ϵ value is 0.05, which is also the lowest value it reaches in the actual training process. So, the robot takes random actions with

a five percent chance, and it desires big rewards instantly, but it also still continuously learns something from the episodes it runs. The episodes ran in these cases are not saved for the robot, so it has the same previous knowledge no matter the order these tests get ran.

The focus was on spoofing the sensor the way the robot could rise the ramp up. If the robot gets the ramp up while the LiDAR is being spoofed and it has the previously learned model from the flat environment behind it, and the spoofing has not dramatically changed the way the robot works, the spoofing has helped the robot to overcome its previous obstacles, so the spoofing has been successful and it is very beneficial for the robot. This would provide an important information that the spoofing of the sensors could be beneficial also in real life scenarios, also when applied it for any other even a slightly similar situation.

5.2. Learning Parameters

For getting the best possible results, the comparison of learning algorithms was needed to find out which one works the best in this project situation. The most known reinforcement algorithms, Q-learning and Sarsa, were compared in the end.

Different parameters were tried with both the Q-learning and Sarsa algorithms. The initial parameters were: α=0.1, γ=0.7, ϵ=0.9 and they were then changed for multiple different times, and test runs were made for each of the changes. Conclusion got from this, α value needed to be small, and the ϵ needed to be high, as it would then be discounted as the episodes were ran. Discount rate for ϵ was set as 0.999 and what it means is that in every episode, new ϵ is set by multiplying the current value with the discount rate, unless the ϵ goes to 0.05 in which case it is not reduced further.

With the α and ϵ parameters decided, it was time to try different γ values. After multiple different runs, the initial values ended up being the best for all of the parameters. The better reinforcement algorithm for this project was the Q-learning algorithm. Also the reward function has a huge factor on the learning process. The function decided to be used behaved nicely with the initial parameters.

The rewarding function rewards the robot based on how far it travels in the map. The closer the robot gets to the desired point before it crashes, the bigger rewards it gets. Also going straight instead of turning is encouraged by rewarding the robot for it. The desired point was placed to the end of the first straight line in the environments. Teaching the robot to go around the corners was not beneficial for this case. That would not only make the robot perform worse, but it would increase the already long training times even more. Basically, the robot gets rewarded when it goes as directly as possible to the desired point, which also encourages the robot to try to overcome the slanting by going straight up.

5.3. Spoofing

The idea about spoofing the LiDAR when the robot is facing a ramp, is to feed such values to the LiDAR sensor that mimic the ones it would receive in a flat environment. This means inputting greater values to laser ranges between 120 and 270 degrees, and

in some cases to rest of the ranges, when the readings go under a certain point, which was 0.2 in this case.

For this a script was created, which publishes larger values in an area where the LiDAR would think there is a wall, even if there is none. The script would have to get triggered some how, so that the script would automatically get started when needed.

Two different triggers for LiDAR spoofing were tested, one suitable for almost any type of environment and one fitted to needs of this project. The first way is to check the orientation of the robot from the odometry sensor values and if the Y goes above the absolute value of 0.02, it will not consider this as crashing since this only means that the robot is on a slanted surface. The other solution used for testing was a bit more reliable. In this solution the spoofing was triggered when the robot enters a certain area based on the position of the robot on the X and Y coordinates.

Whichever solution is chosen, when the trigger happens, the script will add 0.5 metres to laser ranges all around the robot. Multiple different values were tried for specific angles and all-around the robot, but in the end, the solution mentioned above was used. Also the Y-coordinates in the spoofing script are defined the way that if the robot goes close to sides of the corridor, spoofing will end so the robot would receive real laser values, and the robot recognizes the wall in front of it to start a new episode.

5.4. Trained Vs. Untrained Models

Training the robot in the standard environment was an important part of the project. The purpose of training is that the robot could use the previously learned knowledge and therefore avoid walls more efficiently and also to reach its destination more efficiently. The Numpy package was used from the Python library to save the states of the robot after the training had been finished. If a previous training file was saved, the robot will load and use that training data. If the robot is not expected to learn much more and utilize the training data to move around, the learning parameters should be changed in the way that epsilon would be set as 0 and gamma to 0.1. This change in the parameters encourages the robot to use its previous knowledge the most, instead of still keep on exploring new stuff too much.

5.4.1. Reward Function 2

Because it was found out earlier that the second implemented reward function had better learning performance, it ended up being used for the evaluation (see both algorithms in section 4.2.1). Figures 10 and 13 portray the learning curve of the robot training in a standard environment with the reward function 2. In Figure 10 the robot was trained for 4000 episodes to see if any learning had happened. Based on the decreasing amount of negative rewards and growing amount of positive rewards as well as the mean of rewards increasing, learning had indeed happened, even though it did not happen very fast.

Figure 13 portrays the previous training mentioned which has been continued with 3000 more episodes, giving a total of 7000 training episodes. Here can be seen that this additional training has not provided very much progress in learning. On these

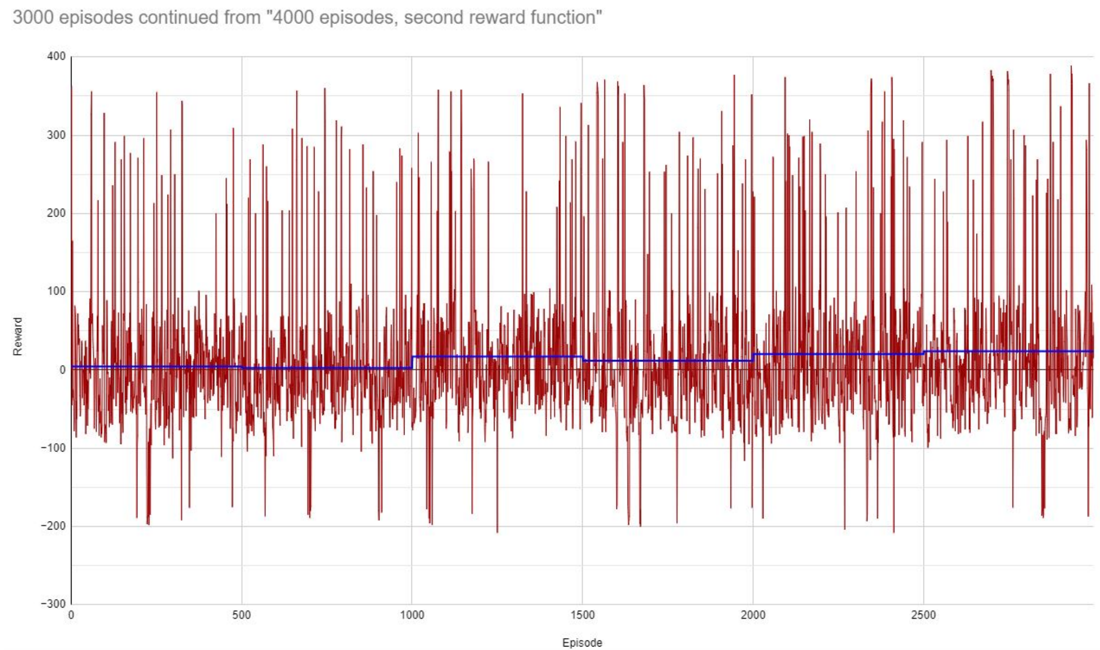


Figure 13. A reward graph of training in standard environment with the 4000 episodes from Figure 10 plus 3000 episodes more, adding up to a total of 7000 episodes. Algorithm 2 is used.

basis, 4000 episodes seems to be enough for a decent training of the robot. Any larger number of episodes were not tried, because 4000 episodes of training already took around 8 hours, and the additional 3000 episodes took almost as long, as the episodes got longer since the robot had already learned quite much. Evaluation runs were ran with the knowledge gathered from these 7000 episodes.

In Figure 14 can be seen a graph where the learned model from the training of Figure 13 was used. In the upper graph, which is the graph from the standard environment, it can be seen that the training has clearly worked because after a while the robot only received positive rewards. That is due to the fact that the robot has learned how to achieve these good positive rewards. The robot also knows which places and which moves to avoid, so it would not get any unnecessary negative rewards. This case proved that the saving and loading of training data works and is very useful.

The lower graph represents the case where the robot has been moved to the slanted floor environment but spoofing is not used to help the robot get over the ramp. This means that the robot has been constantly crashing into the start of the ramp and it receives a lot of negative rewards. This is similar to the case where the robot would not be trained at all. The couple episodes that spiked up on the positive reward side, is most likely due to the latency which occurs in some cases, and the robot does not get instantly restarted. The restarting happens a little bit later, and in these cases, the robot already got on top of the ramp. However, as can be seen from the graph, this is not constant and can not be taken for granted, and is most likely, as described earlier, due to a bug or a sometimes occurring latency in the system.

In Figure 15 the robot is facing the same situation but this time the LiDAR is being spoofed so the robot could go up the ramp. This still results in a lot of crashing but

significantly less than without the spoofing. The graph illustrates that the robot has reached the desired point multiple of times even within just 400 episodes of evaluation. This can be seen from the occasional positive spikes in the graph. In this graph, if the reward is greater than -50, the robot has usually made it halfway up the ramp and if it manages to get over the ramp, the reward is somewhere close to 0. Bigger rewards require that the robot would reach the desired end point. Comparing these two graphs, it can be clearly seen that the spoofing has helped the robot to get over the ramp.

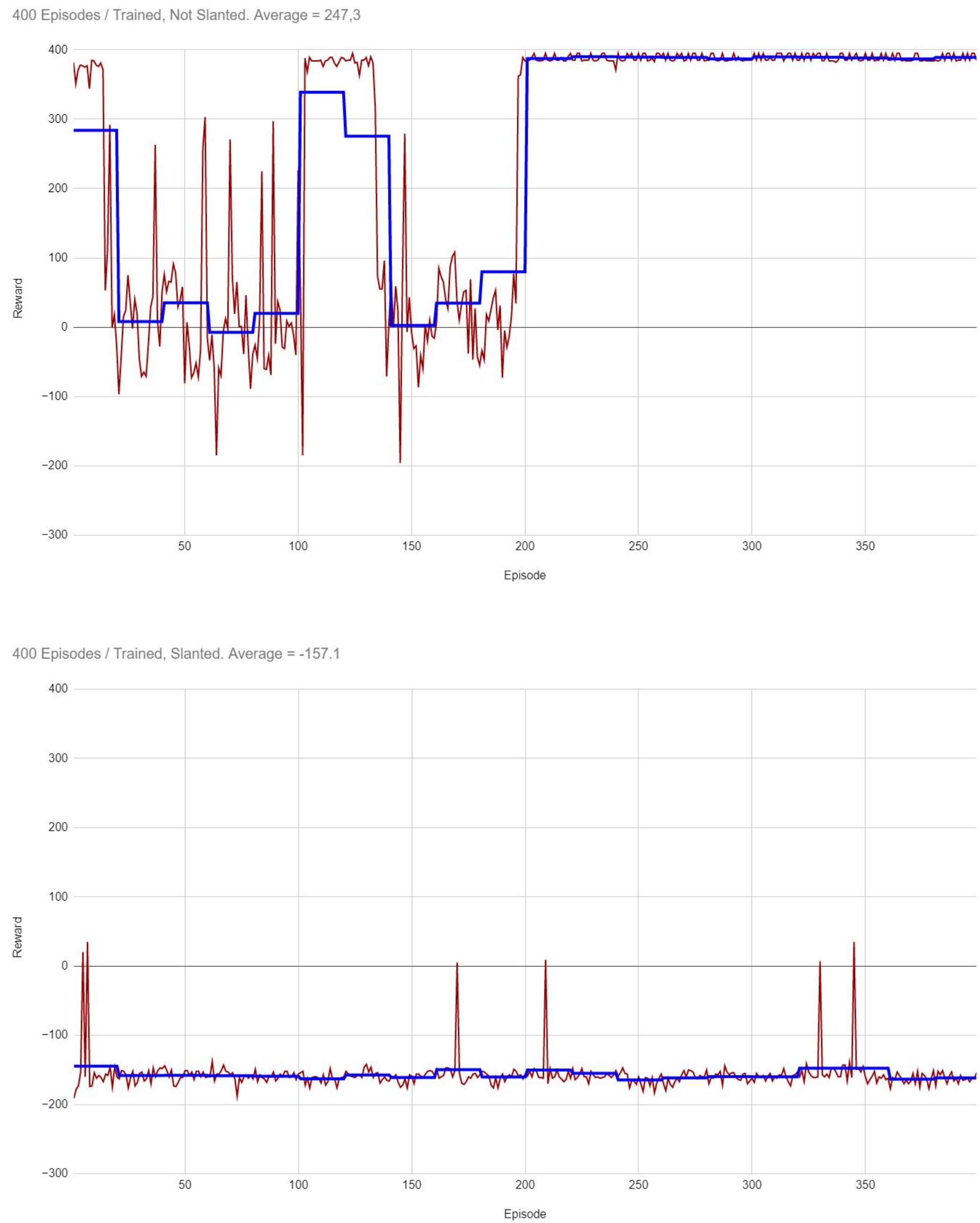


Figure 14. Reward graph of first 400 episodes, both images present trained models. Upper image is in standard environment and lower in slanted floor environment. Reward function 2 is used in both cases.

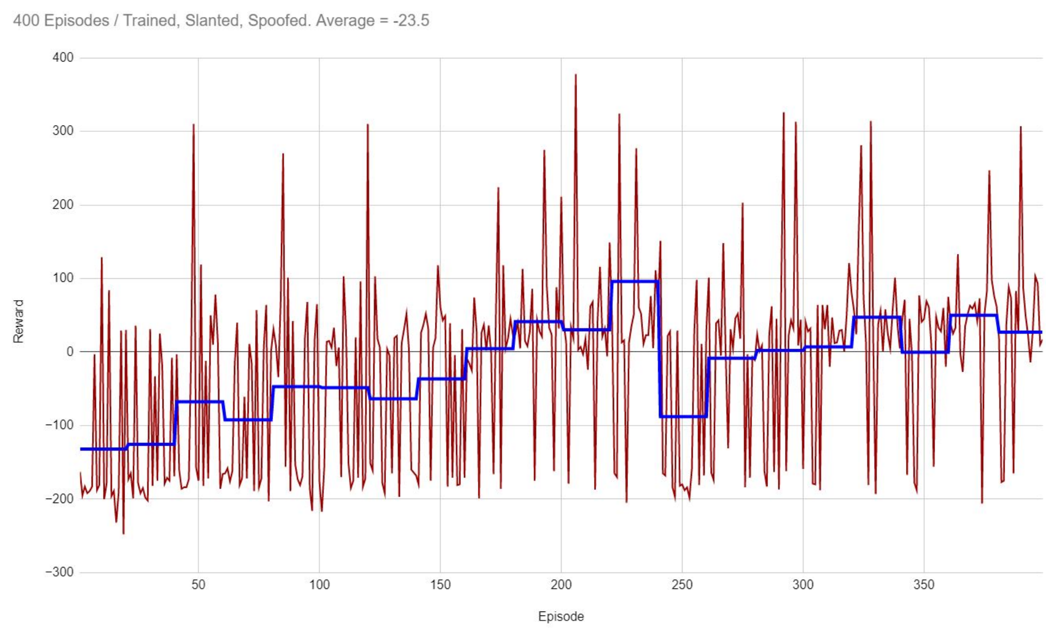


Figure 15. Reward graph of first 400 episodes with a trained model in slanted floor environment, with LiDAR spoofing being used. Reward function 2 used.

5.5. Data-Analysis

For making the comparing of the graphs easier, Graph 1 indicates the upper graph in Figure 14, which is from the trained, standard environment. Graph 2 indicates the lower graph from that same figure, which is from the trained, slanted environment, without spoofing, and Graph 3 points to the graph from Figure 15, which is from the trained, slanted environment, with spoofing.

As the graphs illustrate, the graphs ranks from the best to the worst in the following order: Graph 1, Graph 3, Graph 2. This result is an expected one, since it can not realistically be expected that the outcome would be as good as in the standard environment, considering that the altered LiDAR data is not a complete copy from the standard environment, so the robot can not act precisely the same. However, it still suggests that the robot benefits from the spoofing, as it can be seen when comparing the Graphs 2 and 3. Even though the Graph 3 does not reach to the same level of success as the Graph 1 does, but it still performs significantly better than Graph 2.

When comparing the average rewards across the plots, the results points to the same outcome. The Graph 1 average is 247.3, which is clearly the best one. It shows the benefits of transferring the learnt data well. In the beginning the robot is still crashing every now and then, but it is clearly getting more comfortable with the environment. After 200 episodes the robot is constantly moving to the desired location and receiving very high rewards. The robot also stops crashing completely at this point.

The reward average from Graph 2 is -157.1, which indicates that the robot is not performing well at all. The only times that the robot has not restarted pretty much instantly can be easily detected in the graph. These five episodes the robot has most likely reached around 1/4 of the ramp based on the rewards of around 0 to 25. Due to multiple subsequent runs being done, the five higher rewarded runs are most likely caused by some sort of malfunction with the ROS simulation instead of the robot suddenly performing better. It is highly plausible that there has been slight delay with the reading of LiDAR sensor data on those specific episodes. In this case the robot would be able to reach the 1/4 of the ramp before the sensors detect the crash, which would end up with the robot receiving slightly higher rewards. Even though these sudden jumps of rewards can be easily detected from the graph, they are happening so infrequently that they did not have much impact on the reward average of -157.1. This is why the sudden jumps were classified as errors and further investigation was not done.

When observing the graph 3, the average reward of -23.5 can be calculated. This reward states that on average the robot is reaching around 3/4 of the ramp. The graph 3 shows that the robot is not instantly crashing and restarting as it did in the graph 2. Already in the first 25 episodes the robot has reached rewards of around 125 which is way above the average. It can be clearly stated already, that the sensor spoofing is having an impact on the robot's performance. Having a closer look at the episodes from 50 forwards, it is obvious that the robot has reached the desired point multiple times. Even the further learning of the robot can be seen from the end of the graph as high rewards are getting more frequent and the very bad episodes of around -200 rewards are getting way less frequent.

The difference between the graph 2 and graph 3 is enormous and it can be stated that the sensor spoofing is doing its trick. However there is still huge gap in the robot's

performance when comparing graph 3 to graph 1. This is due to the fact that the LiDAR spoofing script of this project only provides more common data values, instead of extremely precise ones. To tackle this, the modified data values from the LiDAR should be extremely accurate, perhaps taken from another robot which is moving exactly the same way, but in the standard environment. Or the behaviour of the robot should be changed in a way that it does not depend on the accurate LiDAR data, but perhaps uses the coordinates that it has learned before for the movements.

6. DISCUSSION

6.1. Limitations

The project has some limitations and features that could work with minor edits to the code but have not been tested. One of these features is the angle of the slanted floor and whether changing the angle results in the spoofing not working. The script that is used for spoofing the TurtleBot's LiDAR defines what values are wanted to add to the values that go below 0.2, which is the threshold for the robot to consider it has crashed, and by increasing these values this script should theoretically also work on an even more slanted floor but as said, this has not been tested in this project.

From the two reward functions created, the second one which also was the one used, was highly based on the environments built for this project so that the robot would avoid the walls of the environment. It basically defines some hard coded safe zones in the environment where the robot receives rewards, so it would prefer to go straight ahead instead of turning to the walls. For other environments this would have to be modified as well as the desired point that the robot is trying to reach. Obviously if the map changes the desired point changes along with it. However, it is common on these ROS projects, that the desired task for the robot is tailor-made for the individual project it is being used to, and the rewarding function has a huge factor on how the robot learns, so it can not be taken granted that some random reward function works automatically on another environment and perhaps with a different task.

Due to the use of coordinates in the second reward function, if the robot would be spawned from a place other than the start of the map or the environment would be placed on different coordinates, the rewards would not work as intended and therefore the robot would not perform well. With the first reward function this might work since it only calculates the distance to the desired point instead of defining certain coordinates other than the location of the goal. However, also in the first reward function the desired point would need to be changed, and this applies to most other implementations out there as well.

6.2. Simulation Vs. Real Robots

One of the big subjects which can not be forgotten is the difference in the simulated robots versus the real ones. Is it sure that will the findings of this project be the same with real robots, or is this only functional in a simulation? Of course the only real, and a really good way to find out, would be to test this case with a real robot. However, even if this is not possible to do with the tools used right now, it can be discussed about the differences and possible problems that could occur with a real robot instead of a simulated one.

6.2.1. Spoofing the LiDAR

The first and perhaps the biggest question that comes to mind is that how to spoof the LiDAR in a real robot? Attacking on the LiDAR from outside can be difficult [37],

however, if the LiDAR spoofing is desired and implemented in the robot hardware and software, spoofing should be easier to produce. After all, it is all about data, and if there would be an in-built system where the original data can be blocked, alter it, and then let is pass, there should be no problem to do so.

The more difficult part in this would be the data altering. For the spoofing to work desirably, there must be an intelligent script to alter the data the way, that the robot does not get confused about it and it carries on with its on-going job. To make the robot perform desirably, the data altering can not be fixed. The script doing the altering should be so sophisticated, that it could alter the data to be favourable for the actual space the robot is at that moment. And it still should be able to detect any real obstacles in its path. So all in all, the spoofing script should be intelligent and effective, and it should be able to adapt in the situation where the robot is.

The spoofing should perhaps be triggered by the tilting of the robot instead of the coordinates used in experiment of this project. However, depending on the robot, the height of the LiDAR might be a problem. If the LiDAR is located on a low position of the robot, which in this case means that it is close to the floor, the LiDAR might indicate the robot to not even go on the slanted part of the floor at all. And if this would be the case, the LiDAR would not get spoofed at all. To overcome this problem, some kind of coordinated solution would be beneficial for the robot. For example, the solution could be something like the following: the robot does not draw the map for itself on a certain point - in this case where the slanting starts - of the room, or it draws it according to previous knowledge, and it would discard the fake obstacles it sees on those specific coordinates.

6.2.2. Differences in Robot Models

As explained in the previous paragraph, the LiDAR should be placed at a suitable height, in order to get the project implementation to perform correctly in a real-world situation. In this case, the implementation was performed with the TurtleBot3 robot, which has three different models: Burger, Waffle, and Waffle Pi [38]. From these models, the Burger model stands out in this case, since the height of the model is considerably greater than Waffle and Waffle Pi models. The height of the burger model is 192 mm and other models are 141 mm. Exploring the specifications of models, it can be noticed that LiDAR is located on top of the robot. Because of this, the Waffle model were used in this project. As it had been tried to use the Burger model at the start of the project, the slanted floor could not be recognized in the test environment, because the LiDAR height of the model was greater than the environment's slanted floor height.

According to findings, the Waffle and Waffle Pi models would give the best results with the implementation of this project in a real-world situation. However, there still are many variables that can affect the end results, for example, the angle of the slanted floor, the friction between wheels and floor, clearance between the robot and the ground, and the uneven surface of the floor.

6.2.3. Different Approach

With a real robot instead of a fully simulated one, the same kind of training could be done by spoofing the sensors in the training phase to mimic the environment with a flat floor. In fact, this would be the optimal case for the training, because then the data from odometer would be a precise match for the real case, because it would actually go up the ramp, but only the LiDAR, and perhaps the camera as well, would not see the ramp.

The sensors could perhaps be spoofed using VR-glasses or just by inputting modified data to the sensors. And after the robot has learned the environment and the task, the continuous learning parameters should be tweaked to minimum, so the robot would not learn that the slanted floor is a wall, but that it would still rely on the LiDAR and the camera when they inform the robot that there is a new obstacle in front of the robot and it must be dodged to avoid a collision.

6.3. Transfer Learning

The initial goal was to spoof the robot in the standard environment, train it and move it to a slanted floor environment where it should know how to perform. This would be done by inputting such values to the LiDAR that it would receive if there in fact was a slanted floor, which means small values in the backside of the robot. Then the robot should realize that these kinds of values do not prevent it from moving because there is no actual wall. Modifying the learning algorithm was not managed in a way that would be possible because the robot always crashed right away when it received the spoofed values and could not figure out it could move forward despite these values. This is why the LiDAR is being spoofed when the robot tries to go up the slanted floor to show the kind of data the robot would receive in a flat environment.

This was mainly due to the nature of the reward functions and learning method. Implementation of those functions rewards the robot based on the actions it decides to take and whether those actions take it closer to the desired point. For the transfer learning to work in the way that was originally intended the learning should somehow happen in a way that the robot could explore the environment even if it receives certain values that throw it off and at some point realize there was no wall in the place that it originally thought there was. This would require much more sophisticated algorithms and scripts to work, and most likely it would still cause problems for example when detecting actual, sudden obstacles.

The solution that ended up in the project still works well for evaluating the feasibility of transfer learning. The learned model, which was taught for the robot in the standard environment, was taken into use in the environment with the slanted floor. Then the evaluation runs were run to see how the robot performed. The LiDAR spoofing helped the robot with its task, even though it did not manage to gain the same level of performance than in the standard environment, which is expected. Perhaps with a more complex and efficient spoofing methods and scripts the robot would perform better, and especially if the training would be continued with the spoofed sensor.

6.4. Future Work

In the beginning the idea was to make the robot sweep the floor, which means that it would go through every node of the environment without visiting any previous places. Executing this and combining it with learning algorithms proved to be difficult and would have required much more time. This is something that could be done in the future with more knowledge of ROS, transfer learning and robotics. This would provide valuable information on how this would work with a more complex task.

Spoofing could also be done to any sensor with slight modifications to script of this project. This means that slanted floor is not the only obstacle the robot could face, there could be for example moving objects or dead ends in the environment. The possibilities of spoofing are basically endless, and even better results could be obtainable if more sensors would be spoofed together, and the modified data would be extremely accurate for the desired case.

In the future, it would be beneficial to test the project implementation with a real robot and see how it would behave. Even though the simulation should be relatively accurate in showing how the robot would perform in the real scenario, the final and trustworthy results would only come if a real robot was tested in a real environment.

7. CONCLUSIONS

The aim of this project was to combine transfer learning with sensor spoofing to help mobile robots overcome obstacles more efficiently. In the beginning it was decided that it would be reasonable to start working with well known problems that mobile robots are experiencing. The main problem to tackle in this project is very common with robots such as vacuum cleaners. The cleaners usually struggle with slanted surfaces.

The usage of transfer learning is overall very effective technique when the robot is being taught to move around in unfamiliar environments. However in this particular situation the usage of transfer learning was not effective enough. The problems that the mobile robots have with slanted surfaces is highly related to problems with LiDAR sensor inputs. To overcome this problem, the spoofing of the sensors was needed. This provided valuable information about transfer learning and spoofing, and also some reasonable preliminary results for the problem. The results could get improved, if the development of the spoofing script and methods would be carried on to the next level.

The actual project shaped into using sensor spoofing to help the mobile robot overcome obstacles that it would otherwise have difficulties with. Another important aspect of the project was to be able to use the robot's learnt models across different environments. Based on the analysis of the data obtained, it can be concluded that the spoofing and the transferring of the learnt model from one environment to another can improve the performance of the robot especially in situations that would otherwise cause sensor input related problems, such as moving in environments with the slanted floors.

While the sensor spoofing can be used to improve the robot's performance in varying surfaces, it also raises the question of whether there are other ways to handle the situation. The problems that the robot had in slanted surfaces, was mainly caused by the LiDAR sensors of the robot. The TurtleBot, which is the robot used in this project, has multiple other interesting sensors that could be useful in multiple ways. The other idea was to start using another sensor when the TurtleBot is in a situation where it might have sensor related issues. Instead of spoofing the sensors straight away, the data could be collected with TurtleBot's Raspberry Pi Camera for example. It could be possible that the robot would be able to collect the necessary data inputs no matter what situation it is in by switching the data collecting sensors. This idea would be suitable for further research as it could improve the robot's performance in situations where the sensor inputs are causing problems.

The usage of transfer learning has already proven its strengths in multiple different problem scenarios. The robots might still end up in situations where it simply does not understand what is going wrong. Just like in the case of the slanted floors, the robot might not be able to get past the more difficult test environment even though it has practiced the correct movements over and over again in the standard environments. According to the results got from this project, the sensor spoofing could have very positive impacts if used correctly with transfer learning.

8. REFERENCES

[1] Mohri M., Rostamizadeh A. & Talwalkar A. (2018) Foundations of machine learning. MIT press, 1 p.

[2] Torrey L. & Shavlik J. (2010) Transfer learning. In: Handbook of research on machine learning applications and trends: algorithms, methods, and techniques, IGI global, pp. 242–264.

[3] Pan S.J. & Yang Q. (2009) A survey on transfer learning. IEEE Transactions on knowledge and data engineering 22, pp. 1345–1359.

[4] Barrett S., Taylor M.E. & Stone P. (2010) Transfer learning for reinforcement learning on a physical robot. In: Ninth International Conference on Autonomous Agents and Multiagent Systems-Adaptive Learning Agents Workshop (AAMAS-ALA), vol. 1, vol. 1, pp. 24–9.

[5] Bocsi B., Csató L. & Peters J. (2013) Alignment-based transfer learning for robot models. In: The 2013 international joint conference on neural networks (IJCNN), IEEE, pp. 1–7.

[6] Popov D. & Klimchik A. (2020) Transfer learning for collision localization in collaborative robotics. In: Proceedings of the 3rd International Conference on Applications of Intelligent Systems, pp. 1–7.

[7] Liu Y., Li Z., Liu H. & Kan Z. (2020) Skill transfer learning for autonomous robots and human–robot cooperation: a survey. Robotics and Autonomous Systems 128, p. 103515.

[8] Liang X., Liu Y., Chen T., Liu M. & Yang Q. (2019) Federated transfer reinforcement learning for autonomous driving. arXiv preprint arXiv:1910.06001 .

[9] Zamora I., Lopez N.G., Vilches V.M. & Cordero A.H. (2016) Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo. arXiv preprint arXiv:1608.05742 .

[10] Imanberdiyev N., Fu C., Kayacan E. & Chen I.M. (2016) Autonomous navigation of uav by using real-time model-based reinforcement learning. In: 2016 14th international conference on control, automation, robotics and vision (ICARCV), IEEE, pp. 1–6.

[11] Tzafestas S. (2013) Introduction to Mobile Robot Control. Elsevier, 2 p.

[12] Carballo A., Lambert J., Monrroy A., Wong D., Narksri P., Kitsukawa Y., Takeuchi E., Kato S. & Takeda K. (2020) Libre: The multiple 3d lidar dataset. In: 2020 IEEE Intelligent Vehicles Symposium (IV), pp. 1094–1101. DOI: 10.1109/IV47402.2020.9304681.

[13] Yan Z., Sun L., Duckctr T. & Bellotto N. (2018) Multisensor online transfer learning for 3d lidar-based human detection with a mobile robot. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 7635–7640. DOI: 10.1109/IROS.2018.8593899.

[14] Bellotto N. & Hu H. (2009) Multisensor-based human detection and tracking for mobile service robots. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 39, pp. 167–181. DOI: 10.1109/TSMCB.2008.2004050 Accessed 6.2.2021.

[15] Rabbah M., Rabbah N., Belhadaoui H. & Rifi M. (2018) Python in real time application for mobile robot. SSRN Electronic Journal DOI: 10.2139/ssrn.3186285.

[16] Joseph L. (2018) Learning Robotics Using Python: Design, Simulate, Program, and Prototype an Autonomous Mobile Robot Using ROS, OpenCV, PCL, and Python, 2nd Edition. Packt Publishing, 2nd ed.

[17] Majdik A., Popa M., Tamas L., Szoke I. & Lazea G. (2010) New approach in solving the kidnapped robot problem. In: ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics), VDE, pp. 1–6.

[18] Kerns A.J., Shepard D.P., Bhatti J.A. & Humphreys T.E. (2014) Unmanned aircraft capture and control via gps spoofing. Journal of Field Robotics 31, pp. 617–636.

[19] Psiaki M.L. & Humphreys T.E. (2016) Gnss spoofing and detection. Proceedings of the IEEE 104, pp. 1258–1270.

[20] Rivera S., Lagraa S., Iannillo A.K. & State R. (2019) Auto-encoding robot state against sensor spoofing attacks. In: 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), IEEE, pp. 252–257.

[21] Suomalainen M., Nilles A.Q. & LaValle S.M. (2020), Virtual reality for robots.

[22] Davidson D., Wu H., Jellinek R., Singh V. & Ristenpart T. (2016) Controlling uavs with sensor input spoofing attacks. In: 10th {USENIX} Workshop on Offensive Technologies ({WOOT} 16).

[23] Ingabire O., Kim M., Lee J. & Jang J.w. (2019) An implementation of the path-finding algorithm for turtlebot 2 based on low-cost embedded hardware. International Journal of Advanced Culture Technology 7, pp. 313–320.

[24] About ros. URL: https://www.ros.org/about-ros/, accessed: 09.03.2021.

[25] Quigley M., Gerkey B. & Smart W.D. (2015) Programming Robots with ROS: a practical introduction to the Robot Operating System. " O'Reilly Media, Inc.".

[26] Ros. URL: http://wiki.ros.org/ROS/Introduction, accessed: 09.03.2021.

[27] Tiwari K. (2018), Ros-101. URL: https://github.com/ktiwari9/ros101.

[28] How to launch the turtlebot3 simulation with ros. URL: https://automaticaddison.com/how-to-launch-the-turtlebot3-simulation-with-ros/#gazebo, accessed: 27.03.2021, Published: 15.05.2020.

[29] Openai ros. URL: http://wiki.ros.org/openai_ros, accessed: 09.03.2021, Last edited 11.4.2019 by Miguel Angel Rodriquez Martinez.

[30] Learn the basics of openai ros using a turtlebot2 simulation. URL: http://wiki.ros.org/openai_ros/TurtleBot2%20with%20openai_ros, accessed: 09.03.2021, Last edited 16.9.2020 by Alan Dougherty.

[31] Takaya K., Asai T., Kroumov V. & Smarandache F. (2016) Simulation environment for mobile robots testing using ros and gazebo. In: 2016 20th International Conference on System Theory, Control and Computing (ICSTCC), IEEE, pp. 96–101.

[32] Huang T., How to read laserscan data (ros-python). URL: https://www.theconstructsim.com/read-laserscan-data/, accessed: 27.03.2021.

[33] Robotis e-manual. URL: https://emanual.robotis.com/docs/en/platform/turtlebot3/slam/, accessed: 27.03.2021.

[34] rqt_graph documentation. URL: http://wiki.ros.org/rqt_graph, accessed: 13.04.2021.

[35] turtlebot3_bringup documentation. URL: http://wiki.ros.org/turtlebot3_bringup, accessed: 13.04.2021.

[36] Watkins C.J. & Dayan P. (1992) Q-learning. Machine learning 8, pp. 279–292.

[37] Cao Y., Xiao C., Cyr B., Zhou Y., Park W., Rampazzi S., Chen Q.A., Fu K. & Mao Z.M. (2019) Adversarial sensor attack on lidar-based perception in autonomous driving. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS '19, Association for Computing Machinery, New York, NY, USA, p. 2267–2281. URL: https://doi.org/10.1145/3319535.3339815.

[38] Robotis e-manual features. URL: https://emanual.robotis.com/docs/en/platform/turtlebot3/features/, accessed: 13.04.2021.

9. APPENDICES

9.1. Contributions

Stage 1		
Student	Hours	Contributions
Katri Säily	18	Research on transfer learning, writing
Ville Kivikko	17	Research on transfer learning, mobile robots. Writing
Markus Kyllönen	16	Research on transfer learning, mobile robots. Writing
Lassi Perälä	16	Research on transfer learning, robot spoofing and mobile robots. Writing
Stage 2		
Katri Säily	90	Research and installations, Environment setup and debugging, Sensor data collection and real-time plotting, Scripting, Writing
Ville Kivikko	110	Research and installations, Environment setup and debugging, 3D modeling, Implementing and testing, Scripting, Writing
Markus Kyllönen	115	Research and installations, Environment setup and debugging, Implementing and testing, Writing
Lassi Perälä	100	Project management, Communications with the supervisor, Research and installations, Environment setup and debugging, Implementing and testing, Writing

Stage 3		
Katri Säily	92	Evaluation, writing, saving the model, research on training algorithms, comparing and creating the learning algorithms, scripting
Ville Kivikko	114	Comparing the learning algorithms, creating and comparing the reward functions, scripting, research, training the robot, running the evaluation runs for the robot, evaluation, writing
Markus Kyllönen	103	Scripting, training the robot, writing, evaluation, comparing and creating the learning algorithms, reward functions, research
Lassi Perälä	95	Project management, Communications with the supervisor, Comparing and debugging different learning algorithms, Creating and comparing the reward functions, Running training and evaluation runs for the robot, Research, Evaluation, Writing
Stage 4		
Katri Säily	15	Writing
Ville Kivikko	12	Writing, analysing
Markus Kyllönen	11	Writing, analysing
Lassi Perälä	15	Writing, Communications with the supervisor

Total	
Katri Säily	221
Ville Kivikko	253
Markus Kyllönen	245
Lassi Perälä	226

9.2. Terminal Commands

File names can naturally be different, but the basic commands remain the same. Initiate Gazebo environment, launch file name depends on the world that is launched:

$ roslaunch turtlebot3_gazebo <LAUNCH_FILE_NAME>.launch

Start training the Turtlebot with OpenAI:

$ roslaunch my_turtlebot3_openai_example start_training.launch

Plot laser sensor data in real-time:

$ roslaunch laser_values laser.launch

The learning curve can be plotted in real-time by using ROS rqt-multiplot:

$ roslaunch rqt_multiplot rqt_multiplot

This command opens up a new window where the user can create a plot based on active ROS topics. OpenAI can be found by writing /openai/reward and setting the x-axis to show "episode_number" and y-axis "episode_reward". This way the x-axis will portray the number of training episodes and y-axis the rewards the robot has received in total. In Figure 8 you can see an example of training in one of the standard environments called TurtleBot3 World.

Run LiDAR spoofing script:

$ python3 spoofing.py