Department of Computer Science

Submitted in partial fulfillment for the degree of MEng

Testing robots in a simulation

Ionut Manasia

04 May 2022

Supervisor: Simos Gerasimou

TABLE OF CONTENTS

Executive Summary

Robotic simulation poses a new and better way of testing robotic systems. Instead of deploying the hardware in the field without being fully validated and risking damage, we can do that for the cheap inside of a simulated world.

My mission was to leverage the power of the robotic developing and simulating software in order to create my model and test its fitness. In this paper, I have designed my simulation and my mission, and I have assessed the reliability of my robot in completing the mission.

To accomplish that, I have used ROS, a robotic development software, and Gazebo, a simulation software capable of accurate physics representation. I have implemented the mission and the environment using the software mentioned earlier and used it to test the quality of the robot. Multiple tests have been generated in a random fashion so that I can use them to surface any unexpected behaviour that might happen due to corner or edge cases.

My results have been positive, showing that the robot and its dependent packages are fully capable of navigating the mission. This includes its ability to map the environment, localize itself in the environment, to create a moving path that would avoid collision with other objects. Alongside all of the specified generic tasks, the robot was also capable of completing my designed robotic mission in a timely manner. The use of generative testing has been implemented through controlled randomness both in the design of the mission as well as in the design of the environment. This has made the stress-testing of my robotic model easier, as I could generalize a model of designing the tests instead of continuously developing hard-coded cases. All the hard figures, alongside images and videos of the robot, can be found inside the document or the repository. The results are easily reproducible in this manner. Repository found at: https://github.com/HigherTSC/Bachelor-Project .

Alongside the positive results, some unexpected behaviour is linked to the way the localization and navigation algorithm works. This is in part to the fact that adaptive self-localization is a highly complex task for any robot. The mission can be considered a success, but further study could help cross-validate my findings. Specifically setting up the mission in a real-world environment with actual physical software. This can be relegated as further work that can be done to improve upon my study.

My study poses no legal, social, professional, or commercial issues of any kind. It makes use of specifically open-source software and cites

every material used, both in the documentation and the actual programming of the mission.

1 Introduction

Robotics is a relatively new and exciting field. Autonomously operated robots are seeing more and more widespread use with the popularity of self-driving cars and UAVs. Today, a robot is a complex hardware device comprising multiple sensors and computers controlled by complex software. Their application could have massive potential in many fields, such as marine exploration, space exploration, personal assistance, and even in more productivity-based fields, such as agriculture or industry.

However, the popularity of this new autonomous robot raises a broad new area of challenges to validate their efficiency and consistency. This is why testing them is a valuable way to help these devices improve before deploying them in the field.

The issues that may arrive with testing the robots are that developers have to spend valuable resources and time perfecting a model. At the same time, the general public or the companies that requested said robot is looking to start the operation as soon as possible and with the most negligible costs. Even after testing is set up, particular types of tests, specifically those in the field used for mission validation, could be dangerous if the agent misbehaves.

This is why simulation can be one way to mitigate this because a simulated robot can act adequately close to the actual physical robot. The hardware is not at risk of damage while performing said tests. At the same time, a multitude of extra tests can be performed since the robot, and the environment can easily be reset and modified.

Simulation software has constantly been developing and improving to facilitate that. It has become increasingly robust and is capable of developing, handling, controlling, and simulating complex robotic systems and environments where these systems might have been deployed.

Large communities dedicated to robotics have even started porting their real-life robots to this software with high fidelity. Extensive work is starting to be done in the field [1-3]. Some have even ported complex missions into simulated worlds. Such as Robotis, which has made not only their turtlebot 3[4] available but also a multitude of simulations. One is even as complex as showing turtlebot capabilities of navigating through a racecourse, following lanes, traffic signs, traffic lights, stopping at barriers, and navigating through dark areas [5].

Existing literature in this field is also compromised of mostly robotic engineers that have been capable of nearly creating a 1:1 replica of their robot behaviour inside a simulation [6]. This is being done to show

the software's full capabilities. It has been done as part of the Defense Advanced Research Projects Agency(DARPA)’s Robotic Challenge [7].

The mission was to leverage the power of that software solution for robotic development and 3D simulation to create a robotic model with each own task and test its abilities to perform said tasks. I have decided to create an autonomous mobile robot whose task includes having to visit a certain amount of points in a certain amount of time. This is meant to simulate a recognizance mission. The checkpoints will be objectives that my robot is meant to take photos of, and the time constraint is meant to simulate the battery of a portable camera.

My approach to completing the task at hand was to use robotic development software to create a robotic model and program it to complete the task given. At the same time, I have set up a simulation using a 3D physics-based simulator that will provide the environment necessary for the task to be completed. Variations to the task and the environment have also been added programmatically and used as test cases.

A benchmark case has been created as a case study, where the robot finds itself on an empty map and only needs to follow a small number of checkpoints. It would lead it to run a single lap across the track, which is the most straightforward situation my agent could be put into. Every other scenario where multiple obstacles and checkpoints have been added has been thoroughly examined to identify corner cases. These corner cases have been analysed, including possible causes and ways they could be avoided, as the damage to my mission to be mitigated.

The remainder of this report is structured as follows: Section 2 discusses the core concepts behind the project and reviews the literature. Section 3 describes the methodology used in this study. Section 4 provides and analyses the results of my tests. Section 5 provides concluding remarks.

2 Background and Literature Review

2.1. Robotics and robots

Robotics is a computer science and engineering branch that deals with designing, constructing, operating, and using robots. Robots are programmable machines that are capable of carrying out complex tasks autonomously or while being remotely controlled by an external agent.

Robotics has been playing an increasingly important role in our daily lives, ranging all the way from industrial robots to drones and even robots used for extra-terrestrial exploration. All these devices vary significantly in size and purpose, with industrial robots used to manufacture all types of goods and needing to be precise. Drones also started seeing multiple divergent uses, from filming to delivering medicine in remote areas where storage is not an accessible solution. However, this increased usage of robots lends itself to possible faults that can have catastrophic consequences. For this reason, engineers must continue and develop assurance techniques for this robot and robotic systems.

2.2 Testing robots

Testing robots has predominantly been carried out using physical machines in a real-life environment. While results produced by these tests will prove to give reliable results, they are also complicated to carry out and expensive. Because of the need for physical agents for a real-life test, any faults in the programming or designing of the robot can lead to damage to the robots or the environment. Damage can further cause monetary or time losses, as the robots might need to be repaired or replaced. This can further discourage testing altogether and lead to faulty devices being sent out.

A workaround for this problem can be using simulations to test the robots. A well-designated simulation can convincingly replicate environmental factors, and at the same time, it will allow a low-risk method of conducting said tests. Because of that, more tests can be done with a more extensive array of variables and cases. Cases that might have been avoided while conducting a real-life test on a different basis, such as the sheer improbability of it happening.

2.3 Corner and Edge Cases

In engineering, a corner case refers to a situation in which a device might misbehave because multiple values are extreme and fall outside the limit of its working parameters or outside its typical collection of working parameters. This contrasts to edge cases where the values

are just at the limit value for the device to be still operating inside its working parameters.

Testing for edge cases, also known as fringe testing, is essential because usually, that is where most of the different and more strange behaviour of the device tends to happen. Fringe testing can include testing out of sequence, twisting the expected workflow, or discovering functions the application performs that it was not designed for.

On the other hand, corner-case testing is just as hard to test for as they mainly occur when one or multiple of the components fail and the device is not able to recover. Testing for them will let us find these specific values and can show us if the problem is a fault in the programming or not. They can help us find a specific implementation that will mitigate the damage done to the device in these cases. And if something unexpected happens while the device is in the field, this technique can more easily assess what happened if the symptoms are known and follow one of the results of the tests.

Both edge and corner testing are vital for the complete assessment of the device, but in the field of testing, they are often overlooked as it takes a considerable amount of time to find and test them. They can also be dismissed as something that will never happen. However, in accordance with human behaviour, one cannot expect robots always to follow the rules imposed on them, and it is better to be prepared for when such rules are pushing the device to its limit.

2.4 Simulation and Software

Since simulated testing holds so much potential, multiple pieces of software have been designed with this exact goal in mind. Those framework applications are capable of creating the environment necessary for the testing, containing different rules, obstacles, and even environmental conditions, while at the same time being able to run mock robots, controlled either by their own AI or manually by the user. This type of software gives robotic developers ample options to test their algorithm in as many scenarios as they want to improve their design.

Some of this software include WEBOTS[8], which supports multiple programming languages and has TCP/IP interface that can communicate with other software. Robot Virtual Worlds[9], which was created as an educational tool, LabVIEW[10], which is more of a complex software system for data acquisition, analysis, and control, but it also has numerous libraries for simulating hardware components or OpenRTM-aismt[11], which is a software-based on the RT

middleware standard. On the side simulators, there is USARSim[12], which is used in the RoboCup[13] robot competition. This one is based on Unreal Engine[14], but it is now a separate entity. Even though it is a game engine, Unreal Engine can also be used to create 3D simulations. It has a large community and extensive tools but lacks easy implementation with robotic development software. There is also OpenHRP[15], based on OpenRTM-list, which is primarily designed for humanoid robots.

Out of those packages, the ones that I have decided to use for my simulation were ROS[16] and Gazebo [17]. ROS is an open-source meta-operating system for robots. It is feature-rich and has operating system-like features, such as hardware abstraction, low-level device control, implementations of the most commonly used features available in robots, debugging tools, process visualization, and managing as well as the ability to create and maintain multiple packages. Gazebo, created by the same team, but functioning as a separate entity, is a 3D dynamic physics-based simulator. It is similar to game engines, but it is built to allow a much higher degree of simulation fidelity in essential elements such as physics and sensors. It is also fully integrated with ROS.

2.5. Literature review

The implementation of simulation for testing robots is a well-studied topic, with a wide variety of relevant literature. This literature can be split into multiple categories based on the covered types of robots, such as manipulators, underwater vehicles, and unmanned aerial vehicles (UAVs). One type of study tries to identify and discuss the main differences and challenges of robotic testing inside a simulation.

In their 2020 questionnaire[18], multiple authors send to find out the industry standards for simulation-based testing and conclude with a series of 10 high-level challenges that hinder developers from using simulation in general. They have found general issues, like the reality gap, where one cannot trust that the results of their simulated robot will be the same as the real device. Other reasons are software-based, like the lack of capabilities, simulator reliability, and interface stability. Even though software for controlling robots and creating simulated, physics-based environments has existed for years and is continuously being developed, some consider that the technology is still in its infancy and lacks certain features and the required stability for conducting said tests. Multiple developers have found it easier to test their robots headless than using the GUI on the same topic. And the last type of issue found was the complexity of setting up a good simulation and a good model-type robot. Creating a near real-world identical simulation is time-consuming and requires the creation of several different models and environments. Despite all of these issues,

the same study has found that 84% of the participants have used a robotic simulator for their projects. This shows again that the perceived advantages of using a simulation outweigh the disadvantages.

Another study[19] done by almost the same team has questioned multiple developers and has tried to quantify the practices and challenges of the robotics community while using a simulation and put them into three categories. This helps create a framework for how testing in a simulation can be done optimally. The three main categories they identified are Real-world complexities, Community and standards, and Component integration. This falls in line with the findings of their other study, showing a clear parallel between those categories and the field of robotics simulation. One that has not been discussed in the other study is the Community and standards. This has shown that a good amount of the Community lacks faith in testing. This is due to the different backgrounds found in the robotics community. One developer has said that people from a software background will observe testing differently from others. Another noting is that the robotics community is more focused on the image of robotics rather than the results. Those sentiments seem to imply the importance of testing further when it comes to delivering a good, well-working model.

The opinion at large is that simulation can be beneficial to engineers and developers while designing and testing their products. Even though some limitations have been found and discussed, developers still opted to at least try using simulation before deploying their actual devices in the field. I suspect that as time goes by and the simulation software keeps expanding and becoming more and more complex, the adoption of the method will become more widespread. Now, on a different line of thought, I decided to research the actual development and implementation of robotic testing and how it might differ in a simulation rather than in real life.

Academics are discussing multiple methodologies for testing robots in the simulation; one of them includes the work[20]. They have set up to devise a way of defining tests specifically for mobile autonomous robots and have come up with multiple parameters and rules that should be considered for this explicit purpose. They devised a set of 3 requirements for the robot test process, which are as follows: safety, re-use, and repeatability. While this study is focused on field testing, I can extrapolate how all of these factors can be interpreted in a simulated world. For example, I do not have to worry about any of the three factors. Since I am using a simulation, the safety of the robot is guaranteed as no physical hardware is put to the test that might result in it getting damaged. The re-use and repeatability of the test are in the same vein, infinite, as any simulation can be launched and relaunched as many times as it is needed. Apart from this, they also devise a series of test dimensions, which will be important while

designing my tests, those being sensing-only vs. sensing and acting, tester knowledge of the environment, and the dynamicity of the environment. These are all dimensions that need to be kept in mind, as they affect both the design and ability to interpret the results of the tests accurately.

The methodologies for testing robots developed for either in-field testing or in-simulation testing overlap in some key aspects, such as being able to define repeatable cases that will give us significant results, the ability to analyse those significant results, and even anticipate from the design, what might pose difficulty to the robots. The simulation also comes with its own set of challenges that have been discussed, so additional methodologies for testing such systems in simulation have to be found.

The validity of simulation-based testing compared to field testing is discussed again in a study[21]. This is done by performing a wide range of automated, generated tests in a simulation-based world and then comparing the results with those of the field tests. This paper gives us a firm overview of the exact capabilities of simulation-based testing. It also discusses in depth how the authors have decided to engineer their tests, considering multiple methods based on the amount of data, computability, and validity. Some of the types of tests discussed are high and low fidelity simulation. A high fidelity simulation is close to its real-world analogue but requires much computational power and a detailed simulation model.

In contrast, a low fidelity simulation is more limited in its scope and environmental details. They chose to go for low fidelity simulation. Fortunately, different studies that this type of simulation may also yield relevant results. He has chosen a generative approach to create test cases, where one needs to generate base simulations and mission templates. This requires less real-world data than hard-coded tests but more design since the engineer has to develop a generalized model that will adequately fulfil the model. This requires proficiency in a particular field. Finally, once the tests had been set up and compared to those in the field, he concluded that a simulation is a reliable tool for finding failing points. Even though it was limited to skidding type of failures, it still managed to catch multiple high-level bugs and errors in the systems.

There are a plethora of studies that try to use the exact setup of ROS and Gazebo[22-27]. One of them[22] is similar to the above [21]. They are trying to check the validity of the simulation, looking for similar behaviour between that of the simulated robot and the one deployed in the field. The authors propose using the simulation software to test the validity of two mapping algorithms and a navigation algorithm, both indoors and outdoors. They use SLAM to perform 2d unknown

environmental mapping and then AMCL to navigate the created environment. They conclude by saying that there is no significant difference between the robot's behavior in simulation, as long as it has been appropriately set up, and the robot in real life. The only actual separating parameters come to the robot's speed to perform some tasks due to the difference in interprocess communication. However, this issue has been resolved by tuning some of the parameters in the navigation stack. This further reinforces the validity of simulation in adequately assessing a robot's behavior and gives us a good idea of the navigation tasks that can be performed inside of a simulation.

Some of the other studies are more specific and use multiple different types of robotic systems, such as manipulators [22], a UAV[23], and even more specific cases in a golf cart [27], a crawler[25], and a simulated goal-keeper in [26].

One can see that the capabilities of simulation, in specific Gazebo allowing for the re-use of code, have a significant impact on the time it takes to design and create a robot as well. The authors in [22] talk in their conclusion about how a robot used to take them even a couple of months to create, but Gazebo reduced the time to less than two. That is because, with the aid of simulation, developers can test the software while they are designing the hardware, allowing for multitasking. This heavily decreases the amount of dead time a team might have when building their robot.

A particularly interesting use of simulation is discussed in [25], using the software to help design robots for rescue missions. The main focus of their study was modeling the hardware inside of Gazebo, which is another area covered by the software. That is the ability to import and edit 3D meshes and models created either inside or outside the program. Gazebo is fully capable of loading external 3D objects that can then be linked to a gazebo object. After the 3D model had been created, the task was to animate and stimulate each part of the model, which was complex in its manner. However, it can be solved with integrated tools available in ROS, such as splitting models in joints and assigning each joint to its interfaces that would control the angle and velocity. This method of controlling a simulated robot is the primary method used in ROS because of the need to account for each independent movement that a robot would be capable of performing. Some models, like self-driving cars, can only move their own bodies in unison. At the same time, the smaller robot can rotate in place, or even UAVs that might be capable of moving each of their propellers at a different speed and orientation to control their unique 3D navigation.

From the available work done in the field, I can see that simulation is an invaluable asset for robotic development and can also be used to test a broad array of problems. It allows for a valid way of testing the

behaviour of a system in different situations and helps with their final designs. Simulation is uniquely potent in simulated navigation tasks, with the help of multiple tools designed explicitly for that. Furthermore, due to the inexpensive and repeatable nature, testing is also easily facilitated inside the simulation, allowing us to look for cases that might be met in a quotidian mission. However, one might glance over due to the complexity and risk of testing the model in such an environment.

3 Methodology

3.1. ROS, Gazebo, and the Simulation World

Before I can talk about the simulation world, I have to expand on the features and nature of ROS [16] and Gazebo[17]. Gazebo is part of the Payer Project. It allows for the simulation of robot movements and sensing in 3D environments that can be as simple as mazes or dorm rooms or as complex as multiple floor buildings. It works on a Client/Server architecture and a topic-based Publish/Subscribe model, allowing for interprocess communication.

Gazebo consists of two interfaces, one native Player interface, and the client makes its data accessible through shared memory. Every object in a simulation is associated with one or more controllers that, as the name suggests, are used to control that object's state and actions, including the robot and every part of the environment. Their data is then published into the shared memory using interfaces. Those interfaces can have their data read by different processes, allowing for interprocess communication between the software used to control the robot and the actual simulation inside Gazebo, independently of the programming language used by the user.

Another functionality of Gazebo is the ability to place the Client and Server on different devices. This way, by communicating with ROS, the software can control the real hardware, the robot, and the simulated device simultaneously. This provides a way of creating more realistic and effective simulations, providing a well-equipped tool for testing robotic systems.

While Gazebo provides a tool for creating, altering, and controlling objects inside a simulation, ROS is the actual application doing most of the thinking necessitated by the robot. ROS is a collection of libraries, drivers, and tools used for the development of robotic systems. It has numerous packages, and just like Gazebo, it is capable of interprocess communication. Another similarity is the ROS executable, called a node, has the same Publish/Subscribe model. The communication data is offered in the form of a topic. Each publisher node can publish data to one or more topics, and each subscriber is able to read the data available inside those topics. ROS is mostly language-independent, so applications or nodes can be implemented in a multitude of programming languages such as C++, Python, Lisp, and even JAVA.

To allow for the effective development of robotic systems, ROS packages include several highly important, already implemented

packages, such as sensor drivers, navigation tools, mapping tools, path planning algorithms, interprocess communication tools, even visualizing tools, that would let us see what the robot camera is seeing at a certain point. These two software together, with ROS acting as the development tool for the robot and Gazebo as the middleware between ROS and a graphical, physics-based world, make for a high-performance way of implementing and testing such robotic systems.

Skipping to my use of those tools, I have created a simulation that has been designed to resemble a racetrack, containing a variety of straight lines so that it tests the navigation of the robot to the fullest. One part of the track is particularly straight, while another is made of multiple zig-zag-like motions so that while testing inside it, the results should be more conclusive.

The simulation world was created from scratch using tools available in the Gazebo software. This has been done both using the graphical interface of the software as well as editing the markup file that is generated once a world has been created.

With the graphical interface, using the available Model Editor tool, a set of the wall have been drawn resembling the schematic of a racetrack. Each wall has been individually modified to make it smaller so that the world does not look disproportionate to the robot. Moreover, a set of alternating white and red paint textures have been applied to said walls for aesthetic purposes.

Once the model has been created, saved, and placed, the world has been saved into a file with a .world extension, which can be edited by Gazebo or a text editor.

This file, working as it has been written in a markup language, contains the details of each of the objects, from the 3D model mesh applied to them to their position, orientation, and collision boxes.

3.2. The robot

The robot I have used in the simulation is turtlebot3. It is a low-cost robot created for the purpose of increasing the spread of robots by giving customers a fully functional, affordable, and modular device having all the features necessary for robotic development.

As the scope of this thesis is only about simulation, a physical copy has not been used, but the team behind turtlebot provides a fully integrated ROS version of it. It comes in the form of a package containing multiple versions of this robot and simulated functions and multiple available simulations and examples of the robot's capabilities.

The robot model that I used is called burger, and it is based on a real-life product with the same name. It is a two-wheeled robot; it has a taller frame than the other variants. It is a little bit slower, but it contains a front-mounted camera and Kinect sensor, which are necessary for autonomous navigation. The robot is SLAM capable, which is a navigation algorithm discussed further in the following subchapter; and is relatively small and slow, and it is capable of rotating in place, features that will be of high importance while going through an obstacle track.

The robot is abstracted for being used in ROS through a 3D model and URDF file. URDF is a ROS-specific format based on XML, which describes all the robot elements, in my case, the frame, the two wheels, the front-mounted camera, and the front-mounted Kinect sensor.

3.3. Navigation

For navigation, turtlebot uses a SLAM algorithm. SLAM, which stands for Simultaneous Localization and Mapping, is used in various applications such as self-driving cars, underwater exploration, and even aircraft and UAVs. The SLAM method used by turtlebot is G mapping which relies on a particle filter. The purpose of the particle filter is to compute posterior distributions of the Markov process or if there are any noisy observations. The particle filter uses approximation for prediction and updates. The samples from said distribution are given as particles; each particle has a weight that represents its probability of being sampled.

The G mapping algorithms available in ROS provide a laser-based SLAM as ROS node code slam_gmapping. This module creates a 2d occupancy grid map from the collision of the laser and the pose data gathered by the mobile robot.

To create the map, the robot, once initialized in the world, has to be able to provide odometry data, which it can do as it is mounted with a Kinect sensor in the form of a laser range-finder. This laser captures depth data from the environment by bouncing into it, determining what are solid walls. ROS then converts this data into a 3D point cloud, which is further converted into a 2d laser scan. Once the required parameters are successfully provided to the ROS node, a map will be generated in RViz.

Rviz is a ROS application that lets us see the perspective of the robot. The way the map looks in RViz is shown in figure nr. 3-1.

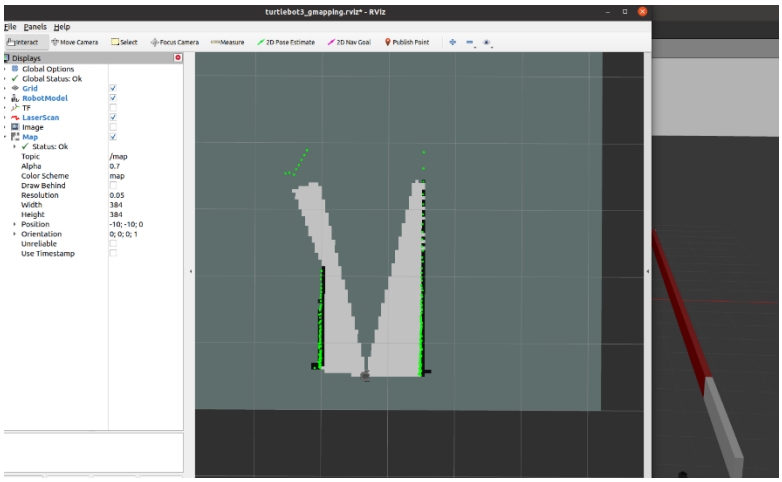


Figure 3-1: RViz in action

Usually, to start creating this data, the robot would be moved manually using the teleop_key package, which uses keyboard inputs, w, a, s, d, x, allowing us to change the angle and velocity of the robot. My model of turtlebot, the burger, can move at a maximum velocity of 0.22 when it comes to simulation.

Once the map has been created, it can be saved using the map_server package, which is then ready to be used by the robot for autonomous navigation.

The robot also needs to localize itself in the environment, for which the map, as well as odometry data, are essential. This is fed to the turtlebot, which uses a variant of the Adaptive Monte Carlo Localization. AMCL is a probabilistic localization system used specifically for robots moving in 2d. The algorithm follows the position of the robot against the map. Turtlebot also uses KLD (Kullback-Liebler divergence) sampling to adapt for sample size.

All of this can be visualised using the rqt ROS package, a debugging tool that creates a graph of all the running nodes and processes and the way they interact and communicate with one another.

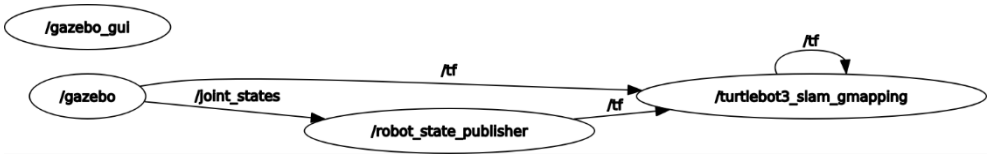


Figure 3-2: SLAM nodes interacting

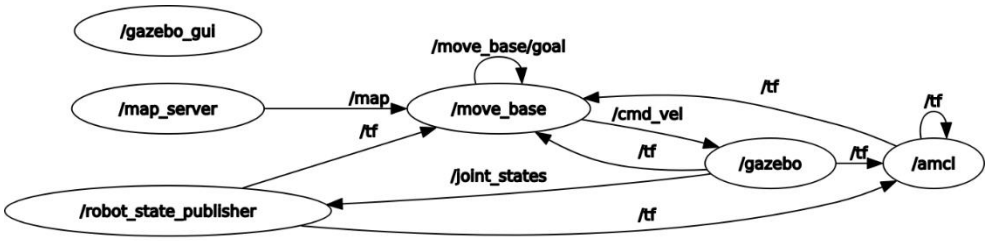


Figure 3-3: AMCL nodes interacting

Once the mapping and localization have been prepared, the robot can start to navigate autonomously. This is being done by ROS, which takes TF and laser scan data and its estimated position from the robot. The information it gathers about obstacles in its path is then fed into two cost maps, one global and one local. The global one is used to make a long-term path for the entire map, while the local one is used for obstacle avoidance and planning a local path. The robot does the planning, and then velocity and direction commands are given to the robot controller.

This is done by running an assortment of packages available in ROS and turtlebot. The initial mapping has been done using the SLAM package available in turtlebot, which will start the corresponding ROS package. In this stage, the robot has been moved around using the teleop package, which allows for manual movement from the user input. Those inputs are w for accelerating forwards, x for doing it backward, a for applying angled velocity to the left, d to the right, and s for immediately stopping the robot.

A partial map has been created using the SLAM node because the straight part of the racetrack provides a challenge for the algorithm, as it loses reliability when an environment is made of long parallel corridors. This happens because the features used for tracking by the algorithm are absent in such places. This is called a tracking error, and it can be solved in multiple ways. The one I chose was by completing the map manually after I got it into a satisfactory state. The partial map has been completed using the GIMP image editor; once the map is saved, it is treated mostly as a regular bitmap image. Manually drawing on the map will have the same effect on the path-planning algorithm used by ROS as automatically mapping it with SLAM.

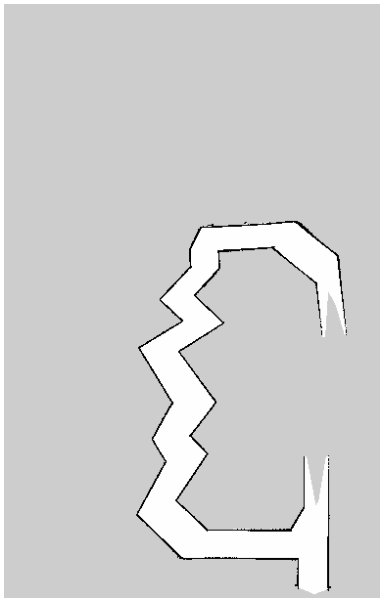


Figure 3-4: Incomplete map

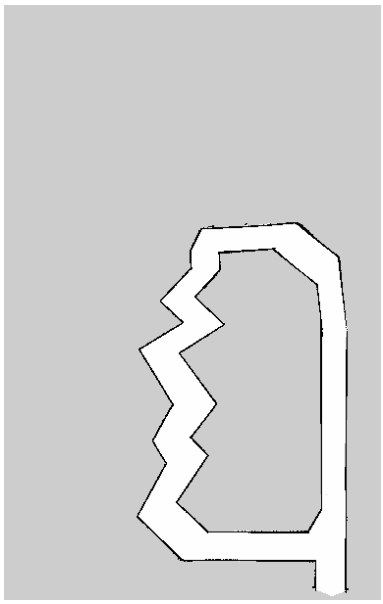


Figure 3-5: Map completed by hand

Once the map has been completed optimally, it can be used with the navigation package. This package follows the principles of AMCL described above and allows the robot to move to specified points by creating a functional path.

3.4. Robot Behaviour

After autonomous navigation had been set up for my robot, I needed a way to make it run laps across the track without requiring me to set goals for it every single time manually. This has been done using another feature of ROS called an action, which is implemented using the actionlib stack. Actionlib allows us to interface with tasks in a programmatical way by setting up scripts to act as nodes.

The action I created is available in the move_bot package, and it started as a c++ script that will interact with move_base, the node used for path planning in the amcl package, which will automatically give it points chosen by me on the map, and feed them to the base as navigation goals. This will make the robot follow the same points in a loop, simulating the constant lapping of the track. This node is called move_base_node, and it is available in the src folder of the catkin move_bot package.

With the use of this node, I can test the initial capability of the robot to navigate through the clear map, which for now contains no obstacles. As it can be seen, once the robot localizes itself in the world, it has no issues going through the track in an expedient matter, showing us that the ROS navigation stack has been implemented correctly in my package and that I can go further testing to check the more advanced

capabilities are necessitated for the mission created. After the initial setup has been saved and thoroughly tested, an expansive set of rules have been created and implemented to satisfy the mission I have developed. The interaction of the node with the navigation stack can also be mapped using the rqt package.

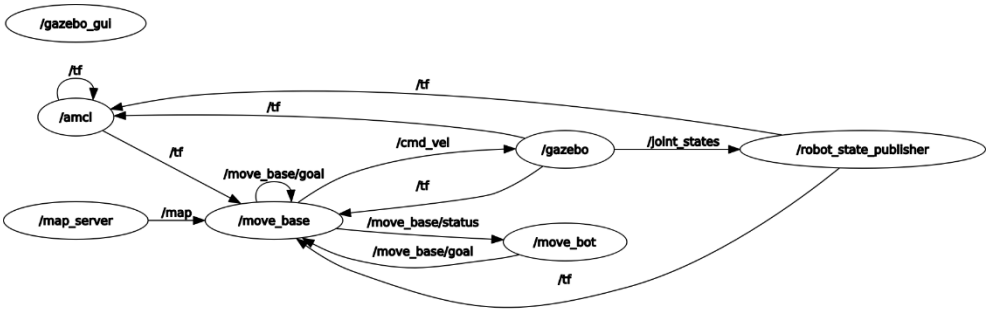


Figure 3-6: Nodes interacting AMCL+move_bot

3.5. The mission

The fundamentals of the mission can be abstracted as giving a series of checkpoints that are needed to be reached in a certain amount of time, which will count as a pass; otherwise, it will count as a fail. So the following list of objectives and testing requirements have been created in order to properly measure the effectiveness of my robotic system in completing its given task:

a. Can the robot properly map the environment?

b. Can the robot properly localize itself?

c. Can the robot avoid the obstacles?

d. Can the robot properly follow the checkpoints?

e. Can the robot do all of these in the required time?

3.6. Generative Testing Implementation

As discussed in one of the studies from the literature review, there are multiple ways of designing tests, of which I have decided to use the generative testing model. Generative testing implies the use of procedurally generated test cases instead of hardcoded ones.

The advantage of those generative tests is that they require a really small amount of data and can create a really broad array of test cases. Those test cases can help us identify fails and corner cases that would otherwise come unnoticed because the exact situation might not have been implemented.

I have decided to implement these generative case testing by randomising sequences of points that would act as goals for the navigation stack that controls the robot and randomizing the distribution of the obstacles on the map. This makes it so that the robot is forced to use its capabilities in a broad spectrum of cases, where obstacles might only leave it a narrow path, where the obstacles block a certain path completely, or when the amount of stops that the robot needs to take is fairly low or fairly high, or even when all those are spread apart in a non-optimal way.

Problems specific to this type of implementation are the modeling of what one might consider a valid and sufficiently important test, which will require the development of a specific model for creating these test worlds or test cases. Moreover, the second problem is that most test cases are bound to be similar to one another. Hence, the hierarchy of ratings in which tests have already produced the same results needs to be considered, too, since running the simulation takes a long amount of time (a couple of minutes).

The model developed for creating the world is first by adding randomness in the world generation. This was achieved by manually editing the markup file that makes up the Gazebo world and implementing the <population> tag after a certain number of different obstacles have been defined.

The obstacles, numbering three, have been chosen from the available gazebo models and edited by hand to fit the scale of the world. These models are a y-axis cinderblock, an x-axis cinderblock, and a construction cone. They have been used for their silhouette rather than their appearance, as each poses a different type of obstruction to the path. The cinderblocks are narrower and longer, making it so that they create a line-like block on the axis chosen, and the construction cone creates point-like obstacles, as it is fairly circular and pretty small. Models have been edited using the Model Editor in Gazebo; first, on the cone case, the mesh size and boundary box size have been decreased since they would have been too large for the size of the track. Moreover, another modification has been done to one of the cinderblocks that needed to be rotated on the model basis since rotating it in the world while the <static> tag is set to true is impossible.

The static tag has been used since I do not want the robot to be able to move along those obstacles since the algorithm will already try to avoid them regardless.

The feature allowing us to spawn those obstacles at random is called a population in Gazebo, which is a tag that can be used inside the markup file and not in the GUI editor, which will spawn a chosen number of clones of chosen 3D model, and distribute them in a box

starting from a middle point. The distribution for these populations can be random, like the one I have used, or more orderly variants like a linear distribution, a uniform one, or a grid pattern.

Model counts have been chosen for the smaller cone obstacles and the bigger cinderblocks, respectively. They have been distributed across the legs of the racetrack so that they will not spawn in parts of the map that are inaccessible by the robot. This will assure me that a different environment is being created every time the world is launched.

The second method used for adding a broader scope to my generated tests was by randomly feeding several checkpoints to the robot. This has been done by modifying the move_robot node. First, more than 100 points were sampled from across the map. Those points have then been ordered, split by the portion of the map, i.e., the first section, first corner, the zig-zag section, and the fourth section, and then saved into arrays inside the c++ script. Then, using the random library from c++, a random number of those points are fed into a loop and published to the controller as the goals for the move_base. This will ensure that the robot has different, automatically generated paths that it needs to follow, which will be given to it in a programmatical way.

Next is an image of the track, with obstacles added at random alongside it.

Figure 3-7: The simulation map

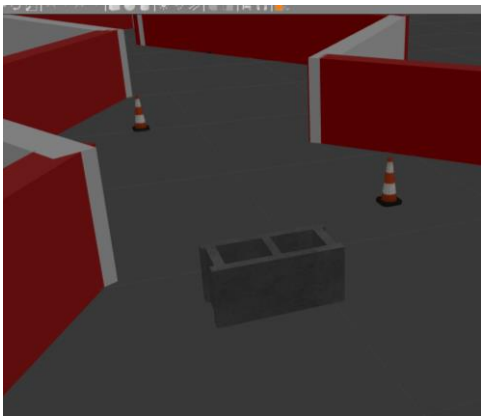


Figure 3-8: Obstacles

4 Results

4.1. Setup

Following the standard practice outlined in [28] for empirical software engineering, I am evaluating the capabilities of my robot using the case study outlined in Section 3.1. Using the simulated world, I am trying to rate its ability in each of the proposed requirements from Section 3.5.

All experiments have been run on a laptop's Intel Core i7-10750H, with an RTX 2060 graphics card, running on ROS Noetic and Gazebo version 11, on native Ubuntu OS 20.04.

The metrics I am using for analysing the performance of the robot are the time T it takes to finally conclude the mission and N, the number of checkpoints it needs to reach along with the mission. I am looking for a higher number of T in proportion to N, the number of checkpoints. The higher the T, the higher the actual performance of the robot since it was able to calculate and re-address its path in a better time. I am assuming that a multitude of fail points can occur while N is the highest because the robot has to continuously update its navigation goal, which will in press update its path as well, taking time as more computational power is required. With this in mind, T has been calculated using the following formula:

T = T finish – T start,

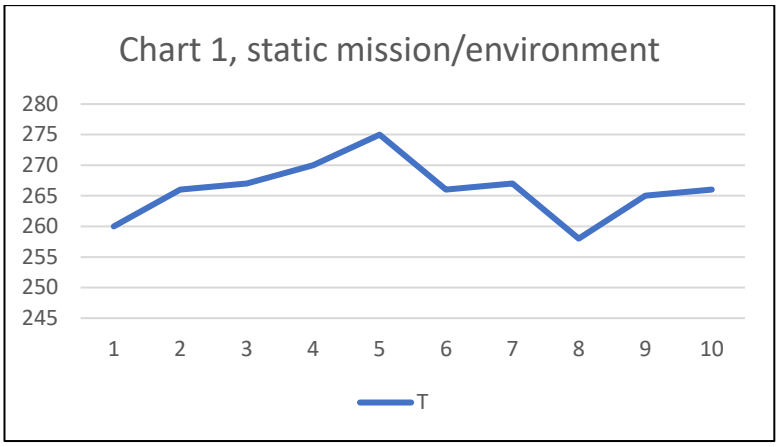
Where T is the amount of time the robot needed to reach the final checkpoint in seconds, going through each of the previous ones, where reaching them was possible, T finish being the final time when the robot reached the start line, given to him as a separate checkpoint, and T start being when the loop inside the node starts running its increment.

For random checkpoints, the decision has been made that a checkpoint should not be behind the previous checkpoint so as not to mess with the order in which the track is being navigated.

4.3. Data and analysis

The number of checkpoints reached and the time it took to finish or fail the mission can be seen in figure 1. This experiment has been done on a completely empty map to focus the testing on the speed of processing that the AMCL algorithm is capable of reaching.

As we can see in figure 1(Chart 1), it takes an average of around 266 seconds for the robot to complete an entire lap if it is given the six checkpoints and no obstacles are placed on the track.



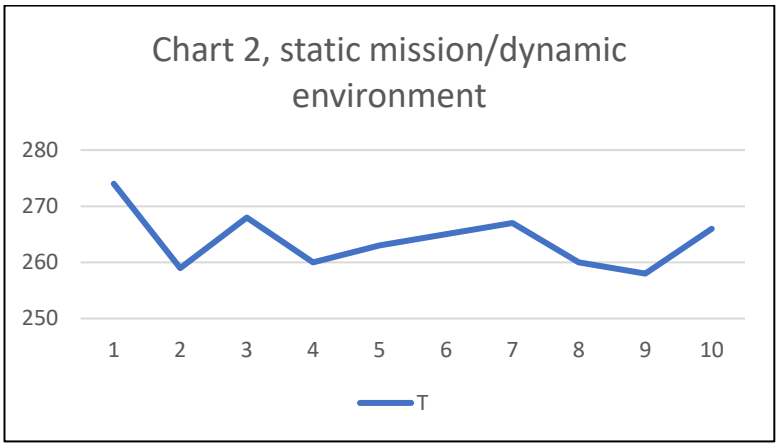
On average, the robot takes around 1-3 seconds to recover and start going to its next position.

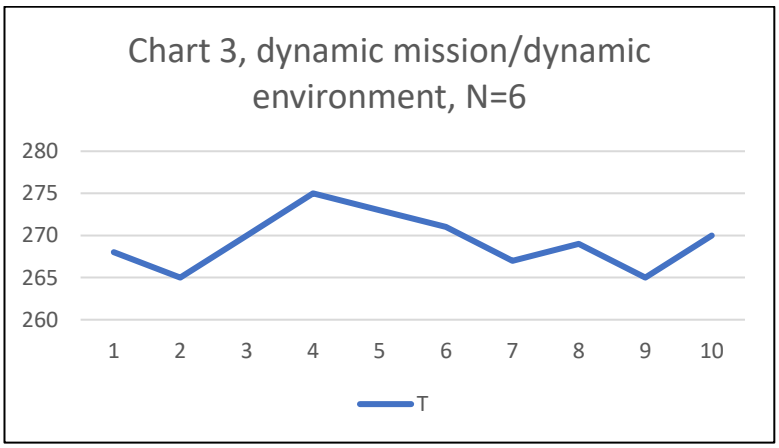
Adding the obstacles inside the track yet keeping the same course of checkpoints shows no massive change in the amount of time that it takes to finish one lap. But now, with some noise added, we can first see that not all courses can be completed. Those have to be set apart as edge cases. Those courses that cannot be completed come in the way of obstacles blocking a lane and not leaving enough space for our robot to pass. As this became a prevalent issue, the number of obstacles was drastically reduced during the design phase, where I first started with a total of 45 cones alongside the track and 20 cinderblocks of each rotation. The final amount was closer to 30 cones and ten cinderblocks.

When I have found that a mission cannot be completed due to these obstructions, I close the simulation and re-run it, disregarding the results when it comes to time. But, these courses are not completely useless, as they first show that the obstacle detection capability of our robot is fine-tuned, as it can detect those blocked paths and it will not try to pass through them. This, in a real-life mission, would show validates the first principle necessary in the testing of robots, mentioned in [21], that being the safety. As the robot will not hit any obstacle, even if that is the only possible course of action to further its goal, that means it will not inflict any damage that could have been caused by the collisions.

Once the obstacles had been lowered, and most of the maps were passable, I started to test the results of the robot in this new, modular environment. With the use of AMCL, the robot is capable of tracking the map as it goes, which will modify the costmap without modifying the local map, that is fed into the navigation node. This way, the robot knows that new objects have been added to the environment without losing the ability to localize itself and navigate, as it does not have to redraw the map from scratch every time.

Speaking of AMCL, I can see one problem with the self-localization part of the algorithm. The robot is fully capable of localising itself on the map once a point closer to its starting position is given to it through RViz. But, at the start of every simulation, the navigation node will place the robot at the point (0,0). As the robot moves, it drifts further apart from the track, and since no features can be seen, as the world around it is empty, it will never be able to localize itself. As I have mentioned, this issue is easily solved by feeding the initial position to the navigation stack. No major errors have been found in regards to localization afterwards, even if the map is not a perfect representation of the environment. The slight differences in the map do not seem to affect the ability of the robot to navigate.





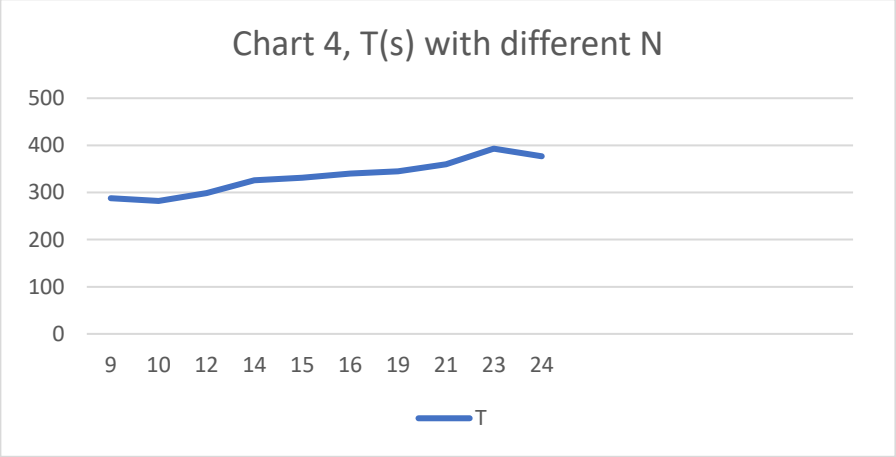
In the previous figures, we can see the performance of the robot once obstacles have been added. One thing that struck me as counter-intuitive is that there is no major difference between the completion time T for N=6 once the robot needs to also avoid the objects in its path. No major difference has been recorded, only marginally higher

times. This can be interpreted as the fitness of the actual obstacle avoidance part and path planning of the navigation stack. The robot is not only fully capable of avoiding obstacles, but it is also mostly capable of going around them. This happens in the path planning part of the node, where the amcl algorithm will create a path that will go to the set goal by trying to avoid any obstacles in its way.

This path can also be seen inside RViz, as a red line, whose starting point is the position of the robot, and the endpoint is the actual goal that needs to be reached.

Implementing the final step of the mission, that being the actual random checkpoints, alongside a random total number of checkpoints N, we can finally see some big variance. As previously expected, the higher the N, the higher the total time T it takes the robot to navigate the track. But another variable has to be added in this case, that being the number M of missed checkpoints. I have decided that the robot can miss a number of checkpoints as long as it still finishes the track, but in the end, when the fit is being calculated, it has to be penalised for them. Some misses are actually false positives when a checkpoint is spawned right inside an obstacle. A pass for this final mission will be if the robot is capable of finishing the track in the amount of T time of no more than 270 seconds + 10 × (N-6).

Analysing the data seen in the next table, we can see that, indeed, the higher the N, the higher the T. This is due to the robot needing to calculate a path each time a new goal has been set. Some of this can get fairly lengthy when there are a lot of checkpoints very close to each other.



There are moments when the robot will get stuck. They will often result in the robot trying to recover itself without succeeding before it has to backtrack. Once backtracking has been started, the robot will try a new

path that, most of the time, will be successful, even if longer than the first invalid path.

In the end, I would conclude that the fitness of the robot is passable, as it can navigate to the end of the track the vast majority of times. It does always avoid the collision with an obstacle. And most of the time, it can calculate a valid path for moving forward, and it does not get stuck all that often. Most of the robots were capable of also finishing the mission in time.

4.4. Unexpected Agent Behaviour

With the navigation algorithm being non-deterministic in nature and the environment, as well as the mission containing a fair amount of noise, cases, where the robot will act in an unexpected manner, was bound to happen. I will further discuss some of the corner cases found.

From multiple runs, I was able to form some patterns. One of them is when the obstacles and the walls in front of the robot form a corner.

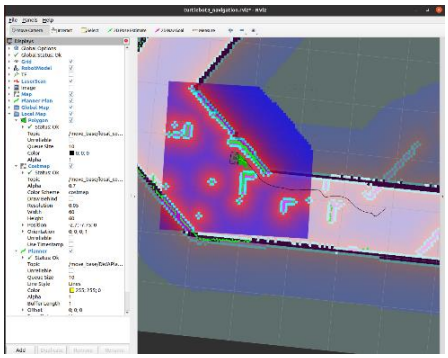


Figure 4-1: Corner case

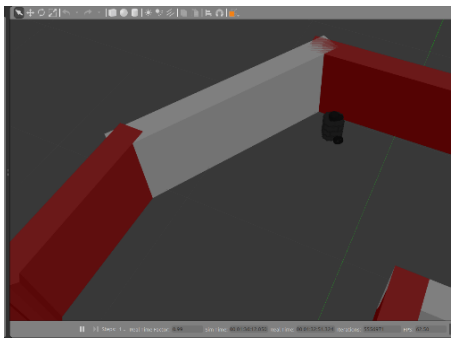


Figure 4-2: Corner case 2

The robot would attempt to pass through that corner at first, but it will get stuck, and path planning will stop working once the robot is there. My guess for why this happens is that the localization algorithm might think that the position of the robot is inside one of those obstacles in the formed corner, as seen in figure 4. I have tried to check by feeding different goals into the planner, but even telling the robot to just go back to where it came from will have no effect. The only way to keep the robot moving is to manually change its position. This can be done in the simulation by changing the x or y coordinates.

One particular case of the following happened when the robot got stuck in the actual corner of the map. From multiple runs, this seems not to be a recurring issue if the target checkpoint has been drifted too far into the corner of the track. The robot would be stuck, acting as if

the walls block its ability to move, even though it could easily just move back or rotate. This can be seen in figure 5.

Another unexpected behaviour I have found while running the test is that sometimes the robot will refuse to backtrack. This happens when the goal is close to the position of an obstacle that is blocking the path of the robot. This is usually solved once the path moves further away from that obstacle. In that situation, the robot will finally attempt to backtrack and find a different path to be able to reach its goal. This can be seen in figures 6 and 7.

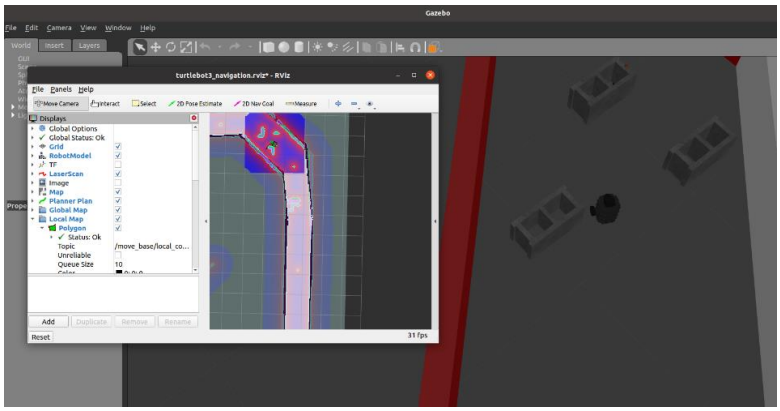


Figure 4-1: Robot getting stuck

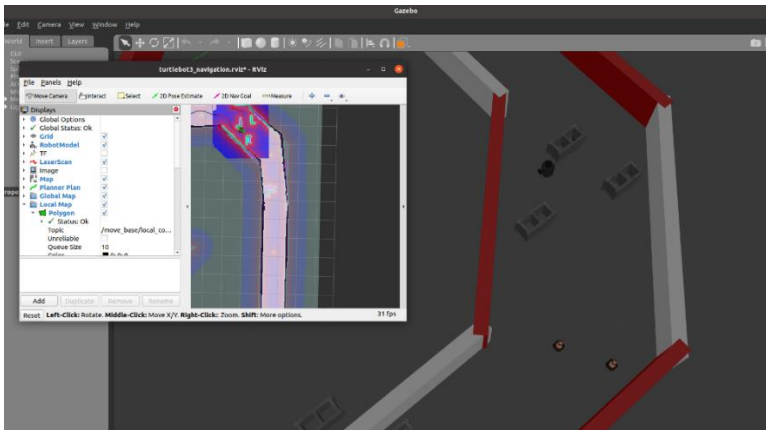


Figure 4-2: Robot getting unstuck once checkpoint changes

And for final unexpected behaviour, due in fact to the non-deterministic nature of the agent, sometimes the robot would fail a task that it was capable of doing the last couple of runs. This has happened in the obstacle variant of the track. It does not happen often, but it poses a threat to the fitness of the robot.

5 Conclusion

My dissertation has sought to create a robotic simulation and test the capacities of the software. I have worked from available literature in the field and used some of the most readily available and feature-packed software in creating my mission. The mission was for a robot to finish an obstacle course in a certain amount of time while also visiting all the checkpoints that were given to it. This was done in order to stress-test the capabilities of the robotic model and prove that simulation is an invaluable tool when it comes to robot testing.

My analyses have shown that the software is more than capable of creating a believable but low-fidelity world that can be used as a test environment. Not only was it possible to create a simulation from scratch and use it as the footprint for our mission. The integration between Gazebo and ROS also allowed me to easily implement the tasks of the mission. They also facilitated all tools necessary to conduct testing.

Data and extensive tests have shown that the robot is clearly capable of navigating inside the simulation and actually finishing the objectives of the mission. It can reach the finish line in the specified time in most cases. Answering the questions posed in the 3.5 section, yes, the robot is capable of localization, moving, avoiding obstacles, reaching checkpoints, and doing all of that in a timely manner.

The testing has also shown some corner case behaviour that poses future problems to be addressed in any further work. The simulation has vastly helped with collecting the data and finding the corner cases. Because of the ease with which I could implement variety and fuzz in the mission.

Further study in the field might be accomplished in a couple of distinctive ways. The first method might be using the same model and deploying it in different missions or further developing the mission used as a case study. This will help collect even more data about the fitness of its algorithms and further help improve the development of the model. The second possibility would be using a custom model or further developing the one in this paper. A model can be specifically created with the goal in mind of being able to complete the task at hand in the most expedient manner. Both methods can even be attempted at the same time, where a better model is being designed for the task, and then that model is being tested in a broader array of environments and missions. The acquisition of real hardware is also a tool that can help validate the results of any further study in the field.

Simulation in robotics might be a relatively new concept, but it is gaining increased use for a good reason. Robots, in their non-

deterministic nature, need to be heavily tested before being dropped in the field, and simulation is the best way of achieving that. It can be used to the advantage of the developers. It highly reduces the time needed to design and test a model, with very few trade-offs.

Bibliography

[1] Harbin, J. R., Gerasimou, S., Matragkas, N., Zolotas, A., & Calinescu, R. (2021, August). Model-driven simulation-based analysis for multi-robot systems. In MODELS 2021: ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)

[2] Secure and Safe Multi-robot Systems (SESAME) H2020 EU Project,york.ac.uk , Available: https://www.cs.york.ac.uk/research/projects/sesame/[Accessed: 20 October 2021]

[3] Safe Airframe Inspection using Multiple UAVs (SAFEMUV),york.ac.uk Available: https://www.york.ac.uk/assuring-autonomy/demonstrators/remote-inspection-using-drones/[Accessed: 20 October 2021]

[4] Robotis, Turtlebot3, emanual.robotis.com. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/[Accessed: 13 February 2022]

[5] Robotis, Turtlebot Autonomous Driving, emanual.robotis.com. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/autonomous_driving_autorace/ [Accessed: 13 February 2022]

[6] C. E. Aguero, N. Koenig, I. Chen, H. Boyer, S. Peters, J. Hsu, et al., "Inside the Virtual Robotics Challenge: Simulating Real-Time Robotic Disaster Response", IEEE Trans. Autom. Sci. Eng., vol. 12, no. 2, pp. 494-506, Apr. 2015.

[7] Defense Advanced Research Projects Agency (DARPA), DARPA Robotics Challenge, theroboticschallenge.org Available: http://theroboticschallenge.org/. [Accessed: 14 February 2022]

[8] Webots, 2022, [online] Available: https://www.cyberbotics.com [Accessed: 14 February 2022]

[9] Robot Virtual Worlds, 2022, [online] Available: http://www.robotvirtualworlds.com/ [Accessed: 14 February 2022]

[10] National Instruments, What Can You Do With LabVIEW?, Available: http://www.ni.com/labview/why/. [Accessed: 14 February 2022]

[11] Japan's National Institute of Advanced Industrial Science and Technology, OpenRTM-aist, openrtm.org, Available: http://www.openrtm.org/openrtm/en/node/629. [Accessed: 14 February 2022]

[12] I. Chen, B. MacDonald, B. Wunsche, G. Biggs and T. Kotoku, "A Simulation Environment for OpenRTM-aist", Proc. SII 2009. IEEE/SICE International Symposium on System Integration, pp. 113-117, Nov. 2009.

[13] Robocup, Robocup Rescue Mission, https://rescuesim.robocup.org, Available: https://rescuesim.robocup.org [Accessed: 14 February 2022]

[14] Epic Games, Unreal Engine 5, unrealengine.com, Available: https://www.unrealengine.com/. [Accessed: 14 February 2022]

[15] OpenHRP3, About OpenHRP3, http://fkanehiro.github.io Available: http://fkanehiro.github.io/openhrp3-doc/en/about.html. [Accessed: 14 February 2022]

[16] Open Source Robotic Foundation, ROS/Introduction, wiki.ros.org, Available: http://wiki.ros.org/ROS/Introduction. [Accessed: 14 February 2022]

[17] Gazebo, Gazebo Home, gazebosim.org, Available: http://gazebosim.org/.[Accessed: 14 February 2022]

[18] A. Afzal, C. L. Goues, M. Hilton and C. S. Timperley, "A Study on Challenges of Testing Robotic Systems," 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), 2020, pp. 96-107, doi: 10.1109/ICST46399.2020.00020.

[19] A. Afzal, D. S. Katz, C. Le Goues and C. S. Timperley, "Simulation for Robotics Test Automation: Developer Perspectives," 2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST), 2021, pp. 263-274, doi: 10.1109/ICST49551.2021.00036.

[20] Laval, Jannik & Fabresse, Luc & Bouraqadi, Noury. (2013). A methodology for testing mobile autonomous robots. Proceedings of the ... IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE/RSJ International Conference on Intelligent Robots and Systems. 1842-1847. 10.1109/IROS.2013.6696599.

[21] Robert, C., Sotiropoulos, T., Waeselynck, H. et al. The virtual lands of Oz: testing an agribot in simulation. Empir Software Eng 25, 2025–2054 (2020). https://doi.org/10.1007/s10664-020-09800-3

[22] W. Qian et al., "Manipulation task simulation using ROS and Gazebo," 2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014), 2014, pp. 2594-2598, doi: 10.1109/ROBIO.2014.7090732.

[23] Johannes Meyer, Alexander Sendobry, Stefan Kohlbrecher, Uwe Klingauf, and Oskar von Stryk. 2012. Comprehensive simulation of quadrotor UAVs using ROS and Gazebo. In Proceedings of the Third international conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR'12). Springer-Verlag, Berlin, Heidelberg, 400–411. https://doi.org/10.1007/978-3-642-34327-8_36

[24] W. Yao, W. Dai, J. Xiao, H. Lu and Z. Zheng, "A simulation system based on ROS and Gazebo for RoboCup Middle Size League," 2015 IEEE International Conference on Robotics and Biomimetics (ROBIO), 2015, pp. 54-59, doi: 10.1109/ROBIO.2015.7414623.

[25] Sokolov, Maxim & Gabdullin, Aidar & Afanasyev, Ilya & Lavrenov, Roman & Magid, Evgeni. (2016). 3D modelling and simulation of a crawler robot in ROS/Gazebo. Proceedings of the 4th International Conference on Control, Mechatronics and Automation. 61-65. 10.1145/3029610.3029641.

[26] 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, abs/1608.05742.

[27] Ilya Shimchik, Artur Sagitov, Ilya Afanasyev, Fumitoshi Matsuno and Evgeni Magid,Golf cart prototype development and navigation simulation using ROS and Gazebo, MATEC Web Conf., 75 (2016) 09005.DOI: https://doi.org/10.1051/matecconf/20167509005

[28] Lionel Briand and Yvan Labiche. 2004. Empirical studies of software testing techniques: challenges, practical strategies, and future research. SIGSOFT Softw. Eng. Notes 29, 5 (September 2004), 1–3. https://doi.org/10.1145/1022494.1022541