

Diploma Thesis Self-driving (Autonomous) Car Simulation in Ros System

Zainab Nadhim Jawad (✉ zainab.much@gmail.com)

University of Pecs

Research Article

Keywords: Autonomous, ROS, Gazebo.

Posted Date: December 1st, 2023

DOI: <https://doi.org/10.21203/rs.3.rs-3649061/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations:

Competing interests: The authors declare no competing interests.

University of Pécs
Faculty of Engineering and Information Technology
Computer Science Engineering Master

DIPLOMA THESIS

Self-driving (autonomous) car simulation in ROS system

Author: Zainab Jawad
Supervisor: Dr. Balázs Tukora

Pécs
2019

Abstract

Autonomous car development process considered as a hard task that required to test and evaluate the proposed system, for this reason, the simulation considers as a powerful tool, as it provides testing the system components interaction, evaluates the suggested autonomous solutions with no risks. This thesis presents a simulated self-driving car robot under ROS and Gazebo simulation environment. The complete task involves creating the robot model, integrating the simulated perception system and enable autonomous navigation inside a generated 2D map of the virtual world. The results emphasize that this method of compiling programming and simulating provide a reliable piece of work which its outcomes have a direct impact on self-driving cars development.

Keywords: Autonomous, ROS, Gazebo.

Table of Contents

Introduction.....	1
1. Research Problem and Methodology	2
1.1. Specification of the problem	6
2. Prior research and background	8
2.1. Analysis of the problem	8
2.2. Literature review	17
2.2.1. Path planning	17
2.2.2. PID steering control.....	19
2.3. Requirements derived from the literature.....	21
2.4. Selection of the solution technique	22
2.4.1. ROS (Robot Operating System)	22
2.4.2. Usage of ROS for autonomous solutions	26
3. The Design implementation.....	37
3.1. Detailed specification of the solution.....	37
3.1.1. Model configuration	37
3.1.2. Perception systems	39
3.1.3. Navigation system	41
4. Design work phases and problems.....	48
4.1. Simulate the model.....	48
4.2. Simulate the sensors	50
4.3. Navigation of the simulated robot.....	51
4.4. Summary of the problems during the work and experiences	59
5. Evaluation	61
5.1. Explanation of realization	63
5.2. Analysis of the realization.....	63
6. Tests and results.....	65
7. Discussion.....	71
8. Conclusion	72
9. Summary.....	74
References	

Introduction

The global evolution had been managed by a combination of rapid technological improvements in various aspects of industries. The autonomous cars rise up the interest within the robotics industry and have been become the focus of many robotic competitions around the world. Expectations state that the coming era will be the autonomous driving era and car power will be measured by the power of the vehicle's sensors, and software [1].

Hence, the public interest in autonomous driving is rapidly increasing with the autonomous technical improvements that removed the driver's errors. However, the practical tests indicate that there were some challenges required more analysis and tests in order to offer an autonomous car with maximum safety to the city roads. This research motivated to conduct by global interest in autonomous driving.

This thesis objective is to design a self-driving car robot that effectively finds a path from its position to the destination position enabling the robot to achieve accurate movement along the track with obstacle avoidance. Therefore, the simulation environment must be used since it will reduce the cost and risks during the practical development. Hence, the specific problems that must be analysed and answered concentrated about how the robot sense its environment and planning its movement. In addition to deep investigation into how ROS and gazebo simulation environments used to develop an autonomous robot simulated model and to identify benefits and challenges of such collaboration. The outcome from this thesis work will have an enhancement to the autonomous car development in both scientific researches and the automotive software industry.

This Thesis begins by defining the research problem and research questions, and then demonstrated the selected research methodology DSR. A literature review is performed to gather background knowledge. The literature review's results also depict the possible solutions to the problems. Specifically, the core part of the thesis is the in-depth description of the performed simulated robot and its navigation software design, followed by the tests and results. Finally, the achieved tasks summarized.

1. Research Problem and Methodology

The selected research method was DSR. After gaining knowledge about the research topics from the literature, related projects in the development of autonomous cars in which ROS was used for, rise the claim that in order to have full autonomous cars officially on roads, more tests and researches have to be taken, as the principle issue is safety. Driving safety experts predict that once driverless technology fully developed the traffic collisions, caused by human error, should be substantially reduced. The core of this thesis is design, program, and compile a self-driving car robot in a simulation environment.

Simulation is the imitation of the interface and is implied by the notion of artificiality. It can be used to better understand the original entity because simulation can help predict behaviour by making explicit knowledge that is indeed derivable but only with great effort [2].

The chosen research methodology was the DSR, which defined as follows:

Design science research is a research paradigm in which a designer answers questions relevant to human problems via the creation of innovative artefacts, thereby contributing new knowledge to the body of scientific evidence. The designed artefacts are both useful and fundamental in understanding that problem [2].

The main research problem is to analyse how ROS and gazebo fit together in development of autonomous car robot and to identify potential benefits and challenges. To solve the research problem the following questions proposed.

Research question 1: How to build a mobile robot model in a simulated environment?

Research question 2: How can a mobile robot move unsupervised through a simulated environment to fulfil its tasks?

The first question was answered by describing the phases needed to create the model system by ROS and gazebo simulation environment. By answering the first question and analysing the system requirements, the benefits and challenges of using the simulated environment in autonomous driving development were identified, thus answering the second research question. The autonomous navigation can be more reliable by having the

robot explicitly with a perception system act upon providing a reliable map in order to navigate autonomously.

With the research questions answered, the problem addressed. In the conducted DSR, the literature reviews were used to view previous solution for the problems in various design phases. This set of information was used a base for the DSR that followed the analysis. Building on these set of information additional requirement were compiled through deep investigation to the project phases. Once the set of requirements was in place, the design part of the DSR began.

The fundamental principle of design research is that knowledge and understanding of a design problem and its solution are acquired in the building and application of an artefact [2]. Due to the practical nature of the present research, a new system is constructed and evaluated; the DSR methodology was a workable research method. DSR is a research methodology depends on understanding and solving the design problem by build and applies artefacts.

The artefacts are used to describe something that is artificial, or constructed by humans. IT artefacts are broadly represented being as Constructs, Models, Methods, Instantiations, and better design theories. This aids to demonstrate the information system-related problems [2].

The artefact is developed in methods and phases, and then evaluated to produce the required outcomes. In this thesis, an autonomous, ROS and Gazebo simulated self-driving car qualify as the artefacts of this DSR. The DSR-frame work chosen in this thesis shown in the figure (1), adopted in applied manner, because it fits well the purpose and the environment of DSR.

Additionally, figure (1) borrows the IS research framework found in (Hevner et al.2004) and overlays a focus on three inherent research cycles. The Relevance Cycle bridges the contextual environment of the research project with the design science activities. The Rigor Cycle connects the design science activities with the knowledge base of scientific foundations, experience, and expertise that informs the research project. The central Design Cycle iterates between the core activities of building and evaluating the design artefacts of the research [2].

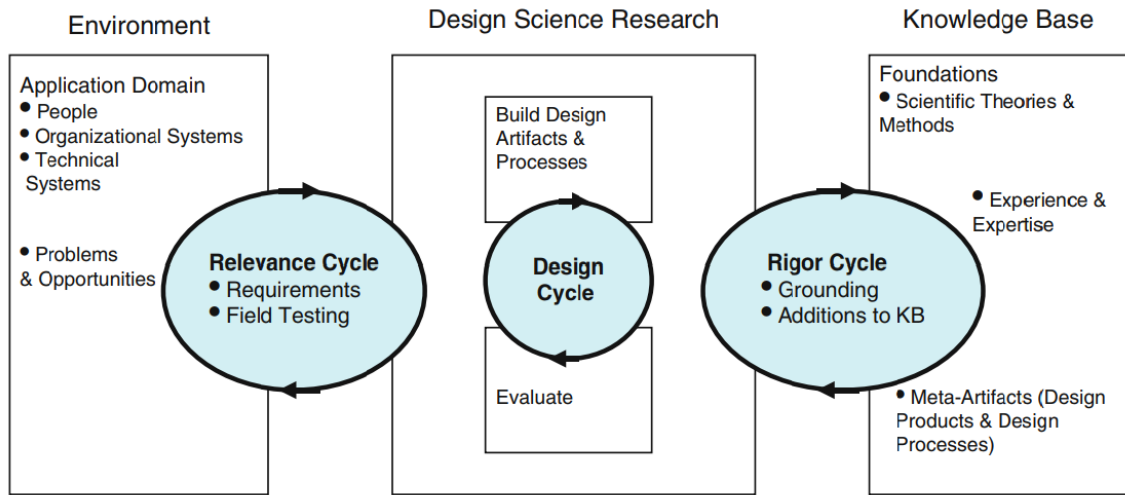


Figure (1) Design Science Research cycles [2]

The DSR combines a focus on the IT artefact with a high priority on relevance in the application domain. The identification and representation of problems in the application environment is the first stage of design science research. The relevance cycle initiates design science research with an application context that provides the requirements for the research and defines acceptance criteria for the ultimate evaluation of the research results.

As importantly, the knowledge base also contains two types of additional knowledge that is the experiences and expertise that define the state of the art in the application domain of the research. Besides the existing artefacts and processes in the application domain.

Design science draws from a vast knowledge base of scientific theories and engineering methods that provides the foundations for rigorous design science research [2]. During the design cycle, artefact is built in an iterative manner.

The environment of the research was the simulation environment, where the created artefact was to operate. The work environment included current practices of simulation environment and simulation model approaches. The information about ROS and its related projects gathered from analyses of the literature and related work. According to the DSR these information's acts to provide the knowledgebase which in turn used in compiling the set of requirements needed for development of the artefact.

The development of the artefacts was done in several activities to achieve the complete solution. Each activity had its objectives to accomplish. The activities were as follows:

Activity1: Design the requirements. The objective here was to address the research question. Along with this, a set of requirements developed. However, the outcome from the literature review formed the list of requirements. Activity completion was evaluated by analysing the validity of the requirements with the practical phases testing.

Activity 2: Program the system components. The objective here was to programme the system's components. The components were specified based on the requirement developed in the previous activity. The activity evaluation was by testing the validity of the system components.

Activity 3: development and test the resulting system. The objective of this cycle was to obtain a complete system to perform the actions according to the programme and make use of programmed system components. The components were programmed in the previous activity. The completion of this activity was evaluated by testing the system and ensures ability of the designed system to fulfil its desired tasks.

The completion of the activities was followed by system evaluation. This involves an analysing to the system performance and capabilities being tested to satisfy the prior set of requirements. The outcome from literature review and the observations upon the DSR gave a solution to the research problem. Consequently, the knowledge collected from this thesis work participated into the knowledge base.

1.1. Specification of the problem

The problem here is to develop an autonomous car robot in a simulation environment. The developed model has the ability to autonomously navigate its path through a pre-described virtual environment world.

The autonomous navigation problem for mobile robot has several individual problems that must be solved in order to reach the full autonomy solution. In this thesis the problem has been treated with parts, each part contributes to the full solution so as to get the optimal system that perform the specified action which is reaching a target location from current specified location. Firstly, the mobile robot localization problem fundamentally referred to the estimation of the robot position inside a 2D map of the environment. Secondly, the locomotion problem consists of determining the optimized route among various routing options depending on path planning algorithms in addition to avoid collisions with obstacles.

However, the development of autonomous car robots considered a hard task; this originates from the unique nature of robotics. Important to realize that the simulation environments enable to simulate self-driving car robot and contribute to solve the autonomous navigation problem. Thus, ROS with gazebo simulation software used to simulate robotics so as to enable safety testing under development phase.

Correspondingly, during the simulation, the principle points to consider were how to build a simulated mobile robot from the ground up with Gazebo and ROS, then how sensors data is being simulated in order to enable the robot to perform actions that complete the required tasks autonomously.

In addition [3] paper states that the autonomous driving is a challenge that involves different risks for roads users. The deployment of technology that enables the automation of driving requires very exhaustive tests in real driving and road conditions, tests that in most countries cannot be carried out either in the quantity or in the actual driving conditions that would be required due to the absence of appropriate legislation. This gap is largely covered by simulation.

In particular, the key challenging to this task was detecting obstacles and mobile robot autonomous locomotion in addition to localization in the virtual world. For instance, consider the case where the mobile robot has to cross obstacles which might be a circular shape of different sizes. Accordingly, the robot must detect the edges of the obstacle using a sensor capable of detecting obstacles, and the model in turn find a path to avoid the obstacles autonomously.

Equally important, the stability issue considered in the solution scenarios contingent upon completion of long-term autonomous navigation without human interaction. Moreover, a velocity controller must be designed and attached to the proposed mobile robot to ensure the robust and stability of the system along the desired trajectory.

Consequently, the key to successful navigation in the proposed simulation environment was the robots ability to sense the environment and how efficiently produces a map, in addition, to use algorithms in finding the shortest path from its current position to the target position. Important to realize, that the mobile robot localization and locomotion problems were considered as a vital topic in research and development.

Hence, in order to prove the competences of the proposed solutions to these problems, a completed set of experiments must be performed for the proposed simulated model, the proposed perception system and the autonomous navigation algorithms in order to measure their effectiveness.

2. Prior research and background

This chapter introduces the analyses of prior research and related works. These analyses introduce the concepts of autonomous driving, ROS, robotic simulators and the solutions under ROS for the autonomous driving problems. Next, a literature review presented in order to drive the solution requirements which later listed in the requirements driven from the literature section, finally the selection of solution section explain the technical related solution provided under ROS and have been implemented throughout the project simulation work in Gazebo simulator.

2.1. Analysis of the problems

Previous studies have been defined the autonomous systems and according to [4] the legal definition of the autonomous cars considered as; “Autonomous vehicle” means a vehicle capable of navigating district roadways and interpreting traffic-control signal devices without a driver actively operating any of the vehicle's control systems.

As an illustration, the self-driving cars are essentially robot cars that can make decisions about how to get from starting point to a target point. In reality, to transform an ordinary car into an autonomous car requires the collaboration of the perception system, analysing and decision making algorithms [5]. Nevertheless, the important question to answer was how to drive the mobile robot along a path? The requirements for answering this question provide a framework for the project design. Basically, the destination point distance has to be measured and feed into the controller in order to turn it into an action, besides additional information about the robot location in the world have to be considered.

Moreover, the autonomous vehicle classification according to [6] defines different levels ranging from fully manual to fully autonomy was published by SAE international Society of Automotive Engineers, based on amount of driver intervention and attentiveness required, rather than the vehicle capabilities, listed below

- Level 0: Automated system issues warnings and may momentarily intervene but has no sustained vehicle control.

- Level 1 ("hands on"): The driver and the automated system share control of the vehicle. Examples are Adaptive Cruise Control (ACC), Parking Assistance, and Lane Keeping Assistance.
- Level 2 ("hands off"): The automated system takes full control of the vehicle (accelerating, braking, and steering).
- Level 3 ("eyes off"): The driver can safely turn their attention away from the driving tasks, e.g. the driver can text or watch a movie.
- Level 4 ("mind off"): As level 3, but no driver attention is ever required for safety, e.g. the driver may safely go to sleep or leave the driver's seat.
- Level 5 ("steering wheel optional"): No human intervention is required at all. An example would be a robotic taxi.

Robotic software

In the development of autonomous driving tasks, the software is a principal instrument. Therefore, the middleware defined according to [7] as “a class of software technologies designed to help manage the complexity and heterogeneity inherent in distributed systems. It is defined as a layer of software above the operating system but below the application program that provides a common programming abstraction across a distributed system”.

In particular, ROS defined as a meta-operating system for the mobile robot it is an open-source middleware, ROS provides the services of operating systems including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, building, writing and running code across multiple computers [8].

Additionally, ROS relies on the concept of open source development, so the contributions from a productive community act upon creating valuable scientific products. For instance, figure (2) demonstrated the contribution of different countries to create open source products. The figure shows that the numbers of scientific papers grew steadily in addition to a considerable increase in the contributions of the developers' community during the period from 2011 to 2017.

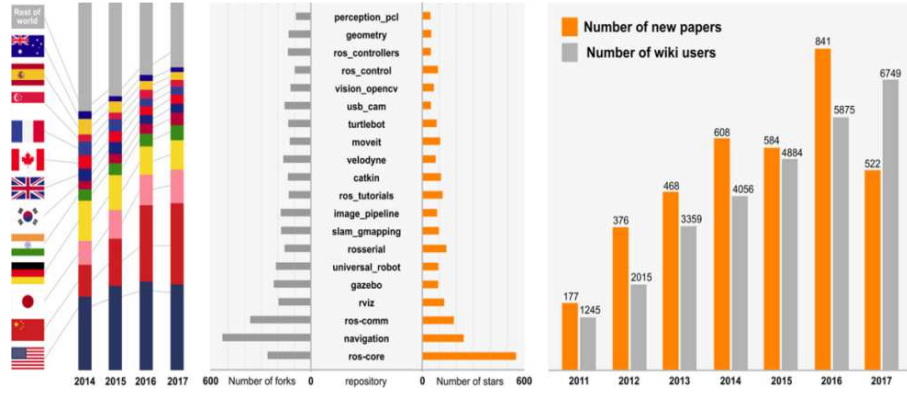


Figure (2) ROS community contributions [9]

Alternatively, a survey was conducted in [10] provide appropriate reference to assist robotic middleware researchers for evaluating strengths and weaknesses for each one of the available robotic middleware. It also defines the middleware appropriateness for the applications. For instance, middleware such as player, orocos, pyro and orca considered as other effective robotics middleware. Nevertheless, ROS have more powerful abilities. However, in table 1 the objectives listed for some of the robotic middleware.

Name	Objectives
Orocos	“Develops a general purpose modular framework for robot and machine control”
Pyro	“Provides a programming environment for easily exploring advanced topics in artificial intelligence and robotics without having to worry about the low level details of the underlying hardware”
Player	Provides a development framework supporting different hardware devices and common services needed by different robotic applications and transfers a controller from simulation to real robots with as little effort as possible
Orca	“Enables software reuse in robotics using component-based development”
ROS	“Provides the operating system’s services such as “hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management”

Table-1 Objectives of several robotics middleware frameworks [10].

Among the most compelling features that the ROS have is that its goal was to design from the file system level to the community, level enables independent decisions about development and implementation, and all can bring together with ROS infrastructure tools. ROS allows a building of reliable robot control and navigation software. Gazebo simulation together with ROS visualizations tools such as RVIS act to create a powerful complete robotics framework, in which results can be directly deployed to the real robot.

Additionally, the robotic simulators are software applications that can visualize robotics model and build a virtual environment, by emulating reality within a certain degree of rigour, it allows inexpensive tests [11]. Even though, they have limitations, such as the simulated models might be inexact due to the lack of information. Nevertheless, experiments bring unexpected parameters into the simulation design considerations so as to provide a more factual simulated model. Furthermore, there exists several commercial open-source simulation software for robotics, for illustration some of these briefly shown below;

- LabVIEW is a complex software system suited for data acquisition, analysis, control, and automation. [12]
- MATLAB with Simulink is a commercially available, multi-domain simulation and modelling design package for dynamic systems. It provides support for ROS through its Robotics System Toolbox [13].
- Webots, since December 2018 it is completely free and open-source. Supports C/C++, Java, Python, URBI and has TCP/IP interface to communicate to other software products visual components [14].
- Virtual Robot Experimentation Platform (V-REP) is a commercially available robot simulator with an integrated development environment. V-REP lends itself to many robotic applications [15].

Despite the existence of several software platforms for robotics simulation and design, ROS (Robot Operation System) however, allows the building of reliable robot model, with a 3D dynamic simulator such as Gazebo the ability to simulate robotics accurately and efficiently will be increased.

The simulated system

The purpose of this thesis is to simulate the mobile robot and find a solution to the localization and locomotion problems in a clear and unambiguous method. Moreover, the creation of an accurate simulated model is important in developing robotic systems, with attention to the perception system that aid in the process of observing the features of the surrounding world. Throughout this thesis, the differential drive robot used because it is considered one of the most cost-effective methods used for testing solutions to mobile robot autonomous navigation problems.

The designed model consists of two wheels and a caster wheel in addition to a front caster wheel added in order to ensure stability and weight distribution of the robot. Furthermore, in [16] analysis for the kinematics, localization and closed loop motion control of a differential drive mobile robot have been described, with aid of simulation concluded that using kinematics motion models the linear and angular velocities of the robot are transformed to right and left wheel speeds and fed as reference speed to PID speed control.

Thus, under the aim of developing a control system design for obstacle avoidance and tracking control in [17] a demonstration for the mechanical structure of differential drive mobile robot presented, in addition to using PID based speed control, the outcomes of [17] was that; “as the values for wheel speeds change the results for linear speed and angular speed of the robot changes. The position, velocities along x and y-axis and the angular speed of the wheels and turning radius of the robot also change.”

Consequently, the sample researches reported here support the assumption that the differential drive robot considered as an effective simulated model to learn and test various methods upon various aspects of the expected simulated mobile robots design environment.

According to [5] a differential wheeled robot will have two wheels connected on opposite sides of the robot chassis which is supported by one or two caster wheels. The wheels will control the speed of the robot by adjusting individual velocity. If the two motors are running at the same speed it will move forward or backward. If a wheel is running slower than the other, the robot will turn to the side of the lower speed. If we want to turn the

robot to the left side, reduce the velocity of the left wheel compared to the right and vice versa. There are two supporting wheels called caster wheels that will support the robot and freely rotate according to the movement of the main wheels.

However, in the control of mobile robot problem, the PID controller considered. It consists of the additive action of the Proportional, the Integral and the Derivative components. Not all of them have to be present; therefore, often employ P-controllers, PI-controllers or PD-controllers [18]. For the remaining of this text, I'll describe the PID controller because any other version can be derived by eliminating the relevant components. This shows a need to explain PID controller usage in order to automatically control the acceleration speed. In general, its main role is to take measurement from the perception system components and compared to some desired trajectory, feed it into the controller to produce the desired steering angle and acceleration speed. Ordinarily, the PID controller bases its functionality on the computation of the "tracking error" e and its three gains K_p , K_i , K_d . In their combination, they lead to the control action u , as shown in figure (3).

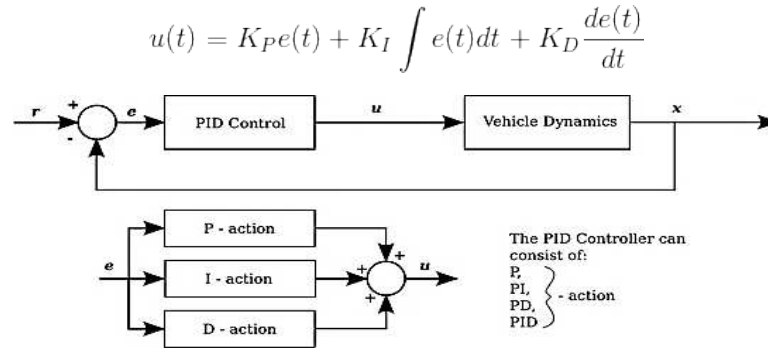


Figure (3) shows PID equation and block diagram, [18]

For instance, in [19] controlling the navigation of the robot using (PID) controller and the values of PID parameters were adjusted in order to get safe and effective navigation of the proposed robot system. The final results show that the proposed control system for the mobile robot was successfully provides a simple and effective method of mobile robot navigation. On the whole, for the controller to be useful there are two facts had to embrace first is the desired path will be in unknown world environment. The second fact is that the world is dynamic.

Furthermore, avoiding an obstacle is another problem that has to be considered in dynamic world when designing a mobile robot controller. For this reason, the perception system must be chosen in order to observe the feature of the space, also, the location of the sensor in the car must be specified so that it will get a wide field to observe and detect the obstacles in the virtual world. In a result, the perception component added to the autonomous system act upon providing the state of the robot in terms of position and orientation as well as aid in obtaining the map for the world. Therefore, active ranging sensors such as 2D laser rangefinder sensor continue to be the most popular sensors in mobile robotics used for obstacle detection. Because active sensors can manage more controlled interactions with the environment, they often achieve superior performance. Passive sensors such as CCD or CMOS cameras measure ambient environmental energy entering the sensor [20]. Nevertheless, wheel encoders also used for figure out where the robot is and count the distance that the mobile robot movement.

There are several robotics sensors that are supported by official ROS packages and many more supported by ROS community. However, Sensors in ROS are classified into 1D range finder, 2D range finders, 3D Sensors (range finders & RGB-D cameras), Audio / Speech Recognition, Pose Estimation (GPS and IMU), Cameras, Environmental, Force/Torque/Touch Sensors, Motion Capture, Power Supply, RFID, Sensor Interfaces and Speed sensor [21]. In general, the featured sensors are documented and should have stable interfaces [21]

In fact, the sensors are used to find the distance from an obstacle and to get the robot odometry data. For instance, the sensors such as ultrasonic distance sensors, or IR proximity sensors are used to detect the obstacles which aid to avoid collisions. Whereas, the vision sensors are used to acquire 3D data of the environment, for visual odometry; object detection and for collision avoidance, also audio devices used for speech recognition.

According to the researches in self-driving car robot, the most common sensors used were the following:

- Ultrasonic distance sensors; depends on the sound waves reflection and used to measure the distance [22].
- The IR proximity sensor; in this sensor the idea based on the reflected light from a surface when hitting an obstacle [22].
- Inertial Measurement Unit (IMU); is an electronic device that measures velocity, orientation, and gravitational forces using a combination of accelerometers, gyroscopes, and magnetometers [22].
- Global positioning system (GPS); compute present position based on complex analysis of signals received from satellite [22].
- Lidar (light detection and ranging), these devices produce a range estimate based on the time needed for the light to reach the target and return [23].
- RADAR (Radio Detection and Ranging); uses radio waves and can work over a relatively long distance [22].
- Camera sensor, Provides image data to the robot that can be used for object identification, tracking and manipulation tasks [22].
- Wheel odometry encoder; used to estimate the position relative to a starting location [22].

In addition, the path panning algorithm must take place after the detection of the obstacles so that it will provide an optimal path from the current position to the destination with the obstacle avoidance ability. However, in order to perform the locomotion of mobile robot, it's necessary to solve several problems such as the initial localization inside the environment, the odometry, perception and mapping, path planning, and the movement speed control.

Correspondingly, the localization of a mobile robot considered as one of the fundamental competencies required by an autonomous robot, since, the knowledge of the robot's location an essential precursor in making decisions about future actions. In addition, localization contains several steps, begins with the measurements taken from the real world and then post-processes those data in order to eliminate the uncertainties that

appeared in the measured information. Afterwards, the robot can be said to have a reliable estimation of its current position [24]. These problems related significantly to the major part of the autonomous navigation which is the map generation.

A map of the environment defined as a list of objects in the environment and their locations. Maps indexed in one of two ways, known as feature-based and location-based. However, both types of maps have advantages and disadvantages. In brief a description, the location-based maps are volumetric, in that they offer a label for any location in the world. Volumetric maps contain information not only about objects in the environment, but also about the absence of objects (e.g., free-space). This is quite different in feature-based maps that can only specify the shape of the environment at the specific locations, namely the locations of the objects contained in the map. Feature representation makes it easier to adjust the position of an object, e.g., as a result of additional sensing. For this reason, feature-based maps are popular in the robotic mapping field; hence, maps are constructed from sensor data. On the other hand, classical map representation is known as occupancy grid map, they assign to each x-y coordinate a binary occupancy value which specifies whether or not a location is occupied with an object.

On the whole, the resulting robot motion is a function of the robot's sensor readings and its goal position in addition to the relative location to the goal position, performed by the means of speed controlling and motion planning.

2.2. Literature review

2.2.1. Path planning

In the first place, route planning is defined as the process of determining a path based on any number of approaches it includes a grid-based approach and network or graph-based approach, finds the shortest path using a network or directed weighted graph of a world model [25].

In particular [24] shows that, if a mobile robot is required to achieve a mission, it has to get feedback from its environment; the self-mapping of the environment was based on laser measurement sensor with dynamic obstacle avoidance and trajectory planning algorithm.

In the same manner, [26] described the motion estimation and mapping using point cloud from a rotating laser scanner and conclude that it can be difficult because the problem involves recovery of motion and correction of motion distortion in the LiDAR cloud.

Unlike, the [27] aim's was to provide iCab platform with the capabilities to be used as a functional intelligent transformation vehicle. The iCab architecture foster the sensor fusion processes, in which the proposed architecture connect the processes in a way to refine information also an enhancement have been added to the decision making process of autonomous path planning and navigation.

In addition the path planning algorithms were the focus of [28] conducted a research on the existing algorithms for solving path planning problem and perform a comparison between the existed path planning algorithms also the paper suggested a design of new algorithm in which it will contribute into real-world robots problems, used ROS with C++ and a simulation program to demonstrate their effectiveness.

Considering the optimum route research and after study of data-driven routing, [29] results based upon the suggestion of machine learning to leveraging information about past traffic conditions in order to learn good routing configurations for future conditions.

Equally important, [30] indicates that the optimal route has been displayed using experienced based fuzzy score of primary and secondary road attributes.

By the same token, considering that road traffic congestion is a major issue in most megacities, [31] present Route Finder, a system that uses standard communication signals from a mobile phone network to determine the optimum route to reach the destination.

Similarly [1] paper emphasizes on use of the information provided by wirelessly connected vehicles about congestion from any potential route used to determine the route with the minimum travel time from the given origin to destination.

For the shortest path algorithms researches, [32] Document gathers all the solutions for each task for navigation and then tests the Dijkstra method for global planning and the Time Elastic Bands method used for local planning. The conclusion was that Time Elastic Bands is a good choice where this method has been tested in a real vehicle.

On the other hand, it is difficult to trace back the history of the shortest path problem [33] investigated lowering the time to find shortest path conducted from at the beginning of the 1950's in the context of 'alternate routing', that is, finding a second shortest route if the shortest route is blocked.

While, [34] Examined the shortest path algorithms and present invention which gave the advantageous effect, that, it is possible to lower a computation time and memory capacity that are required to find a shortest route based on the improved Dijkstra algorithm.

Equally important, the improved A* for route planning algorithm examined to find an effective path which is essential for guiding unmanned surface vehicles (USVs) between way points or along a trajectory [35].

Among the inventions that relates to route smoothing methods and systems for smoothing position data [36] Invention relates to methods and systems for smoothing position data relating to a route travelled by a user in order to generate a more accurate representation

of the route. Embed the GNSS (Global Navigation Satellite Systems) and sensors into a PND Portable Navigation Devices to generate accurate representation of the route.

In [37] paper a comprehensive study on the state-of-art mobile robot path planning techniques, in particular, focusing on algorithms that optimize the path in the obstacle rich environment, hence, conducted for the purposes of providing a better understanding for the researchers in the path planning techniques.

2.2.2. PID steering control

In an investigation into PID controller, [38] show the acronym PID stands for Proportion-Integral-Differential control. Each of these, the P, the I and the D are terms in a control algorithm, and each has a special purpose. PID controllers can work surprisingly well, especially considering how little information is provided for the design.

Likewise, [39] paper, indicates that the based PID controller parameter applied to automatic steering control. The simulation results showed the success of this controller in path following.

Moreover, [40] design a feedback tracking controller, developed a set of control laws, after considering two families of wheeled mobile robots those that are capable of forward motion only and those that can perform forward and backward motion.

Additionally, [41] consider the problem of trajectory tracking for nonlinear systems, and propose a neural networks PID controller. The proposed controller comprises PID control for heading angle and BP Neural network control for the PID parameters adjustment. Results show it can effectively adapted in controlling of an intelligent vehicle.

In contrast, the [42] online self-tuned PID controller for the control of a simulated car model proposed to improve the robustness of the system for varying inputs or in the presence of noise. The proposed method used a first order low-pass filter to avoid the increasing of the parameters individually in this way the resulting system was more robust than the MIT (Massachusetts Institute of Technology) rule dose.

In comparison and analysis of control results under normal right-steering condition [43] propose a Human Simulated Intelligent Control (HSIC) which is a logic-based intelligent control algorithm. The research results show that the HSIC algorithm can simulate human control behaviour.

In an investigation to the mobile robot motion task in obstacle free indoor environment [44] research paper proposed for the DDMR (Differential Drive Mobile Robot), an estimation of robot geometry and speed of its wheel done with forward kinematics. For controlling the angular velocity a PID controller was chosen. The results show that the proposed control system for the linear and angular speeds enable the robot to reach its desired set-points exactly.

By the same token [45] paper present a Model Predictive Control (MPC) for a differential-drive mobile robot (DDMR) based on the dynamic model. The control strategy started with an input–output linearization technique to DDMR, then, based on the obtained linear model, a predictive control law was developed. The proposed approach produces a well system performance.

2.3. Requirements derived from literature

Table 2 displays the requirements selected from the literature review and the analysis of the problem. These requirements were collected from multiple resources, taking into consideration the artefact for design autonomous car development using ROS and gazebo simulation environment.

Requirement number	Requirement description
1.	Autonomous car robot should have sensors like camera, IMU, LIDAR, wheel encoders [23].
2.	Obstacle detection is mandatory for autonomous driving system [23].
3.	Map creation must done to enable autonomous navigation [23]
4.	Autonomous system has the ability to localize its self in a map of the environment [23].
5.	The designed autonomous system must have the ability to plan its route from its current location to a target location using path planning algorithms [25].
6.	The designed autonomous system must responds to a tele-op user input [5].
7.	The developed autonomous system must use ROS messages, services and topics in communication between ROS nodes [5].

Table (2) requirements derived from literature and problem analyses

These requirements emphasize the problem analyses and provide the steps for the solution of the problem requirements. For example the requirement number 3 involve the use of the perception system along with the tele-op node under ROS this implemented with the aid of *gmapping* package whom will be described in detailed in the next chapters of this thesis.

2.4. Selection of the solution technique, explanations

In this chapter, I present the chosen work environment in addition to its effect on the project such as Ubuntu, ROS middleware and gazebo simulator. Also, I've been added the selected solution techniques used in the simulated work, explained with its specifications. Lastly, I explained the selected technique used to provide solutions to project problems.

In the first place the operating system ubuntu version 18.04 LTS [46] was chosen because of its features that comes from the Ubuntu community, it was built on the idea that software should be available free of charge. According to official Ubuntu user statistics report generated from basic non-identifiable system data provided by users installing Ubuntu 18.04 LTS shows that 66% users were opted Ubuntu [46].

Moreover, its main cons are the absence of ads and logos ads on the desktop, privacy and security issues and introducing proprietary elements. While, it's major pros are simplified installation, regular release cycles, encrypted home directories, community building and concern about usability.

2.4.1. ROS (Robotic Operating System)

ROS built on Ubuntu with computer vision, AI libraries and a wide range of sensors supported. It's the most popular development environment for autonomous machines from drones to self-driving cars [47]. Software in the ROS Ecosystem can be separated into three groups [48] [49]:

- language-and platform-independent tools used for building and distributing ROS-based software;
- ROS client library implementations such as roscpp, rospy, and roslisp;
- Packages containing application-related code.

Both the language-independent tools and the main client libraries such as C++, Python, and Lisp also, it has experimental libraries in Java, released under the terms of the BSD

license, and open source software, free for both commercial and educational use [49] [48].

However, its advantages were mentioned in [8] for instance, it provides a flexible framework for robot software. Also, it gives a collection of tools, libraries, and conventions. Its aim was to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms [49].

Furthermore, ROS packages provide support to creating programs for control of real or simulated robots, in addition ROS was built from the ground up to encourage collaborative robotics software development [50]. Whereas its major disadvantages were approaching maturity and still changing, while, the security and scalability were not first-class concerns also; operating systems other than Ubuntu Linux were not supported, but, lately ROS2 version was announced whose have the features to enable it's compatibility to other operating systems until now it is under development [50].

On the other hand, ROS provides a collection of libraries, drivers, and tools used for effective development and building of robot systems with a Linux-like command tool, inter-process communication system, and numerous application-related packages, when used together with a simulation middleware such as Gazebo results in dramatic decrease of the time required for development a reliable and high performance robotic control software systems [49] [48].

In ROS the process name resolving and execution was scheduled by the master server. Other ROS packages also include many sensor drivers, navigation tools, environment mapping, path planning, inter-process communication visualization tool, as well as a 3D environment visualization tool and many others [23].

The fundamental concepts of ROS implementation are nodes, messages, topics and services; the ROS executable process called Node. Its' inter-process communication has either Publish/Subscribe models or Client/Server models the communication data is called Topic [49] [8]. For instance, the Publisher process publish one or more Topics and related processes subscribe to certain Topic can receive the message content [8] [49].

ROS node

ROS master node initiate with *roscore* considered as the essential part of each ROS-based program. Simply, *roscore* define a set of core ROS nodes that are essential for ROS-based application to be able to run, the roscore master node must running for other ROS nodes to communicate [23] [8].

ROS uses internet protocol (IP)-based communication to transfer data between ROS nodes figure (3) illustrates the communication between nodes [51]. Split the robotic program into nodes that can be executed on one machine or on the distributed computer cluster. ROS nodes uses publish/subscribe channels to exchange information amongst them, but they also provide callable services to other nodes [23] [8].

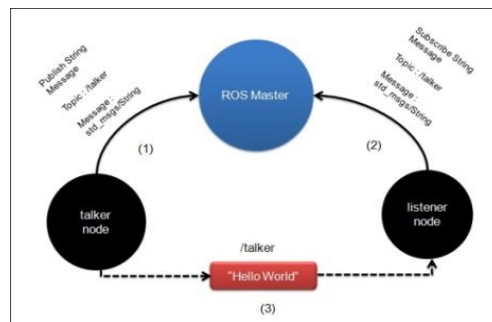


Figure (3) communication between ROS nodes using topics [51]

ROS messages

Nodes communicate with each other by publishing messages to topics. A message is a simple data structure, comprising typed fields. Standard primitive types (integer, floating point, Boolean, etc.) are supported, as arrays of primitive types. Messages can include arbitrarily nested structures and arrays [23] [8].

ROS topics

Topics are named buses over which nodes exchange messages. Topics have anonymous publish/subscribe semantics. In general, nodes that are interested in data subscribe to the relevant topic; nodes that generate data publish to the relevant topic. There can be multiple publishers and subscribers to a topic. Topics are intended for unidirectional, streaming communication. Node needs to perform remote procedure calls, i.e. receive a response to a request, should use services instead [23] [8].

ROS services

Although the topic-based publish-subscribe model is a flexible communications paradigm, its “broadcast” routing scheme is not appropriate for synchronous transactions, which can simplify the design of some nodes. Request / reply is done via a Service, which is defined by a pair of messages: one for the request and one for the reply, this enables higher performance at the cost of less robustness to service provider changes [23] [8].

Rviz

Rviz, abbreviation for ROS visualization, is a powerful 3D visualization tool for ROS. It allows the user to view the simulated robot model, log sensor information from the robots' sensors, and replay the logged sensor information [52]. By visualizing what the robot's view, the user can debug a robot application from sensor inputs to planned (or unplanned) actions.

Rviz also, displays 3D sensor data from stereo cameras, lasers, Kinects, and other 3D devices in the form of point clouds or depth images. 2D sensor data from webcams, RGB cameras, and 2D laser rangefinders can view in rviz as image data. In addition to Rviz the `rqt_graph` tool programmed to become a GUI tool that can visualize the graph of interconnection between ROS nodes and topics [52].

Gazebo

Robot simulators are essential tool in robot projects. A well-designed simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI system using realistic scenarios. It is free offers the ability to accurately and efficiently simulates robots in complex indoor and outdoor environments [53].

Figure (5) illustrates the graphical user interface (GUI) for Gazebo after building the descriptions of the simulated robot. It simulates robots, sensors, and objects in a three-dimensional virtual world, generates both realistic sensor feedback and physically plausible interactions between objects [53].

2.4.2. Usage of ROS for autonomous solutions

In this section the technical strategies that have been investigated, in addition to the requirements derived from the literature were used here to provide the general overview for the solution methods that have been adopted during the design phase of this project.

The technical solution

The project's technical problem has been treated with parts; each part contributes to the full solution. In order to reach the autonomous navigation for the mobile robot several phases have to be undertaken, by dividing the autonomous navigation problem into modules or phases that each one of these modules provide a solution for individual problem. After combine together all the solutions, I got the optimal system that perform the specified action which is reaching a specified target location from current location.

Furthermore, the first part treated the robot localization problem. While, the next treated problem was sense the environment and determining the obstacle locations, after all, the final task gave the solution to determine optimized route among the various routing options depending on path planning algorithms.

Moreover, the aim here is to integrate the ROS navigation stack with the simulated robot model in order to enable autonomous navigation. Yet in order to produce the excepted movement, the ROS navigation stack has a set of configuration files as well as nodes and supported packages that have to install. In order to adopt this configuration the robot must be:

- Well-designed differential drive robot and it could solve the localization. However, the localization considered the basic problem hence any mobile robot system has to answer the default question of "Where am I?" in [16] for a Differential drive robot, the localization was determined using odometric prediction (commonly referred to as dead reckoning) which is accurate enough in the absence of wheel slippage and backlash. In this thesis the dead reckoning was adopted used in the specification of the robot location in the simulation environment.

- The robot has to publish information about the relationships between all joints and sensor positions, and it has to send messages with linear and angular velocities. Many ROS packages require the transform tree of a robot to be published using the tf software library. The "tf" software library is responsible for managing the relationships between coordinate frames relevant to the robot in a transform tree. tf uses a tree structure to guarantee that there is only a single traversal that links any two coordinate frames together, and assumes that all edges in the tree are directed from parent to child nodes. Odometry information published using tf and nav_msgs/Odometry message. The nav_msgs/Odometry message depicts an estimation of the robot position and velocity in free space [49].
- Laser sensor used to create the map. Publishing data correctly from sensors over ROS is important for the Navigation Stack to operate safely. If the Navigation Stack receives no information from a robot's sensors, then it will be driving blind and, most likely, hit things. Many sensors that can be used to provide information to the Navigation Stack: lasers, cameras, sonar, infrared, bump sensors, ...etc. [21]. In [56] utilization of both LiDAR and the RGB camera to detect and localize objects accurately, the results shows that fusion aids to obtain more accurate position and label data for each detection it also create detection beyond the range of LiDAR while within the range of the camera.

In order to react with the environment, sensors and path planning algorithms with movement control operation added to the simulated differential drive robot model. As a result of these additions, the simulated model operates with ROS navigation stack to navigate autonomously. In this project the chosen perception systems were 2D laser range finders and a camera sensor explained in details in perception system section.

Speed control of mobile robot

To date various methods have been developed and introduced to solve the mobility problems. In particular the methods described in [55] illustrated in figure (4) which state the relation between simulation, hardware, controllers and transmissions, it describes the importance of having a simulator and hardware running the same ROS code.

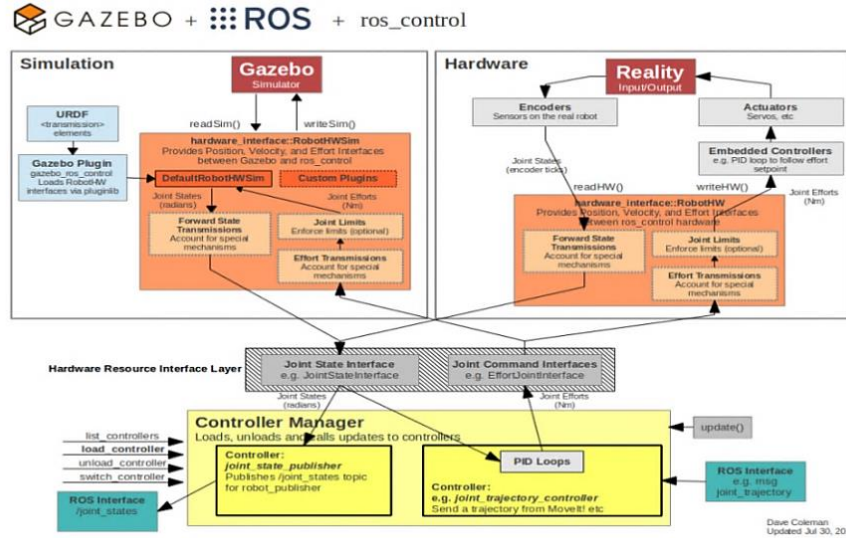


Figure (4) shows data flow of ros_control and Gazebo simulator [55]

In this project the use of gazebo_ros_control package includes transmission tag elements I've been added those tags to the main urdf file of the robot. Those added tags useful to setup simulated controllers to actuate the joints of the robot. While, for this project, this package could be adopted for further development but, after the experiments it was clear that the use of this package could be replaced by differential drive controller model plugin that provides a basic controller for the model in Gazebo simulator.

PID control package

The PID controller package [56] is an implementation of a Proportional-Integral-Derivative controller. It has the features of easy to interface to uses std_msgs/Float64 for set-point, plant state, and control effort, Low-pass filter in the error derivative with a parameterized cut-off frequency provides smoother derivative term [56] . However, the PID controller node used in case it's required to adopt packages such as gazebo_ros_control package which enable the user to modify the PID gains manually.

For the differential drive the pid_velocity implementation through a PID controller using feedback from a rotary encoder to the wheel power to control the velocity of the robot wheel. A typical differential drive setup will have two of these PID velocity controllers, one for each wheel [57] . Additionally, [56] mentioned two ways provided by ROS for

supporting multiple controllers' requirements these are used to connect controller nodes to different topic names, and the PID controller node provides a third:

1. Push each controller node into its own namespace.
2. Remap topic names [56].

The differential drive controller

The control for the differential drive wheel system in the form of velocity command, where it is spilt then sent on the wheels of a differential drive wheel base. The odometry computed from the feedback and publish. The controller works with wheel joints and uses a velocity twist where it gave the x component of the linear velocity and the z component of the angular velocity. It also provides acceleration and jerk limits besides automatic stop after command time-out [58].

Additionally the differential drive controller method provided from [59] provides a basic controller but the constraints were that the differential drive has to be well organized. Below part of the code snippet

```
<gazebo>  
<plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">  
Then set the parameters regarding to the joints of the differential drive robot  
</plugin>  
</gazebo>
```

To summarize there are multiple ways to apply the control mechanism sending command velocity to the base controller in the odometry frame. The programming codes for the chosen methods in this project follow the tutorials that have been given in [60] [59] [61].

The perception system

An approach to obstacle detection for collision-free and efficient humanoid robot navigation based on monocular images and sparse laser range data presented in [62] the results shows that this approach of combine sparse laser data and visual information provide significantly efficient navigation.

In this project a perception system of 2D Laser range finder and a camera sensor simulated as a solution in order to enable the mobile robot perception to the surrounding virtual world of obstacles and enable the robot to create a map for the simulated environment then navigate through that map with aid of ROS navigation stack. The ROS navigation stack uses information from sensors to avoid obstacles, it assumes that these sensors are publishing either *sensor_msgs/LaserScan* or *sensor_msgs/PointCloud* messages over ROS [63].

Moreover, the distance sensor uses laser-based distance sensors such as LDS Laser Distance Sensor, LRF Laser Range Finder, LiDAR Light Detection and Ranging, ultrasonic sensors and infrared distance sensors. In addition, the vision sensors also include stereo cameras, mono-cameras, omnidirectional cameras, and recently, Real Sense, Kinect, Xtion, which are widely to identify obstacles.

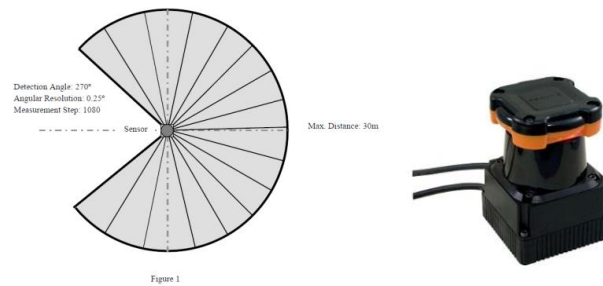


Figure (5) UTM-30LX - Long Distance Laser Range Finder [64]

The chosen simulated sensor was scanning Laser Range Sensor UTM-30LX / use laser source ($\lambda = 870\text{nm}$) to scan 270° semi-circular field, figure (5) shows the UTM-30LX scanning laser range finder. Its advantages are small in size, accurate and high-speed device used for robotic applications. It measures distance to objects in the range and coordinates of those point calculated using the step angle. Sensor's measurement data along with the angle are transmitted via communication channel [64].

After deep investigation to the advantages and disadvantages for this kind of sensors I found that their disadvantages are lack of measurements for transparent objects. The alternatives might be ultrasonic which have the ability to measure transparent objects but its limitations are small measuring range (up to several meters) requires the user to closely control the measurements due to the risk of errors resulting from reflections of

random items also it has low accuracy and low efficiency measurement for objects reflecting the signal at a high angle [65].

Hokuyo scans are taken in a counter-clockwise direction. Angles are measured counter clockwise with (0) pointing directly forward [64]. In order to simulate the sensor a package have been programmed and a node in order to publish scan topic that used by ROS navigation stack.

- `hokuyo_node`

hokuyo_node is a driver for SCIP 2.0 compliant Hokuyo laser range-finders, published Topics: *scan* (*sensor_msgs/LaserScan*), Scan data from the laser [66].

On the other hand, the use of camera sensor is essential to clone the human driver behaviour as claims in the emitted idea of autonomous driving. The camera sensor not only provides the system with 3D measurements of obstacles in the world but also detects the texture and colour of the objects in the surrounding environments.

In this project to have a camera sensor operate and act upon sensing the virtual worlds of environment a camera sensor package were used and programmed with the differential drive robot system and the ROS navigation stack, figure (9) shows the output of the camera sensor after testing.

Under ROS the *image_view* node compiled along with the *image_pipeline* stack designed to process raw camera images into useful inputs to vision algorithms; rectified mono/colour images, stereo disparity images, and stereo point clouds [67].

The camera sensor package:

- Image_view package: ROS image topics Viewer [67].
- *image_view* Node: Subscribed Topics; *image* (*sensor_msgs/Image*) [67].

Simultaneous localization and mapping (SLAM) problem

SLAM (Simultaneous Localization And Mapping) developed to let the robot create a map of unknown environment. This method of learning a grid map for unknown spaces; enable the mobile robot to detect its surroundings and estimates its current location [68].

Map creation

In this project to build the map, the *gmapping* package used, which relay on SLAM to build a map from sensor data published over ROS. Furthermore, for the mobile robot to navigate with the navigation stack it must have a static map additionally, this map enables the robot to localize itself and receive commands to navigation to a specific location within the map [69].

The obstacles defined inside the map during learning phase, while the autonomous navigation with ros navigation stack used that saved map to autonomous navigate.

The *map_server* [70] package within the navigation stack provides *map_server node* used to save the map formats in two main files:

- Image format *.pgm* : describes the occupancy state of each cell of the world in the colour of the corresponding pixel.
- A configuration *.yaml* format: contain parameters such as image, resolution, origin, occupied_thresh, negate and an optional parameter such as mode [70].

Gmapping package

It provides laser-based SLAM (Simultaneous Localization and Mapping), it uses highly efficient Rao-Blackwellized particle filter to learn grid maps from laser rang data [68]. Rao-blackwellized particle filters were introduced as an effective means to solve the simultaneous localization and mapping (SLAM) problem.

In order to reduce the number of particles, an adaptive technique used to reduce the number of particles in a Rao- Blackwellized particle filter for learning grid maps [71]. The proposed approach takes into consideration the movement of the robot and also the most recent observation in order to compute an accurate proposal distribution [71].

Under ROS “the gmapping package contains a ROS wrapper for OpenSlam's Gmapping. The gmapping package provides laser-based SLAM (Simultaneous Localization and Mapping), the node called *slam_gmapping* in gmapping package. Using *slam_gmapping*, a 2D occupancy grid map created from laser and pose data collected by a mobile robot”. [72]

Node `slam_gmapping`

The `slam_gmapping` node takes in `sensor_msgs/LaserScan` messages and builds a map by publishing (`nav_msgs/OccupancyGrid`). Hence, the map can be retrieved via a ROS topic or service [72].

Path planning

The path planning task, involves finding a solution of optimal path with obstacle avoidance in order to have a connection from a start point to the destination point. The solution has been divided into local path planner and global path planner.

In this project under ROS navigation stack the `local_base_Planner` implemented by means of Trajectory Rollout and Dynamic Window approaches explained in [73]. Whereas, the `global_path_planner` implementation using Dijkstra's algorithm, however, in [74] a comparison with A* shows that Dijkstra's algorithm provides best behaviour in creating a path to reach the target location.

Dijkstra's algorithm

This algorithm used to find an optimum path from current position to the target position; it depends on a local optimization at every phase and concludes the final optimization. Its advantages are it is the most common used to find the shortest path between two locations on the map such as the Google Maps or in a graph and the distances between the locations refers to edges. Whereas, its disadvantages are in the robot motion planning problem, it takes time to give out a solution. This fact also proved in [75] in which also defined the Heuristic A* algorithm used for finding path from a start node to a goal node. Even though Heuristic A* algorithm faster than using Dijkstra's, but it produces not necessarily branches for the same paths [75].

Adaptive Monte Carlo Localization

To properly estimate the robot position inside map of the environment, `amcl` (Adaptive Monte Carlo Localization) package used for navigation inside the generated map. This package uses the laser scan and odometry readings data as well a laser-based map. Also a particle filter used to track the pose of a robot inside a known saved map. [76]

ROS Navigation stack

A collection of packages, built with the objective of performing navigation of a mobile robot in an unknown environment. It includes packages that provided solutions depend on algorithms such as A*(star), Dijkstra, in addition to methods such as SLAM and AMCL [63].

The diagram figure (6), illustrates the components that are required for autonomous navigation under ROS navigation stack, it divided into input and output configurations. For instance, the components such as amcl and map_server are essential components for the autonomous navigation [63].

"It assumes that the robot must publishing coordinate frame information using *tf*, receiving *sensor_msgs/LaserScan* or *sensor_msgs/PointCloud* messages from all sensors that are required to use with the navigation and publishing *odometry* information using both *tf* and the *nav_msgs/Odometry* message ". [63]

The navigation stack uses two cost maps to store information about obstacles in the world. One cost map is used for global planning, meaning creating long-term plans over the entire environment, and the other is used for local planning and obstacle avoidance.

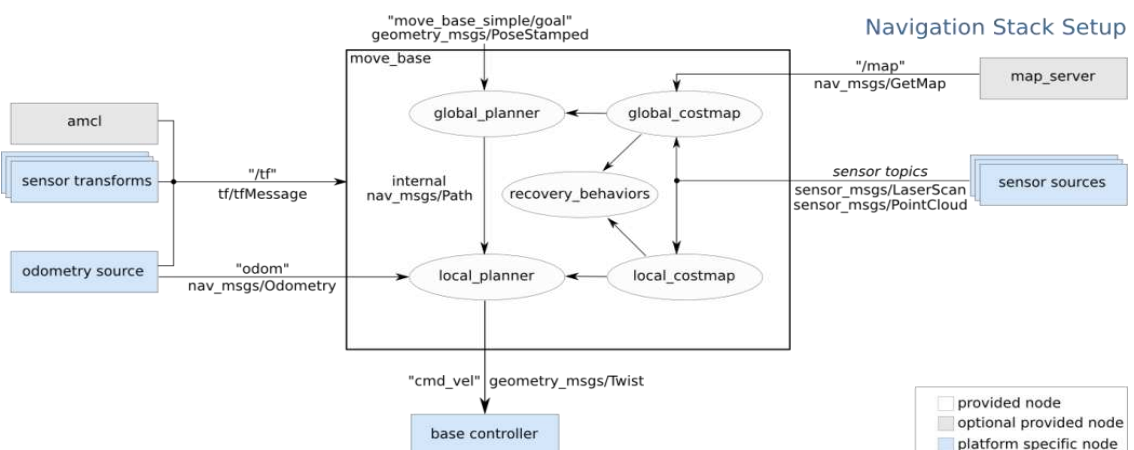


Figure (6) shows ROS navigation stack setup [63]

Odometry and transform tree Information

One of the navigation stack constraints is that the robot must publish information about the relationships between its coordinate frames using */tf*. Also it requires that odometry information being published using */tf* transform tree and the *nav_msgs/Odometry* message [63] [77].

Base controller

The navigation stack assumes that velocity messages sent using *geometry_msgs/Twist* messages, also it assumed to be in the base coordinate frame of the robot on the *cmd_vel* topic [63].

Nav_core package

The `nav_core` package illustrated in figure (7) contains key interfaces for the navigation stack. This package combine the `base_global_planner` which include (`global_planner`, `navfn` and `carrot_planner`) with `base_local_planner` which include (`base_local_planner`, `dwa_local_planner` , `eband_local_planner` and `teb_local_planner`) to produce a recovery behaviour acts upon autonomously navigate and avoid obstacles [78].

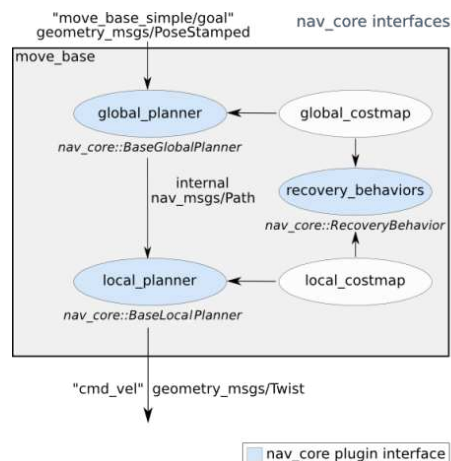


Figure (7) shows the nav_core package interfaces [78]

Move_base package

After the completion of the */odom* and the */tf* topics of the mobile robot, this package considered along with the sensors and */map* topics as input to the *move_base* node which publish a stream of velocity topic *cmd_vel(geometry_msgs/Twist)* to the mobile base robot , basically, linear and angular velocities. Its subscribed topic is *move_base_simple/goal* which provides a non-action interface to *move_base* [79].

Additionally, the *move_base* node takes in *geometry_msgs/PosedStamped message* to the implemented *SimpleActionServer* ; track the status by *SimpleActionClient* [79].

- Action Subscribed topics (*move_base/goal*, *move_base/cancel*)
- Action Published topics (*move_base/feedback*, *move_base/status*, *move_base/result*)

Recovery behaviour

As an explanation of the recovery behaviour according to [79] firstly the obstacles outside the map region are cleared from the robot's map. Secondly, the robot rotates in place to clear out space figure (8). “If this too fails, the robot will more aggressively clear its map, removing all obstacles outside of the rectangular region in which it can rotate in place [79]”. Otherwise, the robot will aborted the process and consider its goal is infeasible. This recovery behaviour configured using parameters such as (*costmap-2d*, *nav_core*, *base_local_planner*, *globalplanner*, *clear_costmap_recovery*, *rotate_recovery*). [79]

move_base Default Recovery Behaviors

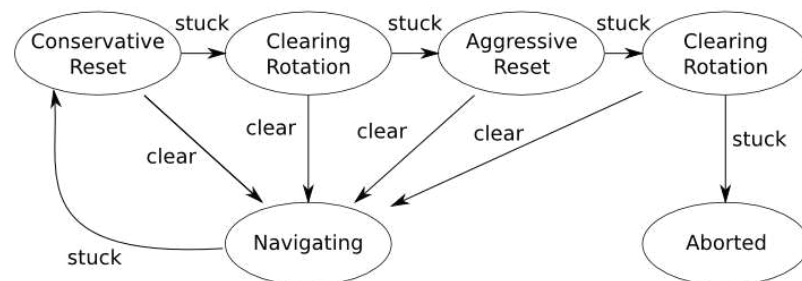


Figure (8) illustrates the *move_base* recovery behaviour [79]

3. The design implementation

In activity 1 of the DSR and after compiling the requirements from the literature and the analysis of the problem along with deep investigation to the ROS and its features mentioned in the previous chapter, the system design begins. The system was developed in phases.

3.1. Detailed specification of the solution

The design phases start with configuration of differential drive model that is to be simulated in gazebo and controlled by ROS, I use the differential drive robot, that is adopted in multiple robot projects for its flexible control mechanism and its compatibility with the adopted solution for autonomous navigation such as ROS navigation stack. Accordingly, the project's practical work was divided into three phases.

Phase 1: Create a simulated model in Gazebo that is controlled under ROS

Phase 2: Add sensors to enable perception

Phase 3: Enable autonomous navigation

3.1.1. Model configuration

The URDF Universal Robotic Description Format associated with an XML macro language *.xacro* aids to create shorter and readable XML files; therefore, it was used in the configuration of differential drive robot under ROS. But, it has a number of lacking features such as it can only specify the kinematic and dynamic parameters of a single robot in isolation, whereas it is unable to determine the pose of the robot itself within the virtual world. Therefore, a format created in Gazebo simulator in order to solve the shortcomings of URDF called the Simulation Description Format (SDF) more scalable files, used to describe objects and environment elements for robot simulators using XML.

According to [80] several steps required to get URDF files properly working in Gazebo the steps that act actively in this project were:

- An <inertia> element within each <link> element must be configured.
- Add a <gazebo> element for every <link>
- Add a <gazebo> element for every <joint>

From the software developer prospective, one of the most useful tools during the simulation were Plugins which are extensions in Gazebo simulation [80], used to add additional functionality to *urdf* under ROS. Therefore, several plugin types supported by Gazebo simulator [80] used under ROS each with its specific functionality.

Consequently, in phase 1 the objective was to create a simulated model in Gazebo that is controlled under ROS. For that, the URDF were used besides, the complete robot description for example the links, joints and transmissions should be in the robot tag.

Also, In order to make use of this URDF file in gazebo simulator additional tags concerning the robot pose, inertial and frictions elements and other parameters were added in the description package main robot configuration *.xacro* file */autodiff.xacro*, with the appropriate plugins in the file *autodiff.gazebo*. Figure (9) gave the project's simulated differential drive robot in the wire-form view in Gazebo simulator.

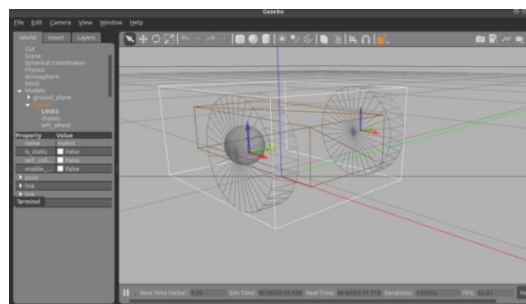


Figure (9) screenshot for the project's simulated differential drive

3.1.2. Perception system

There are several robotics sensors that are supported by official ROS packages and ROS community as well. For this project the Long Distance Laser Range Finder simulated with the aid of node under ROS *hokuyo_node*, used and attached to the simulated differential drive robot main chassis [80]. The simulated Hokuyo sensor added to the main URDF *autodiff.xacro*, defined the link, visual and inertial tags in addition to add to the *autodiff.gazebo* file plugins that define the sensor parameters and the controller, figure (10) illustrates the laser detects' the obstacles.

For instance, part of the code listed here;

```
<plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_laser.so">
<topicName>/autodiff/laser/scan</topicName>
<frameName>hokuyo</frameName>
</plugin>
</sensor>
</gazebo>
```

Then test it, using this command line:

```
$roslaunch autodiff_description autodiff_rviz.launch
```

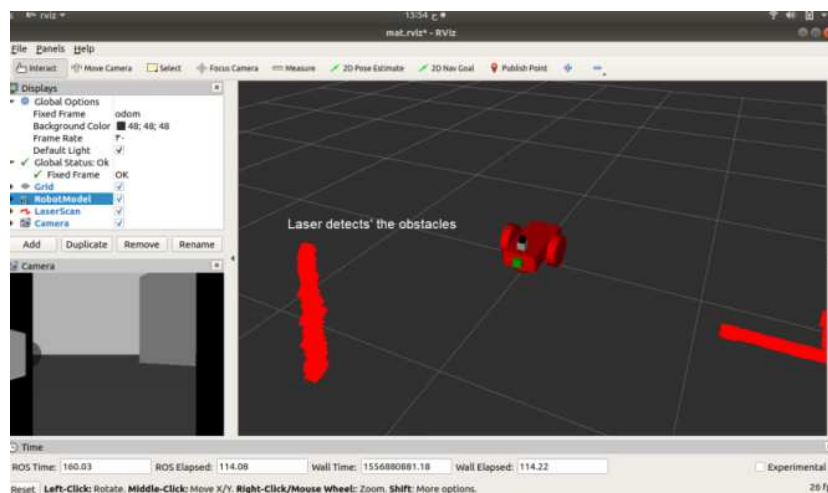


Figure (10) screenshot shows the laser sensor and camera sensors visualized in Rviz

Also, the camera sensor provides image data to the robot that can be used for object identification and tracking. Adding the camera sensor to the urdf main differential drive robot model configuration file, also adding the required camera plugin to the (*autodiff.gazebo*) [80] [67]. Hence, part of the code listed here

```
<plugin name="camera_controller" filename="libgazebo_ros_camera.so">
<cameraName>autodiff/camera1</cameraName>
<imageTopicName>image_raw</imageTopicName>
<cameraInfoTopicName>camera_info</cameraInfoTopicName>
```

To test this node for viewing the camera output Figure (11) illustrates output of using the following command line.

```
$roslaunch image_view image_view image:=/autodiff/camera1/image_raw
```

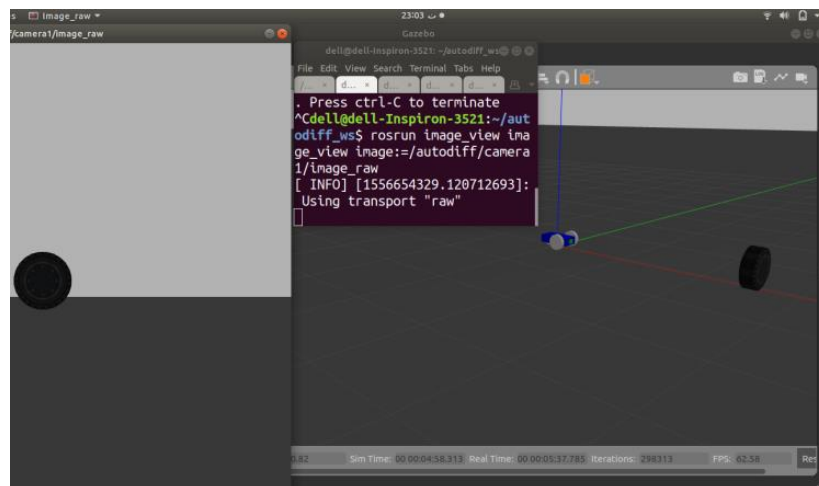


Figure (11) Output image from camera sensor visualized in gazebo

The image_pipeline package stack

Contain *camera_calibration* | *depth_image_proc* | *image_proc* | *image_publisher* | *image_rotate* | *image_view* | *stereo_image_proc* ; its main function is to fill the gap between getting raw images from a camera driver and higher-level vision processing [67].

3.1.3. Navigation of the simulated model

ROS navigation stack

The use of ROS navigation stack is one of the more practical solutions of mobile robot navigation system; it requires necessary pre-requisites such as transform tree, odometry, laser scanner, and base controller. And contain the *move_base* node, which links a global and local planner to accomplish the global navigation task [79].

ROS Navigation Stack inputs:

- Odometry, published to the navigation stack with message of type *nav_msgs/Odometry*, hold the current position and velocity of the robot.
- Sensor messages data either *sensor_msgs/LaserScan* or point-cloud data messages *sensor_msgs/PointCloud*.

The outputs:

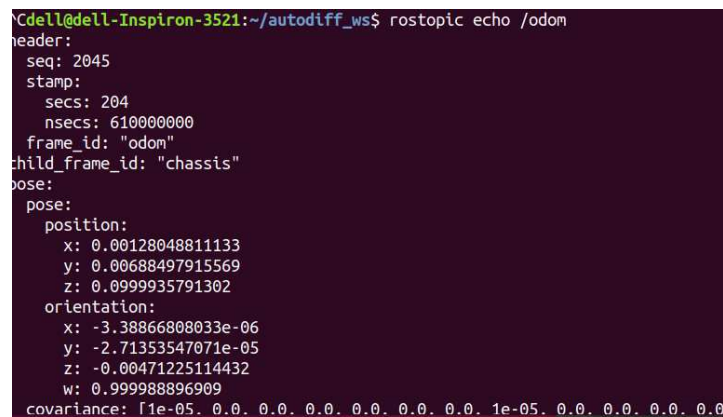
- The *base_controller* took the outputs from the navigation stack using *geometry_msgs/Twist* message and convert it to velocity for the robot.

Other vital packages of the ROS navigation stack are AMCL package which has a main role in the localization of the robot onto the saved map, and the *map_server* package provides the save/load ability to the generated map.

Consequently for this project, in order to make use of ROS navigation stack, and follow the design constraints, I have been created a navigation package *autodiff_naviagtion* in this package I add the solution for creation a map, saving the created map and load that saved map to perform autonomous navigation under ROS navigation stack [63].

Publishing odometry information over ROS

The nav_msgs/Odometry message provides estimation of the position and velocity of a robot in free space. The navigation stack requires that odometry source publish both a transform and a nav_msgs/Odometry message over ROS that contains pose and velocity information. The pose in this message provide estimation of the robot position [77]. In figure (12) illustrates the odometry readings of the simulated robot after executing *\$rostopic echo /odom*



```
Cdell@dell-Inspiron-3521:~/autodiff_ws$ rostopic echo /odom
header:
  seq: 2045
  stamp:
    secs: 204
    nsecs: 610000000
  frame_id: "odom"
child_frame_id: "chassis"
pose:
  pose:
    position:
      x: 0.00128048811133
      y: 0.00688497915569
      z: 0.0999935791302
    orientation:
      x: -3.38866808033e-06
      y: -2.71353547071e-05
      z: -0.00471225114432
      w: 0.999988896909
  covariance: [1e-05, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1e-05, 0.0, 0.0, 0.0, 0.0
```

Figure (12) screenshot illustrates the odometry

The transform tree

As robotic systems get more complicated, being able to focus on precisely the task frame and only the relevant coordinate frames becomes critical. Most robotic systems are fusing data from many different sensors with different coordinate frames, figure (13) illustration of project's transformation tree. The tf library was developed as ROS package to provide this capability; it has two standard modules, a Broadcaster and Listener.

These two modules are designed to integrate with and the ROS ecosystem but are generally useful outside of ROS. The library has been adopted by the greater ROS community as the primary way to keep track of positional information [82].

Using the *view_frames* graphical debugging tool that create a graph of the simultaneous transform tree; its usage comes from run two commands; the first *\$roslaunch tf view_frames* then after run the second command *\$evince frames.pdf*. [82]

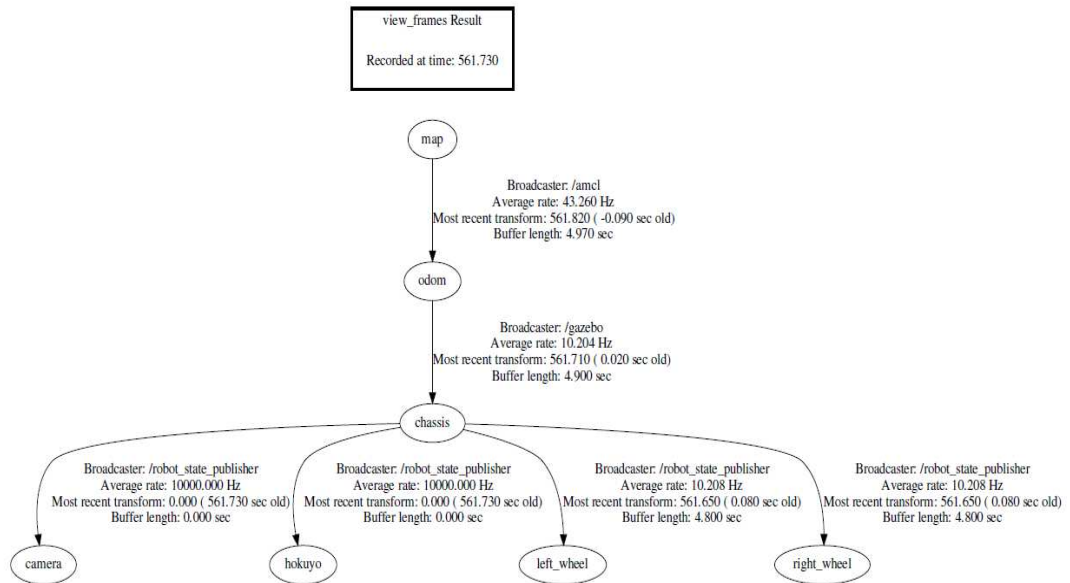


Figure (13) shows the project's transformation tree

Creating a map

The map building requires *gmapping* node. The *slam_gmapping* node subscribes to topic *sensor_msgs/LaserScan* and builds a map (*nav_msgs/OccupancyGrid*) message which represents a 2-D grid map. The generation of the map shown in figure (14), stored in two files one in the *yaml* form whereas the other of a *.pgm* form. The *.yaml* file describes the map meta-data, and names the image file which encodes the occupancy data.

The user-generated static map via the *map_server* package building map; initialize a cost map, *costmap_2d::Costmap2DROS* that match the width, height, and obstacle information provided by the static map. This configuration works with a localization system [63].

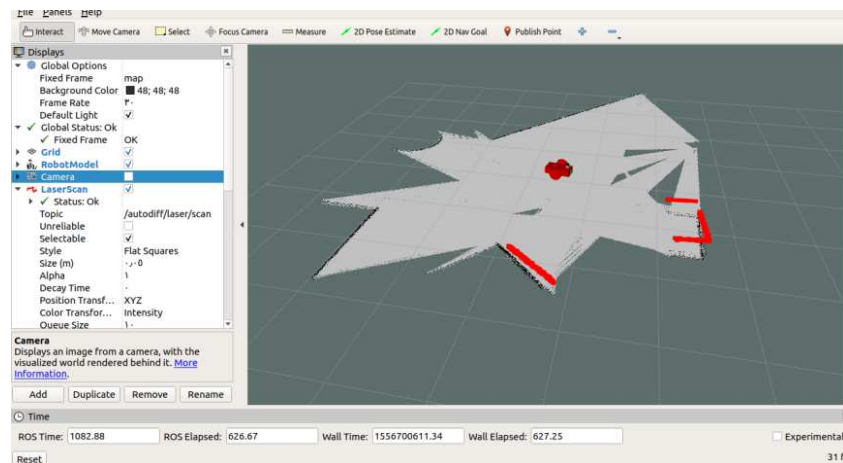


Figure (14) Rviz view during Map generation

Node slam_gmapping

The *slam_gmapping* node takes in *sensor_msgs/LaserScan* messages and builds a map (*nav_msgs/OccupancyGrid*). The map can be used via ROS navigation stack; Figure (15) shows the terminal screenshot of the project node list after launch *slam_gmapping* node. In addition figure (16) shows the node graph for the robot after launch the *slam_gmapping* package [68].

```
dell@dell-Inspiron-3521:~/autodiff_ws$ source devel/setup.bash
dell@dell-Inspiron-3521:~/autodiff_ws$ rosnodetool list
/gazebo
/gazebo_gui
/joint_state_publisher
/robot_state_publisher
/rosout
/rviz
/slam_gmapping
/teleop_twist_keyboard
dell@dell-Inspiron-3521:~/autodiff_ws$
```

Figure (15) terminal screenshot of project slam-gmapping nodes

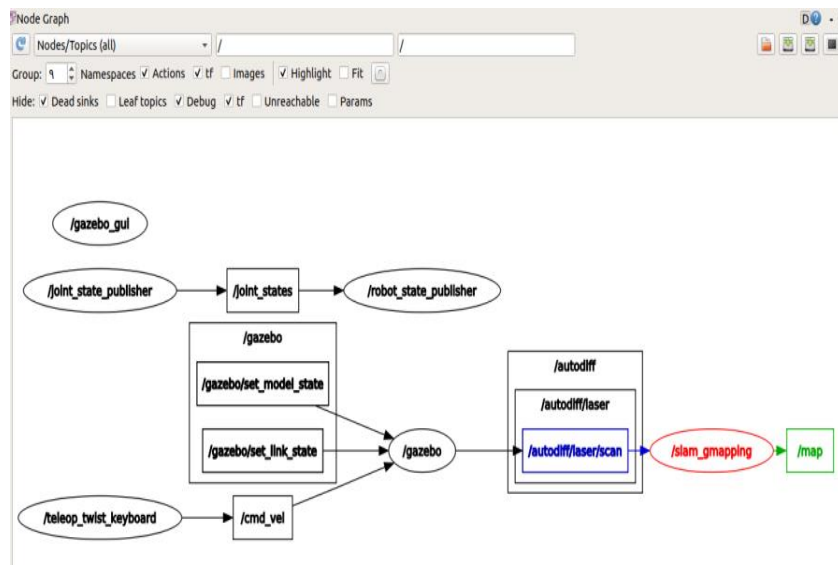


Figure (16) active node/topics graph for *slam-gmapping* node

Tele-operation node

I programmed this node via python, to receive a key-board input sending either linear or angular velocity command to the simulated model. It tested with the following command line and the output illustrated by using *\$rqt_graph* gave the node graph shown in figure (17) in addition to the visualization of the node under gazebo simulator.

\$roslaunch teleop_twist_keyboard teleop_twist_keyboard.py

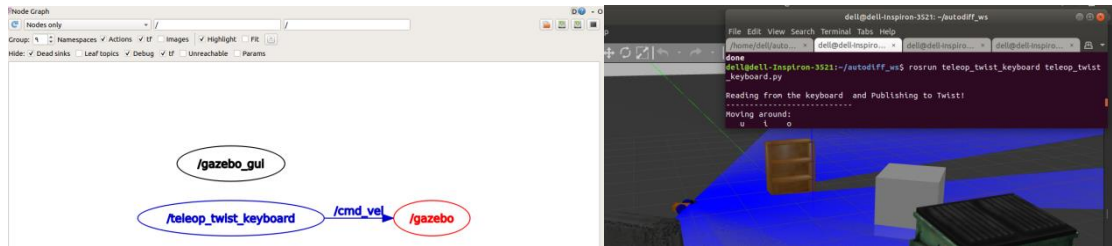


Figure (17) shows the tele-operation node and node graph

Autonomous navigation nodes

The navigation process depends mainly on ROS navigation stack that takes in information from odometry, sensor streams, and a goal pose and outputs safe velocity commands that are sent to a mobile base. The navigation stack uses two cost maps used to store information about the obstacles in world [63]. One cost map used for global planning, it depends on Dijkstra's algorithm, and other for local planning and obstacle avoidance, Dynamic Window Approach (DWA) used for correcting the path in the local coordinates [63]. Figure (18) illustrates the move_base node for this project in active Node/Topics garph;

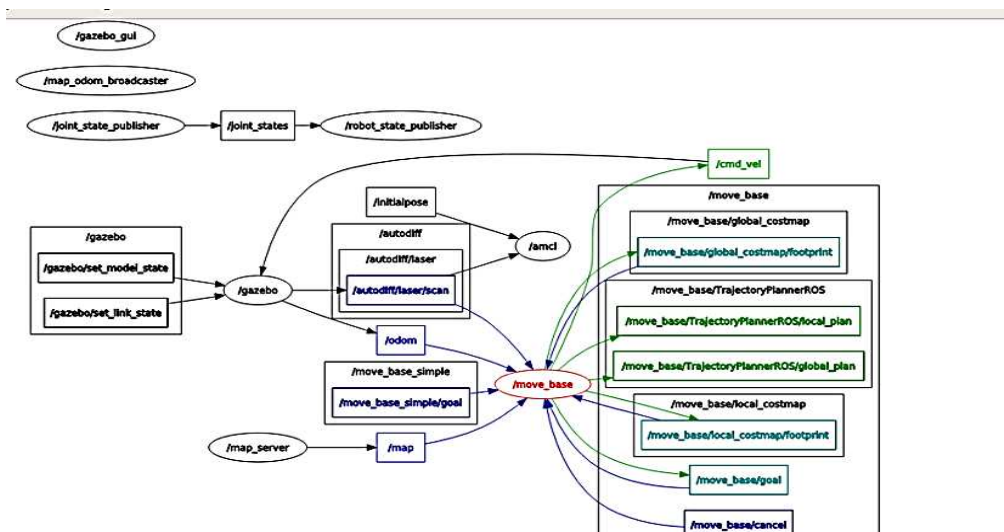


Figure (18) active node/topics of the project for the of move_base node

AMCL Node

AMCL Adaptive Monte Carlo localization approach is a probabilistic localization system for a robot moving in 2D. It implements the adaptive Monte Carlo localization, which uses a particle filter to track the pose of a robot inside a known map.

AMCL takes in a laser-based map; laser scans, and transforms messages and estimated the output position. The services called *static_map* (*nav_msgs/GetMap*); AMCL calls this service to retrieve the map that is used for laser-based localization [76].

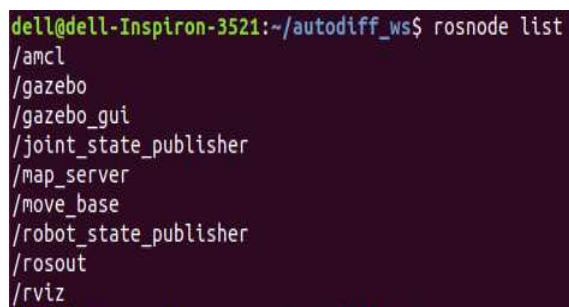
The AMCL launcher firstly launch the saved map configuration file by this tag `<arg name="map_file" default="$(find autodiff_navigation)/maps/try2_map.yaml"/>` where the try2_map.yaml is the configuration file for the generated map in addition to launch the map_server node. Next task of the AMCL is to perform localization of the mobile robot for that it includes the laser sensor and particle filter to estimate the mobile robot position. Part of the node launch code snippet

```
<node pkg="amcl" type="amcl" name="amcl" output="screen">
```

```
  <remap from="scan" to="autodiff/laser/scan"/>
```

```
  <param name="odom_frame_id" value="odom"/>
```

In addition the move_base node launched by the amcl figure (19) shows a screenshot for the terminal of node list after executing `$roscnode list`



```
dell@dell-Inspiron-3521:~/autodiff_ws$ roscnode list
/amcl
/gazebo
/gazebo_gui
/joint_state_publisher
/map_server
/move_base
/robot_state_publisher
/rosout
/rviz
```

Figure (19) screenshot AMCL node list

Speed control of the mobile robot

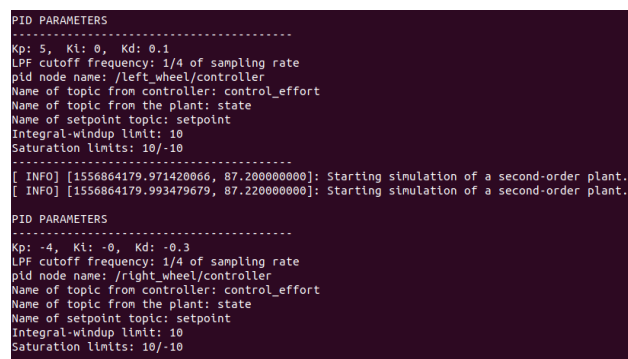
Using the example steps mentioned in [55] in which a working example for rrobot control implemented results in tuning the PID controller with the robot. In this project these mentioned methods adopted and also a control package created with the name *autodiff_control*. According to [55] tutorials, the control package for this project starts by the creation of a configuration file *.yaml* inside the folder *config*, it contains the PID controller gains that loaded in the parameter server via the *roslaunch* command.

Next I create a launch file inside the control package of the project *autodiff_control/launch*, where its duty was to load the controller configuration settings, load the controllers finally start a robot state publisher. In addition, for each and every joint added to the robot there must be a *gazebo_ros_controller* plugin added. For instance, part of the code listed here,

```
<node name="controller_spawner" pkg="controller_manager" type="spawner"
respawn="false" output="screen" ns="/autodiff" args="joint_state_controller
rightWheel_effort_controller
leftWheel_effort_controller"/>
```

PID control node

Implements PID algorithm and applies control effort to try and make plan state equal to set-point. This node subscribe to setpoint, state , PID_enable and publish pid_debug [56]. Figure (20) demonstrate the execution the PID control node.



```
PID PARAMETERS
-----
Kp: 5, Ki: 0, Kd: 0.1
LPF cutoff frequency: 1/4 of sampling rate
pid node name: /left_wheel/controller
Name of topic from controller: control_effort
Name of topic from the plant: state
Name of setpoint topic: setpoint
Integral-windup limit: 10
Saturation limits: 10/-10
-----
[ INFO] [1556864179.971420066, 87.200000000]: Starting simulation of a second-order plant.
[ INFO] [1556864179.993479679, 87.220000000]: Starting simulation of a second-order plant.
-----
PID PARAMETERS
-----
Kp: -4, Ki: -0, Kd: -0.3
LPF cutoff frequency: 1/4 of sampling rate
pid node name: /right_wheel/controller
Name of topic from controller: control_effort
Name of topic from the plant: state
Name of setpoint topic: setpoint
Integral-windup limit: 10
Saturation limits: 10/-10
-----
```

Figure (20) terminal screenshot, PID controller parameters

4. Design work Phases and problems during work

4.1. Simulate the model

In this project part, the steps were:

1- Setup a ROS workspace [83]

```
cd ~ & mkdir autodiff_ws
cd autodiff_ws
catkin init
mkdir src
catkin_make
```

2- Create projects for the simulated robot

```
cd ~/autodiff_ws/src/
catkin_create_pkg autodiff_control
catkin_create_pkg autodiff_description
catkin_create_pkg autodiff_gazebo
catkin_create_pkg autodiff_navigation
```

1. **autodiff_description** package contains the directory and *urdf* files of the robot model.
2. **autodiff_gazebo** package contains the directory and launch files to visualize the model in gazebo.
3. **autodiff_control** package include the directory used to enable control over joints of the model. In case a separated controller used such as *gazebo_ros_control* package, however during this project work *autodiff_control* package contains the config and launch folders for future use.
4. **autodiff_navigation** package include the launch, configurations and maps folders used to adopt methods for building the map in-cooperate with *gmapping* package/node and navigate through the saved map autonomously using *amcl* navigation package/node.

3- Creating the world in Gazebo

This step is to create world under the gazebo simulator, therefore a new world file created in the gazebo package of the project, basically it contain ground plane and a light source. The methods here follow the pervious ROS projects such as turtlebot founded in [84].

4- Create the Model

The programed model required four files:

- `autodiff.xacro`: primary file that loads the other three files and contains items like joints and links. The complete description of links, joints, transmission and all other joints must be included within the robot tag.
- `autodiff.gazebo`: contains gazebo-specific plugins for each component in the design.
- `materials.xacro`: maps strings to colours “materials.xacro” define as a simple Rviz colours file for storing rgba values.
- `macros.xacro`: macros to use the tag and specifies the parameters.

The problems in this part and the available chosen solutions

- 1- Creating the urdf, and the tutorial references were in [85] [83] [86] [59] [87] [88].
- 2- ROS unable to reach package, it was solved by using these references [89] [90].
- 3- The scripts for the client-server configuration did not give the required output; the solution was to recreate the script by follow an instructions given in [91] .
- 4- Unable to locate the packages references were in [92] [93]. After configure the URDF I have to create a world file for visualization and a launch files that according to the tutorials are xml files defines different processes required in order to visualize the differential drive simulated model structure. The problems were with code of the launch and world files. For that I went through a number of practical ROS simulated software projects examples available with [94] [95]. The solution was to create a launch folder and a world folder in the gazebo package. The launch file "*autodiff_world.launch*" initializes some parameters and include the *autodiff.xacro* file, while the world folder contain the supported *autodiff_world.launch* file, it define a basic simulated world with a light source its code I take it from [95] and converted to my own work directory structure.

Accordingly, the test for this part was done by using *roslaunch* command which will process the launch file in the *autodiff_gazebo* package after launching this node the simulated differential drive visualized.

The test for the transform tree by using this command line *autodiff_ws\$ rostopic echo /tf* gave the transforms shown in the following screenshot figure (21). For this part I follow number of tutorials in [96] [82] [86].

```
dell@dell-Inspiron-3521:~/autodiff_ws$ rostopic echo /tf
transforms:
-
  header:
    seq: 0
    stamp:
      secs: 331
      nsecs: 210000000
    frame_id: "odm"
  child_frame_id: "chassis"
  transform:
    translation:
      x: 0.00204619551578
      y: 0.0112712539041
      z: 0.0999936226559
    rotation:
      x: -3.95158449006e-06
      y: -2.69558311504e-05
      z: -0.00852699013312
      w: 0.999963644188
---
```

Figure (21) illustrate the /tf of the simulated robot

4.2. Simulate the sensors

For the robot to sense its environment it requires perception system that contains sensors act as inputs to the robot model. For this project model the perception system depends on two sensors types, camera sensor and laser sensor, the solution for this problem was by following instructions form [87] [97].

In order to simulate the laser scanning *Laser Range Sensor Hokuyo_node* which publish *Laser Scan Topic (sensor_msgs/LaserScan)*. The references were [98] [96] [99] [100] [101] [102] [103] [104] for this part to enable the Laser sensor from detecting object in the virtual world of obstacles.

I have been added a laser sensor plugin to the simulated model URDF, as well as added the laser plugin to the *autodiff.gazebo xacro* file configuration. The references were [87] [99] [59] [66] [101] [107] [104] [106] [103] [107].

The Camera sensor plugin provides ROS interface for simulating cameras by publishing the *Camera_Info* topics and Image ROS messages described in sensor_msgs [87] [102]. This plugin gave the camera view of the robot model. In order to visualize the robots camera sensor, using the following command to test; the output shown in figure (22).

```
$roslaunch image_view image_view image:=/autodiff/camera1/image_raw .
```



Figure (22) Output of the camera sensor

4.3. Navigation of the simulated robot

To move the robot around, I programed a tele-operation package and programmed launch file to launch the tele-op node which allow the mobile robot to move according to specific keyboard input.

During the programming, I make use of both the *roslib* and *rospy* packages that support Python development in ROS. Reference was from [94] [108] [86]. The *geometry_msgs* provides messages for common geometric primitives such as points, vectors, and poses. These primitives are designed to provide a common data type and facilitate interoperability throughout the system [94]. I programed a node to receive a key-board input making use of *geometry_msgs/Twist* to send both linear and angular velocity to the robot model. Results in movement of the model in gazebo simulator, for this to be done I went through a number of tutorials, [109] [110] [111] [91] [112] [113] [114] [115] [116]. Using *roslib* where *roslib* is the base dependency of all ROS client libraries and tools. It contains common tools like the generators for messages and services as well as common message definitions like header and log. It also contains the common path-bootstrapping code for ROS python nodes and tools [108].

By using python code lines to the tele-operation coding figure (23) illustrates a screenshot for the tele-op programed code and the following code line snippet from the programmed code:

```
import roslib; roslib.load_manifest('teleop_twist_keyboard')
.....Other code lines.....
finally:
twist = Twist()
twist.linear.x = 0; twist.linear.y = 0; twist.linear.z = 0
twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = 0
pub.publish(twist)
```

```
#!/usr/bin/env python

from __future__ import print_function

import roslib; roslib.load_manifest('teleop_twist_keyboard')
import rospy

from geometry_msgs.msg import Twist

import sys, select, termios, tty

msg = """
Reading from the keyboard and Publishing to Twist!
-----
Moving around:
   u    i    o
   j    k    l
   m    ,    .

"""
```

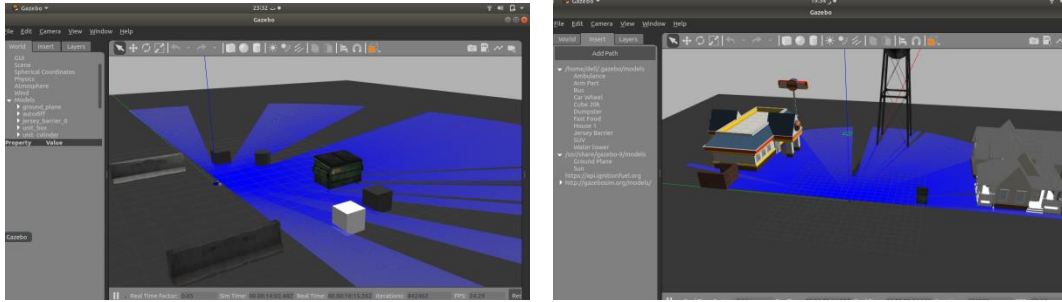
Figure (23) a screenshot for the tele-op programed code

Test this node by `$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py`

The problems in this part of the simulation work

The first problem is that the camera sensor does not capture any images the solution was to add *image_pipeline* package from ros.org official website to the workspace and the references for these were [117] [106] [118] [67]. While the 2nd problem was that the laser does not hit any object, the solution founded in [102] [119] [120] [121] [100].

Consequently, the 3rd problem was creating world of obstacles, there are two solutions the 1st was to use the turtle-bot world of obstacles, unlike the 2nd was to create a simulated world with gazebo. Below figure (24) a screenshot of the created world under Gazebo simulator



1st Simulated world of obstacles

2nd simulated world of obstacles

Figure (24) the simulated world of obstacles

The solution for the 3rd problem, world of obstacles insertion, was done when I used the turtle-bot world of obstacles found in these references [84] [122] [119] [123]. It was simulated and tested with other project packages such as *gmapping* and *amcl* and found that the use of the chosen solution is more stable and provide more connectivity to other project packages under ROS.

Driving around with the created model and with the programmed keyboard node in addition to the simulated obstacles world, the problem of stability occurred figure (25) below illustrated the problem in gazebo

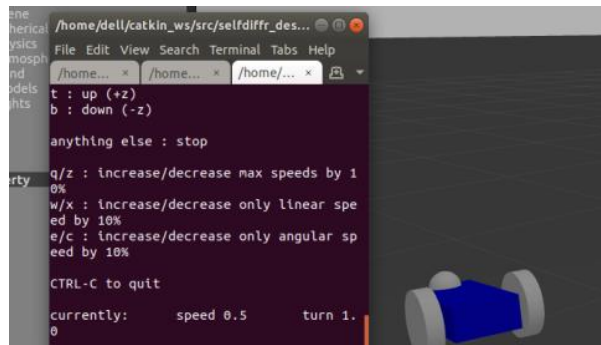


Figure (25) shows screenshot for the problem of the model

The solution was by using a number of resources to modify the urdf, in order to gain stability of the model, the website resources are [98] [124] [125] [57] [101] [126] [127] [128] [87] Fixing the model by adding additional front wheel coaster in the main *urdf* file of the robot, the used code was

```
<collision name='caster_front_collision'>
  <origin xyz="0.15 0 -0.05" rpy="0 0 0"/>
  <geometry>
```

```

    <sphere radius="0.05"/>
  </geometry>
.....set some parameters.....
</collision>
<visual name='caster_front_visual'>
  <origin xyz="0.15 0 -0.05" rpy=" 0 0 0"/>
  <geometry>
    <sphere radius="0.05"/>
  </geometry>
</visual>

```

Using the ROS navigation stack

The ROS navigation stack used to enable the autonomous navigation. Gave the solution to autonomous navigation assumes a virtual world environment of obstacles and a special constraints for the robot model in order to apply the solutions from the ROS navigation stack, the references were in [129] [63] [69]

Create the map

In order to create a map I load the gazebo simulator by launch the *autodiff_world.launch*, which launch the robot model [130]. Part of the code snippet here.

```

<param name="robot_description" command="$(find xacro)/xacro.py '$(find
autodiff_description)/urdf/autodiff.xacro'"/>
<node name="autodiff_spawn" pkg="gazebo_ros" type="spawn_model" output="screen"
args="-urdf -param robot_description -model autodiff" />

```

The launch command line *\$roslaunch autodiff_gazebo autodiff_world.launch*, Figure (26) shows the simulated model in gazebo.

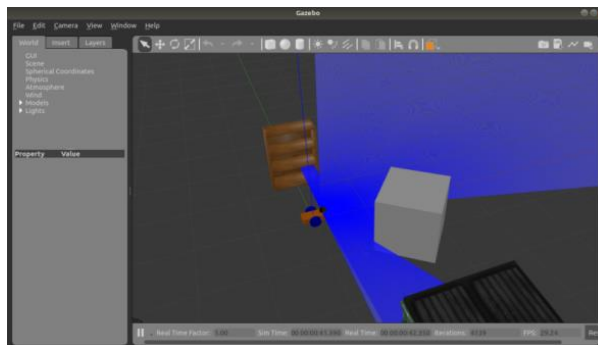
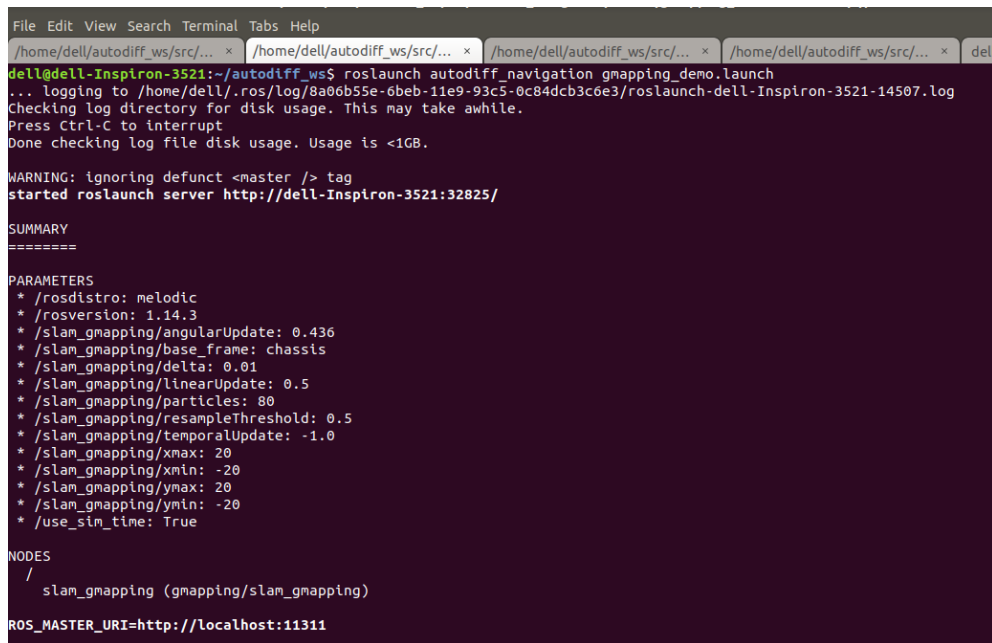


Figure (26) launch the simulated model in gazebo

By using the *gmapping* package provided in references [68] [129] [130] [131] and create the launch file in the workspace/navigation package/ launch folder. This launch file set some parameters and remap from */scan* topic to */autodiff/Laserscan* topic, in order to create a map. Then after, launch the *gmapping* node by *\$roslaunch autodiff_navigation gmapping_demo.launch*, the results of launches this node shown in the terminal screenshot figure (27).



```

File Edit View Search Terminal Tabs Help
/home/dell/autodiff_ws/src/... x /home/dell/autodiff_ws/src/... x /home/dell/autodiff_ws/src/... x /home/dell/autodiff_ws/src/... x dell
dell@dell-Inspiron-3521:~/autodiff_ws$ roslaunch autodiff_navigation gmapping_demo.launch
... logging to /home/dell/.ros/log/8a06b55e-6beb-11e9-93c5-0c84dcb3c6e3/roslaunch-dell-Inspiron-3521-14507.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

WARNING: ignoring defunct <master /> tag
started roslaunch server http://dell-Inspiron-3521:32825/

SUMMARY
=====
PARAMETERS
* /roscdistro: melodic
* /roscversion: 1.14.3
* /slam_gmapping/angularUpdate: 0.436
* /slam_gmapping/base_frame: chassis
* /slam_gmapping/delta: 0.01
* /slam_gmapping/linearUpdate: 0.5
* /slam_gmapping/particles: 80
* /slam_gmapping/resampleThreshold: 0.5
* /slam_gmapping/temporalUpdate: -1.0
* /slam_gmapping/xmax: 20
* /slam_gmapping/xmin: -20
* /slam_gmapping/ymax: 20
* /slam_gmapping/ymin: -20
* /use_sim_time: True

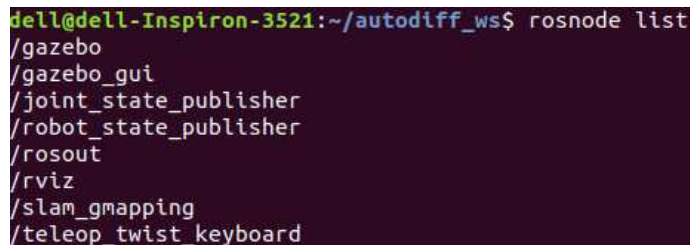
NODES
/
  slam_gmapping (gmapping/slam_gmapping)

ROS_MASTER_URI=http://localhost:11311

```

Figure (27) launch the *gmapping* node

In addition launch the command line *\$roscnode list*, gave a list of the active nodes, figure (28) illustrates the terminal screenshot



```

dell@dell-Inspiron-3521:~/autodiff_ws$ roscnode list
/gazebo
/gazebo_gui
/joint_state_publisher
/robot_state_publisher
/roscout
/rviz
/slam_gmapping
/teleop_twist_keyboard

```

Figure (28) node list after launch the *gmapping* node

In order to start building map of the environment, I have been programmed a customized Rviz file, contains the robot model, the laser subscribe to the topic *autodiff/laser/scan*, the

camera sensor subscribed to (*autodiff/camera1*) topic, and since the *gmapping* node running adding a map subscribed to /map topic. I have saved these configurations in the custom Rviz file *gmap_teleop.rviz* , figure (29) visualized the model in Rviz with the specific configurations under the map frame; the changed code line to launch the specific configuration was

```
<node name="rviz" pkg="rviz" type="rviz" args="-d $(find
autodiff_description)/rviz/gmap_teleop.rviz"/>
```

The launch command line was

```
$roslaunch autodiff_description autodiff_rviz_gmapping.launch
```

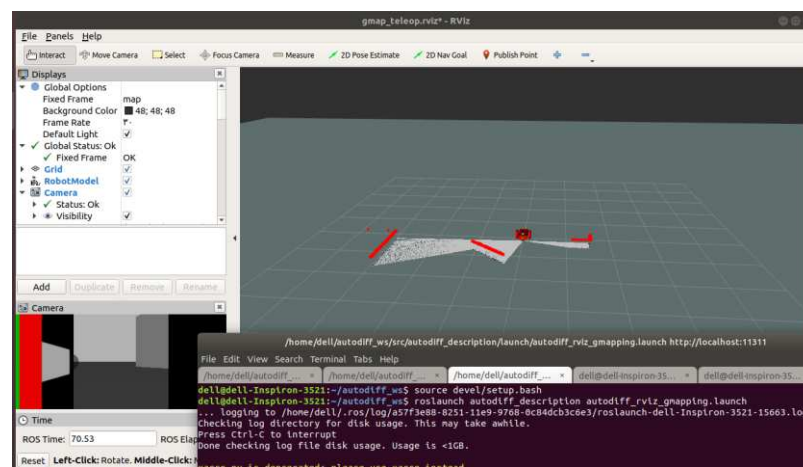


Figure (29) launch Rviz

Launch the tele-operation node *\$roslaunch autodiff_navigation autodiff_teleop.launch* sending the appropriate keyboard inputs, in order to generate the map in which each obstacle circulated with a boarder. Figure (30) illustrate the tele-operation node terminal window and rviz visualizer during the map generation with *slam_gmapping* node.

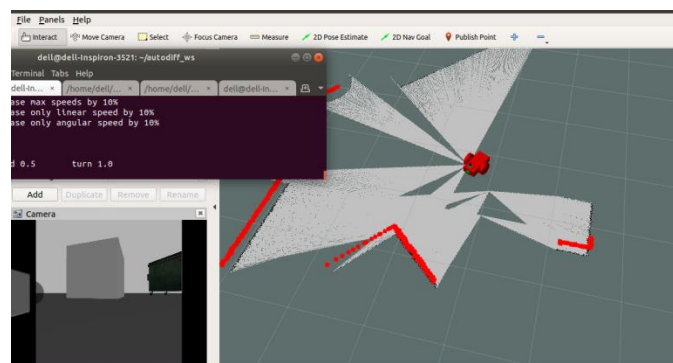


Figure (30) shows map creation

Problems here in this part of the project, were majorly concerned with the *gmapping* package and launching the *slam_gmapping* node. An error occurred with the ROS, that, it cannot locate the node, the solution was by following some of the answers on my question [132] and I've been deduced that this package *gmapping* and its related *slam_gmapping* node has a dependency from another packages which is *open slam gmapping* [72] , however, these dependencies have to be installed in order to the to launch the *slam_gmapping* node and to execute the map building accurately. Consequently, the 2nd problem occurred when implemented *gmapping* that is, shutting down before launching rviz, the solution by using multiple references [133] [72] [134] [135] [136] [137] [129].

Eventually, saving the generated map using this command in a separated directory named maps to be used by the autonomous navigation when launching the amcl node.

```
$roslaunch map_server map_saver -f ~/autodiff_ws/src/autodiff_navigation/maps/try2_map
```

The resulted map saved in try2_map.pgm figure (31) illustrates the generated map .pgm while the other required file will be in .yaml form shown below , demonstrates the try2_map.yaml code :

```
image: /home/dell/autodiff_ws/src/autodiff_navigation/maps/try2_map.pgm
resolution: 0.010000
origin: [-20.000000, -20.000000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```



Figure (31) the generated saved map

Autonomous navigation

The next part of the navigation is to use the saved map in addition to the saved custom Rviz configuration parameters files, this involved on using the following commands: First launch the model in gazebo `$roslaunch autodiff_gazebo autodiff_world.launch`. Then launch the AMCL `$roslaunch autodiff_navigation amcl_demo.launch` ; figure (32) and figure (33) illustrates the node graph and the node list respectively after launch the amcl node;

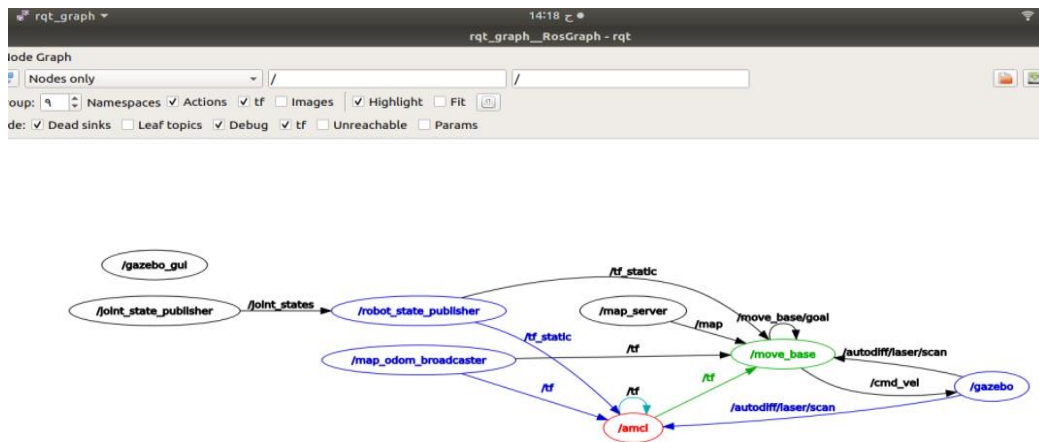


Figure (32) shows node graph for AMCL node

```
dell@dell-Inspiron-3521:~/autodiff_ws$ roslaunch autodiff_navigation amcl_demo.launch
dell@dell-Inspiron-3521:~/autodiff_ws$ rosnode list
/amcl
/gazebo
/gazebo_gui
/joint_state_publisher
/map_server
/move_base
/robot_state_publisher
/rosout
/rviz
```

Figure (33) the ROS node list terminal screenshot

The navigation setup and target references were: [138] [139] [140] [116] [137]. The AMCL node uses a particle filter to estimate the robot position inside the saved map. Hence, figure (34) gave the visualization in Rviz, subscribing to the published topic by amcl node, *particlecloud* (*geometry_msgs/PoseArray*) this gave the set of pose estimates being maintained by the filter, marked in figure (33) with yellow arrows.

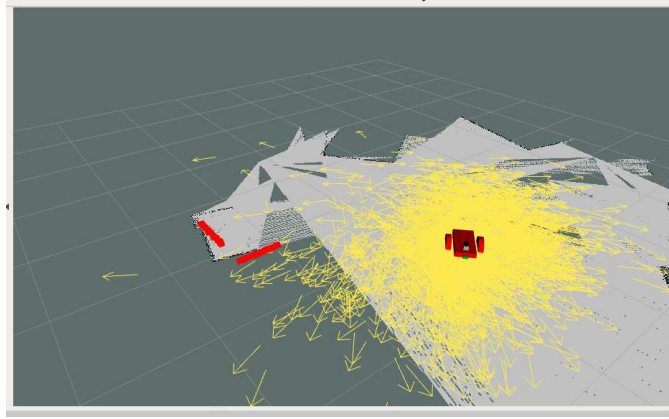


Figure (34) Rviz visualization using the generated saved map

4.4. Summary of the problems during the work and experiences

This work designed to address three main problems; firstly the configuration of the differential drive mobile robot, secondly, the localization of the simulated model itself inside the simulated environment then after the generation of a map that correctly detects the surrounding obstacles finally, the path planning algorithms adopted in order to specify the shortest path which include avoiding collisions during the drive from a start point to its target destination. In addition, this work provide a high level from dealing with ROS packages and enhance the ability to apply new algorithms by programming, compiling and testing for the suggested algorithms and provides a means to enable the decision on what are the preferred methods and what the expected errors that may occur in case it deployed on a hardware.

However, challenges occurred during the developing a reusable robotic software these are coming from the robotic mechanisms, sensor configurations, and hardware control architectures, which also affected by integrating new capabilities, that use architectural design mismatch with the reusable framework. Therefore, in this work, a dynamic process of designing, programming and compiling for the autonomous solution was presented and gave deep investigations to the expected problems and its corresponding suggested solutions.

The detailed investigation to the problem shows that the selection of path planning algorithms is critical issue to ensure that the robot not perform extra turns or visit area already covered. If not, the map generation process may take long time.

Experiences

According to a pre-determined literature review, the use of the designed packages from ROS depends on the previously investigated algorithms which provide the reliability of using this programming technique in the solutions for fully autonomous car system.

The skills required in programming under ROS required a strong C++ or python programming skills because the packages are written in these two languages. Furthermore, the other skills must be in Linux operating system, it was already shown the dependency with ROS in chapter 2.

The experiences learned in this project provide a better understanding of all phases required in autonomous mobile robot development. Significantly, this project contributes in building a simulated self-driving car with the ability to navigate its way on a track; the work area is to develop a reliable environment for simulation and control of mobile robots using the ROS and Gazebo software environments. For this reason, I learned skills to design simulates and program an interactive autonomous mobile robot from scratch with the aid of ROS and Gazebo.

Additionally, I've learned that it is necessary to examine various types of algorithms and compare, analyse and evaluate the performance for each algorithm. In addition to the power of coding that converts the traditional algorithms into a flexible framework so as to be used in further technical development.

5. Evaluation

This chapter does the evaluation of the developed artefact upon the previously specified requirements. This included reporting the fulfilment state of each requirement classified in chapter 2, then an evaluation explanation for each requirement. The requirements list with its evaluation step:

Requirement number 1: Autonomous car robot should have sensors like camera, IMU, LIDAR, wheel encoders.

Status: Achieved.

Explanation: The simulated sensor produced the designed required output.

Requirement number 2: obstacle detection is mandatory for autonomous driving system.

Status: Achieved.

Explanation: The simulated model was able to detect the obstacles using the sensors and avoided collide during the movement with aid of path planning algorithms.

Requirement number 3: Map creation must be done to enable autonomous navigation.

Status: Achieved.

Explanation: following the design constraint the resulted simulated model was compatible with the ROS navigation stack. In essence it has been able to respond to the key-board input, results in the creation for a map of the obstacles in the surrounding virtual world.

Requirement number 4: Autonomous system has the ability to localize itself inside a saved map of the environment.

Status: Achieved.

Explanation: Achieved by implementing a sensor-based localization, with regarding to the tests performed on the odometry of the mobile robot gave the robot pose in terms of the position and orientation.

Requirement number 5: The designed autonomous system must have the ability to plan its route from its current location to a target location using path planning algorithms.

Status: Achieved.

Explanation: Achieved under ROS navigation stack, which include the AMCL package, that act upon the simulated model after programming and compiling an XML launch and YAML configuration files in order to perform a path planning algorithms namely a local path planner implemented with DWA and a global path planner implemented with Dijkstra algorithm, in addition to use of laser scan and odometry readings data as input.

Requirement number 6: The designed autonomous system responds to a tele-op input.

Status: achieved.

Explanation: I've been programed and compiled tele-operation package and a tele-operation node with a launch XML file, which responds to specific programmed key inputs, by utilising from the geometry_msgs/Twist, sending linear and angular speeds to the simulated robot model.

Requirement number 7: The developed autonomous system must use ROS messages, services and topics in communication between ROS nodes.

Status: Achieved.

Explanation: The developed robot model system use ROS topics, messages and services to communicate.

Evaluation of the design phases

Phase 1: Model creation

A differential drive model simulated under ROS and Gazebo simulation environment also visualized in Rviz.

Control the movement

A tele-operation node created and sent keyboard inputs, ensures that the robot moves in both Gazebo and visualized with RVIZ.

Phase 2: Add simulated sensors to the model.

Evaluation: the sensors operated and gave the required output.

Phase3: The use of ROS navigation stack

Evaluation: the simulated differential drive robot has the ability to autonomously navigate, given the target destination node under the ROS navigation stack.

5.1. Explanation of the realization

In this section descriptions of the simulation results using ROS and Gazebo presented. The autonomous car robot equipped with 2D laser range finder (Hokuyo node), camera node simulates a camera sensor used to monitor the environment so as to increase the reliability of the system and a PID control system.

The detailed investigation to the problem shows that the selection of path planning algorithm is critical issue to ensure that the robot not performing extra turns or visit area already covered. If not, the map generation process may take longer time. In addition, the obstacle avoidance algorithms and the map generation by the gmapping package which is laser-based SLAM (Simultaneous Localization and Mapping) and the node is slam_gmapping node used these sensors. The amcl (Adaptive Monte Carlo Localization) package used to navigate inside the generated map. The navigation inside the generated map done by sending goal pose by the user. The simulation result from map generation and robot navigation were presented in the tests chapter.

The system developed under ROS with gazebo simulator in addition to Rviz used to visualize the outcome of the designed model and to test the path planning algorithms, perception system and to perform the autonomous navigation given that the system functionality performance tested and evaluated.

5.2. Analysis of the realization and further development

Greatest success had been achieved up to date in the world of robotics industrial manufacturing. Yet, for all of their successes, these robot systems need additional advanced research to overcome the autonomous mobility problems.

While ROS provide multiple solutions to these assigned problems the need to test and evaluate these solutions still concerned in order to provide more applicable solutions for real robots problems. The benefits of using ROS identified and presented in chapter 2 with its advantages and disadvantages. Furthermore, that ROS developers and also the community have the ability to create and share the ROS packages which make it available to reuse these packages and to enhance the time required to develop new software solutions under ROS so as to create an advanced solutions from the existing one,

instead of using the same steps that was covered earlier from other developers, additionally the visualization tools make ROS development solutions more effective.

In fact, the programmers' developers here have a vital role as they will in-cooperate the path planning algorithms to the manufactured systems. On the other hand, the perception systems act upon the effectiveness of detecting obstacles inside the map as long as the perception systems provide reliable detection; the overall planned autonomous movement would be accurate and reliable. For this reason tests must take place to the completed system and as long as the physical car robots may be expensive to test the simulator provides a complete system to test and can be directly applied to the mobile physical mobile robots. Thus, this project provides a dynamic task of building a simulated system and tests its functionality with the perception systems and path planning algorithms were the results gain benefits for both the scientific researches as well as the industry.

Future development

Testing the simulated model on to a physical robot, may bring additional challenges. However, the simulation tests and results indicate that this method of development can effectively cover most of the design and development problems. Study farther in this project may be the adoption for the voice and face recognition.

The contribution of this thesis work:

- a) Indicates the benefit from using simulation models under ROS and Gazebo.
- b) A research method for creating precise maps by suitable combination of the ROS packages was presented.
- c) The programmed results can be successfully used in the control of physical robots.
- d) Provide a dynamic task for designing, implementing a self-driving car under software simulation development.

This thesis presented my first effort for developing a simulated model and enhances the use of the ROS navigation stack for autonomous navigation. In which a simulated model utilizes its information from the sensors and the path planning to track its trajectory and avoid obstacles in the track robustly and consistently. Benefits of this work procedure involve the knowledge to solve the localization, investigate the perception system and also the locomotion of the designed model autonomously.

The tests begins for this project with firstly launch the model with gazebo simulator `$roslaunch autodiff_gazebo autodiff_world.launch`. Then Launch the model in Rviz and testing the *Robot model* and its `/tf` illustrated in figure (35) shows the model visualized in Rviz in addition to the visualization of the `/tf` inside Rviz, also the figure to the right shows the nodes/topic active graph for the model.



```
dell@dell-Inspiron-3521:~$ rostopic echo /odom  
header:  
seq: 5399  
stamp:  
secs: 540  
nsecs: 10000000  
frame_id: "odom"  
child_frame_id: "chassis"  
pose:  
position:  
x: 0.512291416291  
y: 0.011721920769  
z: 0.099798956861  
orientation:  
x: -6.10088070573e-05  
y: 0.0008408769135  
z: 0.676282122784  
w: 0.736642233114  
covariance: [1e-05, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1e-05, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1000000000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1000000000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]  
twist:  
linear:  
x: 0.016300005715
```

65

Testing ROS topic list for the model shown in figure (37) by using this command line *\$rostopic list*

```
dell@dell-Inspiron-3521:~/autodiff_ws$ rostopic list
/autodiff/camera1/camera_info
/autodiff/camera1/image_raw
/autodiff/camera1/image_raw/compressed
/autodiff/camera1/image_raw/compressed/parameter_descriptions
/autodiff/camera1/image_raw/compressed/parameter_updates
/autodiff/camera1/image_raw/compressedDepth
/autodiff/camera1/image_raw/compressedDepth/parameter_descriptions
/autodiff/camera1/image_raw/compressedDepth/parameter_updates
/autodiff/camera1/image_raw/theora
/autodiff/camera1/image_raw/theora/parameter_descriptions
/autodiff/camera1/image_raw/theora/parameter_updates
/autodiff/camera1/parameter_descriptions
/autodiff/camera1/parameter_updates
/clock
/cmd_vel
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/image_view_1556654328991364443/output
/image_view_1556654328991364443/parameter_descriptions
/image_view_1556654328991364443/parameter_updates
/odom
/rosout
/rosout_agg
```

Figure (37) shows rostopic list

Test building the map

Firstly, the map building under *slam_gmapping* node with this command line *\$roslaunch autodiff_navigation gmapping_demo.launch*, figure (38) shows a screenshot for the node and its parameters.

```
dell@dell-Inspiron-3521:~/autodiff_ws$ roslaunch autodiff_navigation gmapping_demo.launch
... logging to /home/dell/.ros/log/8a0b55e-6beb-11e9-93c5-0c84dc3c0e3/roslaunch-dell-Inspiron-3521-14507.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

WARNING: ignoring defunct <master /> tag
started roslaunch server http://dell-Inspiron-3521:32825/

SUMMARY
=====
PARAMETERS
* /roscpp: melodic
* /rosversion: 1.14.3
* /slam_gmapping/angularUpdate: 0.436
* /slam_gmapping/base_frame: chassis
* /slam_gmapping/delta: 0.01
* /slam_gmapping/linearUpdate: 0.5
* /slam_gmapping/particles: 80
* /slam_gmapping/resampleThreshold: 0.5
* /slam_gmapping/temporalUpdate: -1.0
* /slam_gmapping/xmax: 20
* /slam_gmapping/xmin: -20
* /slam_gmapping/ymax: 20
* /slam_gmapping/ymin: -20
* /use_sim_time: True

NODES
/
  slam_gmapping (gmapping/slam_gmapping)

ROS_MASTER_URI=http://localhost:11311

process[slam_gmapping-1]: started with pid [14522]
[ WARN ] [155669551.245680319, 23.840000000]: MessageFilter [target=odom]: Dropped 100.00% of messages so far. P
message not filtered because topic is not in the filter list. For more information
```

Figure (38) shows the *slam_gmapping* launched

In addition test the *slam_gmapping* node by using *\$roscpp node info* command line, figure (39) a screenshot for the node info;

```
dell@dell-Inspiron-3521:~/autodiff_ws$ rosnodetool info /slam_gmapping

Node [/slam_gmapping]
Publications:
* /map [nav_msgs/OccupancyGrid]
* /map_metadata [nav_msgs/MapMetaData]
* /rosout [rosgraph_msgs/Log]
* /slam_gmapping/entropy [std_msgs/Float64]
* /tf [tf2_msgs/TFMessage]

Subscriptions:
* /autodiff/laser/scan [unknown type]
* /clock [unknown type]
* /tf [tf2_msgs/TFMessage]
* /tf_static [tf2_msgs/TFMessage]
```

Figure (39) shows info about node slam_gmapping

Then, launch Rviz `$roslaunch autodiff_description autodiff_rviz_gmapping.launch` figure (40) a terminal screenshot shows the Rviz terminal. Finally, Launch the tele-op node `$roslaunch autodiff_navigation autodiff_teleop.launch`, shown in figure (41)

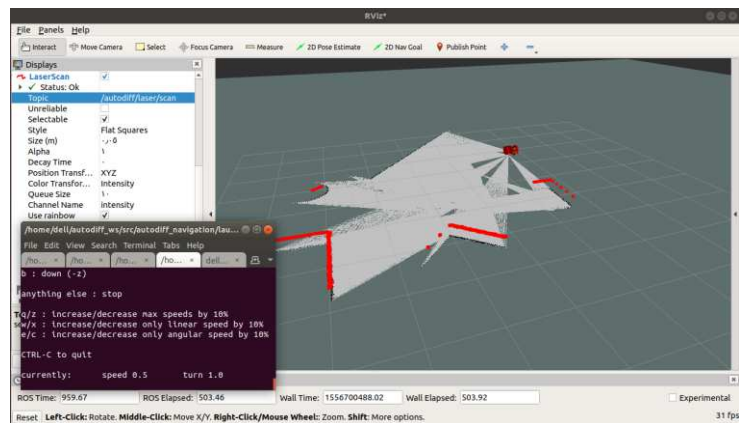


Figure (40) illustrate map generation in Rviz with the use of teleop node

```
File Edit View Search Terminal Tabs Help
/home/dell/autodiff_ws/src/... x /home/dell/autodiff_ws/src/... x /home/dell/autodiff_ws/src/... x /home/dell/autodiff_ws/src/... x dell@
dell@dell-Inspiron-3521:~/autodiff_ws$ source devel/setup.bash
dell@dell-Inspiron-3521:~/autodiff_ws$ roslaunch autodiff_navigation autodiff_teleop.launch
... logging to /home/dell/.ros/log/8a06b55e-6beb-11e9-93c5-0c84dcb3c6e3/roslaunch-dell-Inspiron-3521-15365.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://dell-Inspiron-3521:34301/

SUMMARY
=====
PARAMETERS
* /roscpp: melodic
* /rosversion: 1.14.3
* /teleop_twist_keyboard/scale_angular: 1.5
* /teleop_twist_keyboard/scale_linear: 0.5

NODES
/
teleop_twist_keyboard (teleop_twist_keyboard/teleop_twist_keyboard.py)

ROS_MASTER_URI=http://localhost:11311

process[teleop_twist_keyboard-1]: started with pid [15380]

Reading from the keyboard and Publishing to Twist!
-----
Moving around:
u l o
j k i
m , .

For Holonomic mode (strafing), hold down the shift key:
U I O
J K I
```

Figure (41) terminal screenshot for the teleop node

Building a map to the surrounding environment of obstacles and saving the map to a path file with aid of *map_server* and this command line *\$roslaunch autodiff_navigation map_saver -f ~/autodiff_ws/src/autodiff_navigation/maps/try2_map* , figure (42) shows screenshot for the terminal of map saving command execution, the saved map illustrated in figure (43).

```
dell@dell-Inspiron-3521:~/autodiff_ws$ source devel/setup.bash
dell@dell-Inspiron-3521:~/autodiff_ws$ roslaunch map_server map_saver -f ~/autodiff_ws/src/autodiff_navigation
/maps/try1_map
[ INFO] [1558604023.143310736]: Waiting for the map
[ INFO] [1558604108.765828841, 626.340000000]: Received a 4000 X 4000 map @ 0.010 m/pix
[ INFO] [1558604108.766136388, 626.340000000]: Writing map occupancy data to /home/dell/autodiff_ws/src/aut
odiff_navigation/maps/try1_map.pgm
[ INFO] [1558604109.629888518, 627.190000000]: Writing map occupancy data to /home/dell/autodiff_ws/src/aut
odiff_navigation/maps/try1_map.yaml
[ INFO] [1558604109.630149419, 627.190000000]: Done
dell@dell-Inspiron-3521:~/autodiff_ws$
```

Figure (42) terminal screenshot for map saving command



Figure (43) the *try2_map.pgm* generated saved map

Then, in order to test autonomous navigation, I have to close the previous program except the Gazebo, then after launch the *amcl* node by using this command line

\$roslaunch autodiff_navigation amcl_demo.launch; the figure (44) below shows the node list after launch the *amcl*, while figure (45) shows active node/topics graph after launch the *amcl*.

```
dell@dell-Inspiron-3521:~/autodiff_ws$ roslaunch autodiff_navigation amcl_demo.launch
/amcl
/gazebo
/gazebo_gui
/joint_state_publisher
/map_server
/move_base
/robot_state_publisher
/rosout
/rviz
```

Figure (44) node list

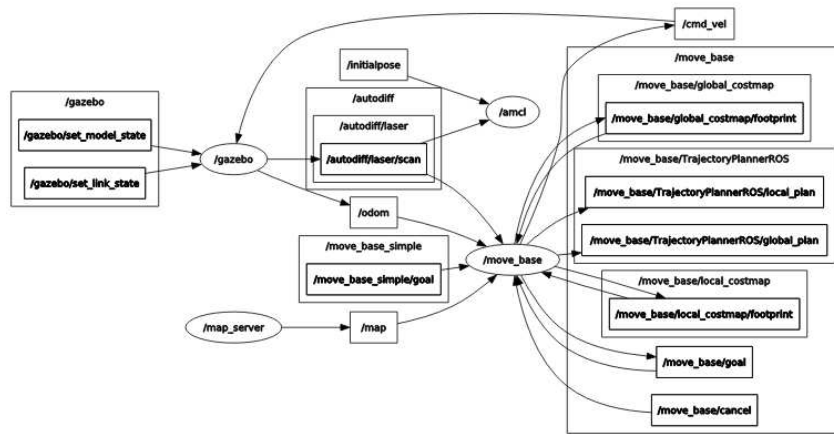


Figure (45) illustrates node graph amcl active node/topics

Then launch Rviz by `$roslaunch autodiff_description autodiff_rviz_amcl.launch` shown in figure (46), then start testing the autonomous navigation using the 2D Nav.goal sent the goal position to the simulated model so that it will autonomously avoids the obstacle in the selected path.

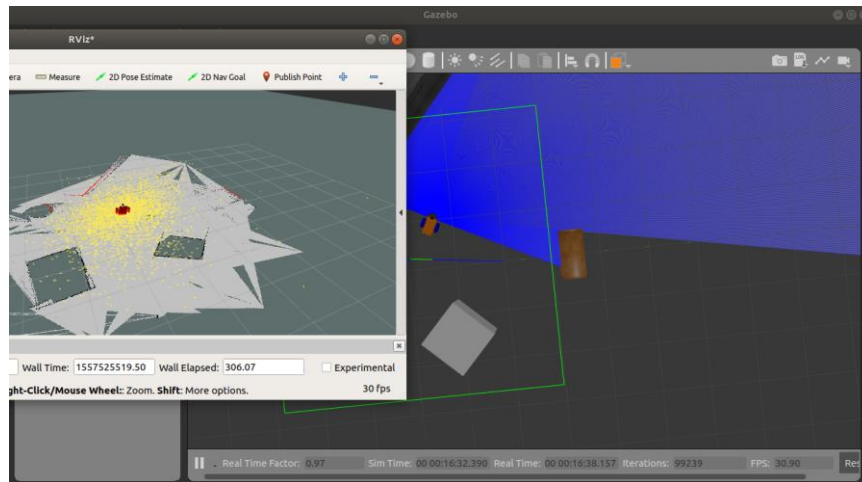


Figure (46) Rviz and Gazebo illustration to the autonomous navigation

Results

This research was conducted with the aim of designing, programming and compiling an autonomous car system under ROS and gazebo simulation environment. The further aim was to clone the human driver behaviour so as to reach for a system that operates inside the city roads with safety. In order to reach the practical solution, in this project, a differential drive mobile robot were designed and tested, and then a perception system consists of laser scan sensor in addition to a camera sensor where chosen and simulated. The complete design and programmed system were connected with ROS navigation stack in order to use the autonomous solution, with the requirements that derived from literature described in chapter (2).

The complete system evaluated and tested in chapters (5) and (6) respectively. Accordingly, the results demonstrates that the designed model autonomously navigates through the virtual world of obstacles using the generated 2D map and follow the shortest path from its current position to the target position; make use of the perception simulated system in addition to the use of path planning algorithms in order to choose the shortest path.

As a result, the autonomous solution in this work follows a practical design phases that is to be able to develop the autonomous navigation solutions. The requirements obtained from the literature review were specified, consequently, after completing the design work an evaluation section conducted to ensure the success for the phases that have been followed the design requirements. However these requirements act upon providing the complete reliable solution for the autonomous system.

7. Discussion

The purpose of this thesis focuses on the use of ROS and Gazebo simulation environment to develop a self-driving car robot and to search into the benefits and challenges of the design problems. The research used the DSR methodology to test the use of ROS in autonomous car development by implementing the system as described in chapter three.

The requirements for the system design were derived from the literature review. In addition, DSR was conducted to answer the research questions. Therefore, the answer to the first research question addressed by simulating the model under ROS and gazebo and was developed per phases. Then, the answer to the second question addressed by the analysing the research problems in addition to the background knowledge gathered from literature review. Moreover, the setup of ROS navigation stack gave assistance in the solution to obstacle avoidance and gave best path planning for locomotion. Ultimately, the system evaluation indicates that the results met the design requirements.

Overall, the structure of the developed work in this thesis offers a full project background to develop and enhance new methods for future research and development in self-driving cars. On the whole, by following the given methods a self-driving car can develop with the open source ROS and powerful Gazebo simulator, so the developer able to test the suggested methods with no risks.

To sum up, the project phases and activities participate in the knowledge-base of the self-driving car. In addition to a deep description of ROS advantages and drawbacks gave another importance to this thesis in the research knowledge base. Eventually, in this thesis, a complete guide to developing a self-driving car in a simulation environment was described, make it possible for programming developers utilise their programming skills in the industry sector with inexpensive means.

8. Conclusion

The objective of this thesis was to investigate the design of an autonomous mobile robot model under ROS and Gazebo environment, in addition to gain accurate illustration for the benefits and challenges of such an approach. The research used the DSR methodology to design and evaluate a robot system under ROS middleware and Gazebo simulator the system developed in phases, that strictly followed the technical solution constrained described in chapter 2 required to enable the integration with the ROS navigation stack.

The simulated system was a differential drive robot as an applicable simulated model. Its implementation drew from the requirements presented in chapter 2, which were developed from the literature and the analysis of the problem. The DSR's design phase included planning the desired system's structure and workflow, acquiring components, simulating the *urdf* of the differential drive mobile robot and developing the design in phases follow the simulation design requirements. The evaluation assessed how well the system met the design requirements.

Results illustrated that the simulated model for autonomous driving provides a testing application that will enhance the ability to adapt and evaluate new algorithms. This thesis shows how ROS and Gazebo can be used to develop autonomous cars by following the steps in this DSR, thereby answering the first research question. The benefits and encountered challenges when using the simulation environment in autonomous driving development during this DSR presented. Accordingly, in this thesis, the benefits and challenges were observed concerning the use of ROS and gazebo in developing a simulated autonomous car. This thesis provides an applicable tool to test and evaluate new or updated solutions in various parts of the design.

Furthermore, the benefits and drawbacks of using ROS for robot programming presented. In addition to the fact that ROS an open-source middleware and provide supporting packages for self-driving cars that can be used in development with simulation software; it supports visualization tools that enable the developer to predict how the systems behave. The programming with ROS required solid C++ or python skills because ROS packages written with these two programming languages.

Eventually, chapter 5 provides an evaluation of the designed system and its applicability to operate in autonomous navigation designed simulated environment followed by a chapter 6 which provide the required tests, thereby answering the second research question.

In conclusion, this research will assist in the complex task of overcoming with the issues of autonomous driving. Though the present DSR-developed artefact was a differential drive robot, the application of path planning and the controlling autonomously were the same as those of a real car. As such, the problems, solutions, benefits and challenges discussed in this DSR are viable for ROS-Gazebo simulated autonomous cars model of any size and environments of any type.

With a comprehensive point of view, during this thesis, more benefits were observed than challenges regarding the use of ROS and gazebo in developing autonomous driving. Although the testing of physical robots brings new challenges that cannot be expected, nevertheless, it could be translated into simulated tests and find an efficient solutions. Research on this topic should continue, as it contributes to the autonomous robotics community for both academic and commercial.

9. Summary

In the final analysis, this thesis objective was to determine how ROS middleware and Gazebo simulator can be used in programming and simulating of autonomous driving and to identify the benefits and challenges of such implementation. The specified research problems were addressed through the DSR methodology and the development of the artefacts was done into activities. Moreover the programming was done through phases. Accordingly, the evaluation results indicated that the requirements met. To wrap up, the simulated differential drive robot model together with the suggested and tested autonomous solutions provides an outcome that contributes to the development of the self-driving cars.

Eventually, the achieved tasks were, developing an autonomous car robot system under the control of ROS and simulated with Gazebo simulator in addition to visualized in Rviz. The results prove that these adopted techniques enable the developers from testing various kinds of algorithms and programming solutions in order to enhance the overall productivity for the autonomous mobile robot system.

Furthermore, the achieved designed perception system enable the mobile robot from detecting obstacles and used as inputs to the robot also used to solve the localization and to navigate its path along the desired trajectory. With attention to the major problem of detecting the shortest path, that was analysed and investigated through the literature in order to decide the best methods of finding the shortest path. This task achieved and analysed in detailed. The resulted system achieved the requirements derived from the literature and the design specific requirements under ROS.

In this thesis, the overall observations can aid significantly to advance researches in the field of autonomous driving abilities additionally. The main achieved aim was that the robot has the ability to perform the required tasks unattended for long periods of time; the thesis outcomes produce benefits to both academic and industrial sectors.

Bibliography

- [1] L. Davis, “Dynamic origin to destination routing of wirelessly connected, autonomous vehicles on a congested network.,” *Physica A Statistical Mechanics and its Applications*, vol. 478, pp. 93-102, 2017.
- [2] S. C. A. Hevner, Design Research in Information Systems, Integrated Series in Information Systems, DOI 10.1007/978-1-4419-5653-8_1, C Springer Science Business Media, LLC., 2010.
- [3] P. J. N. C. F. a. A. P. Francisca Rosique, “A Systematic Review of Perception System and Simulators for Autonomous Vehicles Research,” *Sensors*, vol. 19, no. 3, p. 648, 2019.
- [4] “History of self-driving cars,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/History_of_self-driving_cars#cite_note-Phantom_Auto'_will_tour_city-1. [Accessed 5 January 2019].
- [5] L. Joseph, Mastering ROS for Robotics Programming, Birmingham, UK: Published by Packt Publishing, 2017.
- [6] “Self-driving car Levels of driving automation,” wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Self-driving_car#Levels_of_driving_automation, Last edited on 13 April 2019, at 14:47 (UTC).. [Accessed 12 January 2019].
- [7] D. E. Bakken, “School of Electrical Engineering & Computer Science,” [Online]. Available: <https://www.eecs.wsu.edu/~bakken/middleware.pdf>. [Accessed 15 February 2019].
- [8] “Documentation,” Open Robotics, 29 11 2018. [Online]. Available: <http://wiki.ros.org/Documentation>. [Accessed 13 January 2019].
- [9] “10th Anniversary Coverage,” Open Robotics, 5 3 2018. [Online]. Available: <http://www.ros.org/news/misc/>. [Accessed 2 April 2019].
- [10] Ayssam Elkady and Tarek Sobh, “Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography,” *Journal of Robotics*, vol. 2012, no. Article ID 959013, <http://dx.doi.org/10.1155/2012/959013>, p. 15, 2012.
- [11] R. J. F. R. A. J. S. Valter Costa, “Simulator for Teaching Robotics, ROS and Autonomous Driving in a Competitive Mindset,” *International Journal of*

Technology and Human Interaction, vol. 13, no. 4, 2017.

- [12] "What Is LabVIEW?," National Instruments, 2019. [Online]. Available: <http://www.ni.com/hu-hu/shop/labview.html>. [Accessed 12 January 2019].
- [13] "robot operating system," MathWorks, 2019. [Online]. Available: <http://www.mathworks.com/hardware-support/robot-operating-system.html>. [Accessed 13 January 2019].
- [14] C. Ltd, "Webots," Cyberbotics, 2019. [Online]. Available: <https://cyberbotics.com/>. [Accessed 14 January 2019].
- [15] "V-rep," coppeliar, 2019. [Online]. Available: <http://www.coppeliarobotics.com>. [Accessed 16 January 2019].
- [16] S. K. M. J. Majumdar, "Kinematics, Localization and Control of Differential Drive Mobile Robot," *Global Journal of Researches in Engineering: H Robotics & Nano-Tech*, vol. 14, 2014.
- [17] P. W. Z. Boru Diriba Hirpo, "Design and Control for Differential Drive Mobile Robot," *International Journal of Engineering Research & Technology (IJERT)*, ISSN: 2278-0181, vol. 6, no. 10, p. 327, 2017.
- [18] "pid-control," [Online]. Available: <https://www.autonomousrobotslab.com/pid-control.html>. [Accessed 19 January 2019].
- [19] S. M. S. B. S. D. G. Kunal Agarwal, "A Proportional-Integral-Derivative Control Scheme of Mobile Robotic platforms using MATLAB," *IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE)*, e-ISSN: 2278-1676, p-ISSN: 2320-3331, Volume 7, Issue 6, pp. 32-39, 2013.
- [20] R. S. a. I. R. Nourbakhsh, "Perception," in *Introduction to Autonomous Mobile Robots*, London, England, The MIT Press, 2004 , pp. 90-99.
- [21] "Sensors supported by ROS," Open Robotics, 24 04 2018. [Online]. Available: <http://wiki.ros.org/Sensors>. [Accessed 5 February 2019].
- [22] L. Joseph, *Learning Robotics Using Python*, Packt Publishing, 2015.
- [23] E. F. Aaron Martinez, *Learning ROS for Robotics Programming*, Birmingham, UK: Packt Publishing, 2013.
- [24] E. U. Lutfi Mutlu, "Control and Navigation of an Autonomous Mobile Robot with

- Dynamic Obstacle Detection and Adaptive Path Finding Algorithm,” *IFAC*, 2016.
- [25] W. Brian, P. OEL, L. Daniel, E. GLASER, J. L. TOGGWEILER and N. C. COSER, “Automatic driving route planning application”. US Patent WO 2014/139821 A1, 18 September 2014.
- [26] J. Zhang and S. Singh, “LOAM: Lidar Odometry and Mapping in Real-time,” *Robotics: Science and Systems*, 2014.
- [27] H. A. M. D. G. F. D. I. E. A. A. J. M. Marín P., “ROS-based architecture for autonomous vehicles,” in *Symposium SEGVAUTO-TRIES-CM Technologies for a Safe, Accessible and Sustainable Mobility*, Madrid, 2016.
- [28] A. K. H. B. A. A. M. A. a. H. Y. Imen Chaari, “Design and performance analysis of global path planning techniques for autonomous mobile robots in grid environments,” *International Journal of Advanced Robotic Systems*, no. DOI: 10.1177/1729881416663663, 2017.
- [29] M. S. D. S. A. T. Asaf Valadarsky, “A Machine Learning Approach to Routing,” arXiv , 2017.
- [30] V. K. Dalip, “Optimum Route Selection for Vehicle Navigation,” (*IJACSA*) *International Journal of Advanced Computer Science and Applications.*, vol. 7, no. 2, 2016.
- [31] S. I. a. T. A. N. Ahmadullah, “RouteFinder: Real-time optimum vehicle routing using mobile phone network,” in *TENCON 2015 - 2015 IEEE Region 10 Conference*, Macao , 2015.
- [32] A. H. ., D. M. a. A. d. I. E. Pablo Marin-Plaza, “Global and Local Path Planning Study in a ROS-Based Research Platform for Autonomous Vehicles,” *Journal of Advanced Transportation*, no. , p. 10, 2018.
- [33] S. Alexander, “On the History of the Shortest Path Problem,” Mathematics Subject Classification, , 2010.
- [34] J.-H. L. K.-D. (. Hong-Soo Kim and S. (. Youn-Suk Jeong, “Method For Finding Shortest Path To Destination In Traaffic Network Using Dljksra Algorithm Or Floyd-Warshallalgorhthm”. LOS ANGELES, CA 90025 (US) Patent US 2002/0059025A1 , 16 May 2002.

- [35] Y. L. R. B. Rui Song, "Smoothed A* algorithm for practical unmanned surface vehicle path planning," *Applied Ocean Research*, vol. 83, pp. 9-20, February 2019.
- [36] L. (. R. J. G. L. (. Marcin Michal Kmiecik, "ROUTE SMOOTHING". United States Patent US 2014/0350850 A1, 27 11 2014.
- [37] D. S. S. J. J. P. P. Victerpaul, "Path planning of autonomous mobile robots: A survey and comparison," *Journal of Advanced Research in Dynamical and Control Systems*, vol. 9, 2017.
- [38] R. Paz, "The Design of the PID Controller," 30 April 2014. [Online]. Available: <https://www.researchgate.net/publication/237528809>. [Accessed 9 March 2019].
- [39] E. M. C. U. B. A. G. a. L. G. Mümin Tolga Emirler, "Robust PID Steering Control in Parameter Space for Highly Automated Driving," *International Journal of Vehicular Technology*, 2014.
- [40] S. Blazic, "On Periodic Control Laws for Mobile Robots," *IEEE Transactions On Industrial Electronics*, vol. 61, no. 7, 2014.
- [41] W. F. W. W. a. Z. W. Gaining Han, "The Lateral Tracking Control for the Intelligent Vehicle Based on Adaptive PID Neural Network," *Sensors* , 2017.
- [42] J. P.-O. ., B. M. A.-H. ., a. A. J. Luciano Alonso, "Self-Tuning PID Controller for Autonomous Car Tracking in Urban Traffic," in *17th International Conference on System Theory, Control and Computing (ICSTCC)*, Sinaia, Romania, 2013.
- [43] H. T. ., Y. H. a. B. T. Haobin Jiang, "Research on Control of Intelligent Vehicle Human-Simulated Steering System Based on HSIC," *Applied Sciences*, 2019.
- [44] N. N. W. Cherry Myint, "Position and Velocity control for Two-Wheel Differential Drive Mobile Robot," *International Journal of Science, Engineering and Technology Research (IJSETR)*, vol. 5, no. 9, 2016.
- [45] S. &. G. E. &. K. R. Bouzoualegh, "Model Predictive Control of a Differential-Drive Mobile Robot," *Acta Universitatis Sapientiae Electrical and Mechanical Engineering*, vol. 10, pp. 20-41, 2018.
- [46] "Ubuntu user statistics," [Online]. Available: <https://www.ubuntu.com/desktop/statistics#user-report>. [Accessed 20 January 2019].
- [47] "Internet-of-things," 2019 Canonical Ltd. Ubuntu and Canonical are registered

- trademarks of Canonical Ltd., [Online]. Available:
<https://www.ubuntu.com/internet-of-things/robotics>. [Accessed 23 January 2019].
- [48] “Robot Operating System,” From Wikipedia, the free encyclopedia, 10 April 2019. [Online]. Available: https://en.wikipedia.org/wiki/Robot_Operating_System. [Accessed 20 January 2019].
- [49] “Core Components,” Open robotics, [Online]. Available: <http://www.ros.org/core-components/>. [Accessed 5 February 2019].
- [50] “ROS Introduction,” Open robotics, 08 08 2018. [Online]. Available: <http://wiki.ros.org/ROS/Introduction>. [Accessed 7 February 2019].
- [51] L. Joseph, ROS Robotics Projects, : Packt Publishing Limited, 2017.
- [52] “Rviz,” Open Source Robotics Foundation, 24 09 2015. [Online]. Available: <http://wiki.ros.org/Tools>. [Accessed 10 February 2019].
- [53] “GAZEBO,” Open Source Robotics Foundation, 2014. [Online]. Available: <http://gazebo.org/>. [Accessed 12 February 2019].
- [54] L. C. T. R. J. B. a. J. G. P. Wei, “LiDAR and Camera Detection Fusion in a Real-Time Industrial Multi-Sensor Collision Avoidance Syatem,” *Electronics*, <https://doi.org/10.3390/electronics7060084>, vol. 7, no. 6, p. 84, May,2018.
- [55] “ROS control,” Open Source Robotics Foundation, 2014. [Online]. Available: http://gazebo.org/tutorials/?tut=ros_control. [Accessed 20 February 2019].
- [56] “PID,” Open Robotics, 25 05 2018. [Online]. Available: http://wiki.ros.org/pid#Controller_Node. [Accessed 4 March 2019].
- [57] “differential_drive,” Open Robotics, 2018. [Online]. Available: http://wiki.ros.org/differential_drive. [Accessed 21 February 2019].
- [58] “diff_drive_controller,” Open Robotics, 20 01 2018. [Online]. Available: http://wiki.ros.org/diff_drive_controller. [Accessed 21 February 2019].
- [59] “Gazebo plugins in ROS,” Open Robotics., 2018. [Online]. Available: http://gazebo.org/tutorials?tut=ros_gzplugins. [Accessed 25 February 2019].
- [60] “Using the low-level robot base controllers to drive the robot,” Open Robotics, 2018. [Online]. Available: http://wiki.ros.org/pr2_controllers/Tutorials/Using%20the%20robot%20base%20co

- ntrollers%20to%20drive%20the%20robot. [Accessed 26 February 2019].
- [61] “controller_manager,” Open Robotics, 21 04 2018. [Online]. Available: http://wiki.ros.org/controller_manager. [Accessed 28 February 2019].
- [62] M. B. a. C. S. D. Maier, “Self-supervised Obstacle Detection for Humanoid Navigation Using Monocular Vision and Sparse Laser Data,,” in *Proceedings of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, Shanghai, China, 2011 .
- [63] “Setup and Configuration of the Navigation Stack on a Robot,” Open robotics, 19 07 2018. [Online]. Available: http://wiki.ros.org/navigation/Tutorials/RobotSetup#Navigation_Stack_Setup. [Accessed 21 February 2019].
- [64] “Scanning Laser Range Sensor UTM-30LX Specification,” 13 1 2010. [Online]. Available: http://wiki.ros.org/hokuyo_node?action=AttachFile&do=view&target=UTM-30LX_Specification.pdf. [Accessed 27 Feb. 2019].
- [65] M. I. Kristin Gross, “Advantages and Limitations of Ultrasonic Sensors,” 2017. [Online]. Available: <https://www.maxbotix.com/articles/advantages-limitations-ultrasonic-sensors.htm/>. [Accessed 7 March 2019].
- [66] “hokuyo_node,” Open Robotics, 20 11 2018. [Online]. Available: http://wiki.ros.org/hokuyo_node. [Accessed 26 Feb. 2019].
- [67] “image_pipeline,” Open Robotics, 30 10 2018. [Online]. Available: http://wiki.ros.org/image_pipeline?distro=melodic. [Accessed 26 February 2019].
- [68] G. Grisetti, C. Stachniss and W. Burgard, “gmapping,” [Online]. Available: <https://openslam-org.github.io/gmapping.html>. [Accessed 27 February 2019].
- [69] “Building a Map for the Navigation Stack,” Open Robotics, 2018. [Online]. Available: <http://wiki.ros.org/navigation/MapBuilding>. [Accessed 27 Feb. 2019].
- [70] “map_server,” Open Robotics, 07 02 2019. [Online]. Available: http://wiki.ros.org/map_server. [Accessed 1 March 2019].
- [71] S. S. a, Bayesian Filtering and Smoothing, Cambridge University Press , 2013.
- [72] “gmapping,” Open Robotoics, 04 02 2018. [Online]. Available: <http://wiki.ros.org/gmapping>. [Accessed 27 February 2019].
- [73] “base_local_planner,” Open Robotics, 04 04 2019. [Online]. Available:

- http://wiki.ros.org/base_local_planner?distro=melodic. [Accessed 2 March 2019].
- [74] “global_planner,” Open Robotics, 29 06 2018. [Online]. Available: http://wiki.ros.org/global_planner?distro=melodic. [Accessed 3 March 2019].
- [75] M. I. I. S. M. (. P. I. I. Jayaraman Seshan (Student, “Efficient Route Finding and Sensors for Collision Detection in Google’s Driverless Car,” *International Journal of Computer Science and Mobile Computing, IJCSMC*, vol. 3 , no. 12, p. 70 – 78, 2014.
- [76] “amcl,” Open Robotics, 24 9 2018. [Online]. Available: <http://wiki.ros.org/amcl?distro=melodic>. [Accessed 3 March 2019].
- [77] “nav_msgs/Odometry Message,” Open Robotics, 09 11 2018. [Online]. Available: http://docs.ros.org/api/nav_msgs/html/msg/Odometry.html. [Accessed 12 February 2019].
- [78] “nav_core,” Open Robotics, 14 06 2018. [Online]. Available: http://wiki.ros.org/nav_core. [Accessed 28 Feb. 2019].
- [79] “move_base,” Open Robotics, 27 09 2018. [Online]. Available: http://wiki.ros.org/move_base. [Accessed 26 February 2019].
- [80] “Tutorial: Using a URDF in Gazebo,” Open Robotics, 2018. [Online]. Available: http://gazebosim.org/tutorials?tut=ros_urdf&cat=connect_ros. [Accessed 20 February 2019].
- [81] “Tutorial: Ros Plugins,” Open Robotics, 2018. [Online]. Available: http://gazebosim.org/tutorials/?tut=ros_plugins. [Accessed Feb. 2019].
- [82] T. Foote, “tf: The transform library,” in *Technologies for Practical Robot Applications (TePRA)*, Mountain View, CA 94043, IEEE International Conference on Open-Source Software workshop, 2013, p. .
- [83] “catkin/workspaces,” Open robotics., 07 07 2017. [Online]. Available: <http://wiki.ros.org/catkin/workspaces>. [Accessed 10 January 2019].
- [84] “TurtleBot,” Open robotics, 04 04 2018. [Online]. Available: <http://wiki.ros.org/Robot/TurtleBot>. [Accessed 3 February 2019].
- [85] “Tutorials,” Open Robotics, 2018. [Online]. Available: <http://wiki.ros.org/ROS/Tutorials>. [Accessed 10 January 2019].

- [86] “rospy Overview,” Open Robotics, 2018. [Online]. Available: <http://wiki.ros.org/rospy/Overview>. [Accessed 12 January 2019].
- [87] “Robotic simulation scenarios with Gazebo and ROS,” Generation Robots, [Online]. Available: <https://www.generationrobots.com/blog/en/robotic-simulation-scenarios-with-gazebo-and-ros/>. [Accessed 10 January 2019].
- [88] “Building a Visual Robot Model with URDF from Scratch,” Open Robotics., 2018. [Online]. Available: <http://wiki.ros.org/urdf/Tutorials/Building%20a%20Visual%20Robot%20Model%20with%20URDF%20from%20Scratch>. [Accessed 11 January 2019].
- [89] stackoverflow, “ros-catkin-make-rebuild-packages,” [Online]. Available: <https://stackoverflow.com/questions/46867518/ros-catkin-make-rebuild-packages>. [Accessed 18 January 2019].
- [90] “[rospack] Error: package 'beginner_tutorials' not found,” Open Robotics., 2018. [Online]. Available: https://answers.ros.org/question/224833/rospack-error-package-beginner_tutorials-not-found/. [Accessed 20 January 2019].
- [91] “Running an Action Client and Server,” Open Robotics, 2018. [Online]. Available: http://wiki.ros.org/actionlib_tutorials/Tutorials/RunningServerAndClient. [Accessed 15 February 2019].
- [92] “Why can't ROS find this file?,” Open Robotics., 2018. [Online]. Available: <https://answers.ros.org/question/241644/why-cant-ros-find-this-file/>. [Accessed 10 January 2019].
- [93] “catkinCMakeLists.txt,” Open Robotics, 2018. [Online]. Available: <http://wiki.ros.org/catkin/CMakeLists.txt>. [Accessed 7 January 2019].
- [94] “turtlebot_simulator,” Open Robotics, 2015. [Online]. Available: http://wiki.ros.org/turtlebot_simulator. [Accessed 28 February 2019].
- [95] “turtlebot simulator,” [Online]. Available: https://github.com/turtlebot/turtlebot_simulator/tree/indigo/turtlebot_gazebo/launch. [Accessed 16 February 2019].
- [96] “Building a Movable Robot Model with URDF,” Open robotics, 29 11 2018. [Online]. Available:

- <http://wiki.ros.org/urdf/Tutorials/Building%20a%20Movable%20Robot%20Model%20with%20URDF>. [Accessed 10 February 2019].
- [97] “Adding a SensorPlugin,” Open Source Robotics Foundation Gazebo, [Online]. Available: http://gazebo-sim.org/tutorials?tut=ros_gzplugins#AddingaSensorPlugin. [Accessed 26 February 2019].
- [98] “ros/common_msgs,” [Online]. Available: https://github.com/ros/common_msgs. [Accessed 8 February 2019].
- [99] “HumaRobotics/mybot_gazebo_tutorial,” [Online]. Available: https://github.com/HumaRobotics/mybot_gazebo_tutorial. [Accessed 13 February 2019].
- [100] “Hokuyo Laser Scanner not working in Gazebo using ROS Kinetic,” Open robotics, [Online]. Available: <https://answers.ros.org/question/250556/hokuyo-laser-scanner-not-working-in-gazebo-using-ros-kinetic/>. [Accessed 13 February 2019].
- [101] “richardw05/mybot_ws,” [Online]. Available: https://github.com/richardw05/mybot_ws/commit/2fc9d018501ac26ff49cb345ffa27a1b15dab92e. [Accessed 27 February 2019].
- [102] “laserscanner-look-through-objects,” [Online]. Available: www.gazebo-sim.org/question/13497/laserscanner-look-through-objects/. [Accessed 13 February 2019].
- [103] “Attach plugin to included laser model/15281/,” [Online]. Available: <http://answers.gazebo-sim.org/question/15281/attach-plugin-to-included-laser-model/>. [Accessed 22 February 2019].
- [104] “Simulating sensors in gazebo,” [Online]. Available: https://github.com/bgloh/mybot_ws/wiki/Simulating-sensors-in-gazebo. [Accessed 13 February 2019].
- [105] “leandroS08/semeiar_ptr,” Open Robotics, 2018. [Online]. Available: https://github.com/leandroS08/semeiar_ptr/blob/master/src/hokuyo_obstacle.py. [Accessed 15 Feb. 2019].
- [106] “ROS Tutorial on Robot Simulation in Gazebo,” [Online]. Available: <https://github.com/SMARTlab-Purdue/ros-tutorial-gazebo->

- simulation/blob/master/README_full.md. [Accessed 27 February 2019].
- [107] “Adding a lidar to the turtlebot using hector_models (Hokuyo UTM-30LX),” Open robotics, 2018. [Online]. Available: http://wiki.ros.org/turtlebot/Tutorials/indigo/Adding%20a%20lidar%20to%20the%20turtlebot%20using%20hector_models%20%28Hokuyo%20UTM-30LX%29, last edited by Vikrant Shah at 2017-08-01 17:52:51. [Accessed 27 February 2019].
 - [108] “roslib,” Open Robotics, 31 12 2013. [Online]. Available: <http://wiki.ros.org/roslib>. [Accessed 26 February 2019].
 - [109] “Writing a Simple Service and Client (Python),” Open Robotics., 2018. [Online]. Available: <http://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28python%29>. [Accessed 13 January 2019].
 - [110] “How to get turtlebot to move in gazebo?,” [Online]. Available: <https://answers.ros.org/question/315020/how-to-get-turtlebot-to-move-in-gazebo/>. [Accessed 27 February 2019].
 - [111] “Writing a Simple Action Client,” Open Robotics., 2018. [Online]. Available: http://wiki.ros.org/actionlib_tutorials/Tutorials/SimpleActionClient. [Accessed 26 February 2019].
 - [112] “geometry_msgs/Twist Message,” Open Robotics, 09 11 2018. [Online]. Available: http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html. [Accessed 27 February 2019].
 - [113] “teleop_twist_keyboard,” Open Robotics, 22 01 2015. [Online]. Available: http://wiki.ros.org/teleop_twist_keyboard. [Accessed 28 February 2019].
 - [114] “Publish on /My_robot/cmd_vel using teleop,” Open Robotics, [Online]. Available: https://answers.ros.org/question/246095/publish-on-my_robotcmd_vel-using-teleop/. [Accessed 28 February 2019].
 - [115] “turtlebot_teleop,” Open Robotics, 08 01 2015. [Online]. Available: http://wiki.ros.org/turtlebot_teleop. [Accessed 28 February 2019].
 - [116] “ros-planning/moveit,” Open robotics, [Online]. Available: <https://github.com/ros-planning/moveit/tree/melodic-devel>. [Accessed 25 February 2019].
 - [117] “ROS Depth Camera Integration,” [Online]. Available:

- http://gazebosim.org/tutorials?tut=ros_depth_camera&cat=connect_ros. [Accessed 20 February 2019].
- [118] “Could not find executable running image_view from rosrn,” Open robotics, [Online]. Available: https://answers.ros.org/question/182030/could-not-find-executable-running-image_view-from-rosrn/. [Accessed 28 February 2019].
 - [119] “adding-obstacles-to-map-during-robot-turtlebot-navigation,” [Online]. Available: <https://answers.ros.org/question/188594/adding-obstacles-to-map-during-robot-turtlebot-navigation/>. [Accessed 18 February 2019].
 - [120] “Range sensor does not detect objects in gazebo,” [Online]. Available: <https://answers.ros.org/question/291726/range-sensor-does-not-detect-objects-in-gazebo/>. [Accessed 19 February 2019].
 - [121] “Adding Hokuyo Laser Finder to Turtlebot in Gazebo Simulation,” [Online]. Available: <https://bharat-robotics.github.io/blog/adding-hokuyo-laser-to-turtlebot-in-gazebo-for-simulation/>. [Accessed 11 February 2019].
 - [122] “Guide for Developers Interested in Robotics,” Developed with TurtleBot.com & ROS.org, [Online]. Available: <http://learn.turtlebot.com/>. [Accessed 16 February 2019].
 - [123] “Editing the Simulated World,” [Online]. Available: <http://learn.turtlebot.com/2015/02/03/6/>. [Accessed 26 February 2019].
 - [124] “Using Xacro to Clean Up a URDF File,” Open Robotics, 2018. [Online]. Available: <http://wiki.ros.org/urdf/Tutorials/Using%20Xacro%20to%20Clean%20Up%20a%20URDF%20File>. [Accessed 8 February 2019].
 - [125] “Using a URDF in Gazebo,” Open Robotics, 2018. [Online]. Available: <http://wiki.ros.org/urdf/Tutorials/Using%20a%20URDF%20in%20Gazebo>. [Accessed 8 February 2019].
 - [126] “Gazebo differential drive vehicle model,” [Online]. Available: <https://discourse.ros.org/t/gazebo-differential-drive-vehicle-model/3052>. [Accessed 26 February 2019].
 - [127] “URDF,” Open Robotics, [Online]. Available: <http://wiki.ros.org/urdf>. [Accessed 10

- February 2019].
- [128] “Learning URDF Step by Step,” Open Robotics, 2018. [Online]. Available: <http://wiki.ros.org/urdf/Tutorials>. [Accessed 11 Feb. 2019].
 - [129] “ros-planning/navigation_tutorials,” 2019. [Online]. Available: https://github.com/ros-planning/navigation_tutorials. [Accessed 19 February 2019].
 - [130] “How to Build a Map Using Logged Data,” Open robotics, 2018. [Online]. Available: http://wiki.ros.org/cn/slam_gmapping/Tutorials/MappingFromLoggedData. [Accessed 28 February 2019].
 - [131] “ros-perception/openslam_gmapping,” [Online]. Available: https://github.com/ros-perception/openslam_gmapping. [Accessed 12 February 2019].
 - [132] “317711/gmapping-ros-melodic-error/,” Open Robotics, 2019. [Online]. Available: <https://answers.ros.org/question/317711/gmapping-ros-melodic-error/>. [Accessed 4 March 2019].
 - [133] “Problem with gmapping launch file,” [Online]. Available: <https://answers.ros.org/question/225227/problem-with-gmappinglaunch-file/>. [Accessed 28 February 2019].
 - [134] “slam_gmapping,” Open robotics, 2018. [Online]. Available: http://wiki.ros.org/slam_gmapping. [Accessed 26 February 2019].
 - [135] “melodic gmapping,” [Online]. Available: <https://answers.ros.org/question/296170/melodic-gmapping/>. [Accessed 27 February 2019].
 - [136] “Wiki:slam_gmapping,” [Online]. Available: http://www.ros.org/wiki/slam_gmapping. [Accessed 26 February 2019].
 - [137] “Navigation Stack with gmapping,” Open robotics, 2018. [Online]. Available: <https://answers.ros.org/question/9432/navigation-stack-with-gmapping/>. [Accessed 23 February 2019].

- [138] “Creating a Map,” [Online]. Available: <http://learn.turtlebot.com/2015/02/03/8/>. [Accessed 3 March 2019].
- [139] “The PR2 Robot,” Open robotics, [Online]. Available: <https://robots.ros.org/PR2>. [Accessed 2 March 2019].
- [140] “Basic Pathfinding Explained With Python,” [Online]. Available: <https://www.codementor.io/blog/basic-pathfinding-explained-with-python-5pil8767c1>. [Accessed 3 March 2019].
- [141] “GMapping,” [Online]. Available: <https://openslam-org.github.io/gmapping.html>. [Accessed 12 February 2019].