ABRAHAM AYELE GELAN AUTONOMOUS SEARCH AND RESCUE ROBOT USING ROS PLATFORM NEU 2019

AUTONOMOUS SEARCH AND RESCUE ROBOT USING ROS PLATFORM

A THESIS SUBMITTED TO THE GRADUATE SCHOOL OF APPLIED SCIENCES OF NEAR EAST UNIVERSITY

By
ABRAHAM AYELE GELAN

In Partial Fulfilment of the Requirements for the Degree of Master of Science in Mechatronics Engineering

NICOSIA, 2019

AUTONOMOUS SEARCH AND RESCUE ROBOT USING ROS PLATFORM

A THESIS SUBMITTED TO THE GRADUATE SCHOOL OF APPLIED SCIENCES OF NEAR EAST UNIVERSITY

By
ABRAHAM AYELE GELAN

In Partial Fulfilment of the Requirements for the Degree of Master of Science in Mechatronics Engineering

NICOSIA, 2019

ABRAHAM AYELE GELAN: AUTONOMOUS SEARCH AND RESCUE ROBOT USING ROS PLATFORM

Approval of Director of Graduate School of Applied Sciences

Prof. Dr. Nadire ÇAVUŞ

We certify this thesis is satisfactory for the award of the degree of Masters of Science in Mechatronics Engineering

Examining Committee in Charge:

Prof.Dr. Ahmet Denker	Supervisor, Department of Mechatronics Engineering, NEU
Prof.Dr. Bulent Bigeham	Department of Electrical and Electronic Engineering, NEU
Assist.Prof.Dr. Perveneh Esmaili	Department of Electrical and Electronic Engineering, NEU

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original
to this work.

Name, Last name: ABRAHAM AYELE GELAN
Signature:
Date: august 7, 2019

ACKNOWLEDGMENT

I am very grateful for all the people that showed me love, kindness, and respect in my two-year stay in Cyprus. Special command goes to my family and friends for their love and support. I would like to give lots of thanks for betrescience scholarship for giving me this opportunity to pursue my masters. I acknowledge all the teachers for sharing their unreserved knowledge and help me achieve my full potential on my course work. I would like to express my sincere gratitude and thanks to my beloved supervisor, Prof. Dr. Ahmet Denker for continuous guidance and assistance during my Work on this Thesis.

To ST. John…

ABSTRACT

Building an intelligent robotic system that can maneuver in the unknown unpleasant environment can be difficult in urban search and rescue research. The robot built required to take sensor data, for building a map of the unknown environment to get knowledge of existing landmarks and to figure out its position and orientation inside, the problem named SLAM. Additionally, the autonomous system needs motion planning the systematic way to avoid the obstacle by taking SLAM as input to control the movement it takes around occupied spaces. In order to enhance its capability, autonomous victim detection system integration is necessary. Several pieces of research are done nowadays stipulating a lot of algorithms for achieving autonomous navigation system and object detection. The defacto platform for robot design is ROS (Robot operating system) which provide all-rounded packages for rigorous implementation of the robotic system. On this research, we analyze and implemented algorithms out there for our mission. we see into advanced SLAM, motion planning and victim detection algorithm' s for search and rescue mission. Analyzing ROS, we found out HECTOR mapping, AMCL, ROS Global Planer and TEB local Planer in combination can give robust result comparative from other existing slam and motion planning algorithms on the platform. Victim detection is achieved successfully using YOLO V3 which is the present state-of-the-art deep learning object detection algorithm.

Keywords: urban search and rescue; Robot operating system; SLAM; Motion planning; YOLO V3

ÖZET

Bilinmeyen tatsız çevrede manevra yapabilen akıllı bir robot sistemi oluşturmak, kentsel arama ve kurtarma araştırmalarında zor olabilir. SLAM adlı problemi, sensör verilerini almak, var olan yerlerden bilgi edinmek için bilinmeyen bir ortam haritası oluşturmak ve SLAM adlı problemin içindeki konumunu ve yönünü bulmak için gerekli kılıyor. Ek olarak, özerk sistem, işgal edilmiş alanların etrafındaki hareketleri kontrol etmek için girdi olarak SLAM'ı alarak engeli önlemek için sistematik bir yol planlama hareketine ihtiyaç duyar. Yeteneğini geliştirmek için, özerk mağdur tespit sistemi entegrasyonu gereklidir. Bugünlerde, özerk navigasyon sistemi ve nesne tespitine ulaşmak için birçok algoritma şart koşan birçok araştırma yapılır. Robot tasarımı için defacto platformu, robot sisteminin titiz bir şekilde uygulanması için çok yönlü paketler sağlayan ROS'tur (Robot işletim sistemi). Bu araştırmada misyonumuz için algoritmaları analiz edip uyguladık. arama kurtarma görevinde gelişmiş SLAM, hareket planlama ve mağdur tespit algoritması görüyoruz. ROS'u analiz ettik, HECTOR mapping, AMCL, ROS Global Planer ve TEB local Planer'ın kombinasyon halinde kombinasyonunu, platformdaki diğer mevcut çarpma ve hareket planlama algoritmalarına kıyasla sağlam bir sonuç verebileceğini gördük. Kurban tespiti, mevcut derin öğrenme nesnesi algılama algoritması olan YOLO V3 kullanılarak başarılı bir şekilde gerçekleştirilir.

Anahtar Kelimeler: kentsel arama kurtarma; Robot işletim sistemi; eşzamanlı lokalizasyon ve haritalama; Hareket planlama; YOLO V3

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

USAR:	Urban search and rescue
SLAM:	Simultaneous localization and mapping
ROS:	Robot operating system
DWA:	Dynamic window approach
FPN:	Feature pyramid network
R-CNN:	Recursive convolution neural network
SSD:	Single shot detector
TF:	Transform frame
Urdf:	Universal robot description format
SDF:	Simulation description format
TEB:	Time elastic band
Eband:	Elastic band
AMCL:	Adaptive Monte Carlo localization
YOLO V3:	You only look once version 3
C-space:	Configuration space

CHAPTER 1
INTRODUCTION

1.1 Overview

The use of robots in USAR could be a difficult space for researchers in artificial intelligence as a result of Robots employed in search and rescue want mobility and robustness(Tadokoro, 2009). The scenario in which they will be used is harsh with many unknowns. There are two fundamental subjects that feed each other to accomplish robot autonomous navigation are SLAM (Scaramuzza et al. 2016) and motion planning (LaValle, 2011). SLAM is an analytical problem of creating or recording a map of an undiscovered environment while concurrently updating the robot's position and orientation within it (Scaramuzza et al. 2016). Grid base and visual-based slam are most popular applicable nowadays the most dominant one being the first.

In ROS there are several slam algorithms based on grid-based approach(Santos and Rocha, 2011), gmapping which uses particle filter as a core pose estimator (GvdHoorn, 2018) and hector mapping uses the scan match technique (StefanKohlbrecher n.d.) are widely used. Hector slam represent the environment as Occupancy grid map(Stefan Kohlbrecher, Oskar v, Johannes M, 2011). Occupancy grid map categorized as metric map uses arrays of binary cells (matrix) to show the surrounding world and create estimation about which part of the cell is occupied (Thrun n.d.). Motion planning, on the other hand, uses the map created by the robot to plan a way to accomplish a goal avoiding obstacle and optimizing the speed (LaValle, 2011). Additional we analyze the implementation of deep learning algorithms for detecting victim in unknown clustered USAR environment. Robot operating system provides all this functionality and infrastructure by its different package with certain preconditions. So, this research will be going to address how we can accomplish autonomous navigation and victim detection using ROS by analyzing the existing state of art algorithms for search and rescue mission.

1.2 Problem Of Statement

Robots are playing a vital role in our day to day life in transportation, service, industries and in research as the advancement in the technology is growing fastly. Nowadays robots are capable of doing what human incapable of because robots can be designed uniquely to address the difficult situation. Robot autonomous capabilities are significantly grown in the past decades thus making the robot to do thing by the own in a harsh and hazardous environment. One area at which robot autonomism required is in USAR mission in a disaster environment. Search and rescue environment is a dynamic environment that anything can happen unexpectedly this make things difficult for human exploration some times. In addition, there are areas that human can't reach due to varies unpleasant conditions, therefore, robot value in this situation is undeniable.

One of the deeply studied areas in a robotics automation system is robot autonomous navigation which deals with planning and moving to the goal of avoiding obstacle around the environment. On this research, we are focusing on autonomous navigation system In ROS (robot operating system) platform. ROS provides varies nodes and packages in order to achieve full autonomy. Taking this advantage, we are aiming to analyze the ROS navigation system for our purpose which is search and rescue mission. we aim to simulate and analyze the ROS navigation stack using hocuyo lidar scan and skid steering robot which is a favorite model for the unpleasant environment because of its ability to turn fast and accurately. The camera sensor is integrated into the system for the victim detection system. To achieve a robust goal, we are going to analyze suitable object detection, slam, and move_ base package considering search and rescue environmental conditions on ubuntu 18.04 ROS melodic system by creating URDF (universal robot format) and SDF file for the world and robot description in ROS and gazebo simulation platform. We hope this work will be beneficial for using ROS navigation stack for real search and rescue robot as it assists to understand the tuning and the work behind the seen in within different chapters.

1.3 General Objective

- ✓ To analyze existing ROS algorithm's in navigation system which suits search and rescue robotic systems
- ✓ Implementing victim detection in USAR environment
- ✓ Demonstrate the result using simulation

1.4 Specific Objective

- ✓ Selecting implement suitable slam algorithm for search and rescue mission
- ✓ Creating a robot model and simulation environment for simulation
- ✓ Localizing robot in the environment
- ✓ Analyzing and implementing suitable path planning algorithm
- ✓ Analyzing and implementing suitable victim detection algorithm

1.5 Application And Advantage

Our focus in this work is to analyze the robot capability before the robot touches the ground using the simulated robot on the ROS platform. This will benefit to know the behavior of the real robot beforehand. ROS is the de_facto programming platform in robotics to achieve greater capability with reduced time. Additionally, our work will give insight ROS navigation stack for search and rescue experimentation. As the simulation is exactly similar to real work the reader be forced to understand the way to use the navigation stack of ROS platform.

CHAPTER 2
SEARCH AND RESCUE ROBOTICS

2.1 Introduction

A rescue mission is dependably a competition against time, to attain as quick as conceivable to reach every single survivor candidate but then maneuver gradually enough to avoid creating additional falls, distress, or damages to rescuers and unfortunate casualties (Murphy, 2013). Ideally, the general objective for rescue robotics is to reduce potential harm, the objective for particular automation styles and ability relies on their given work. The type of work that has been given for rescue robots can be searching, reconnaissance, creating maps, west removal, inspection, medical assessment, extraction and evaluation of causality providing logistic supports(Murphy, 2013). The kind of disaster impacts the decision of robot plat-structures and payloads. Rescue robots can be extensively classified into sorts dependent on modality and size. There are four applicable models of rescue robots: ground, aerial, underwater, and water surface. The disaster characteristics and Impact on robotics in case of man-made and natural disasters their promising robot design are like snakes, legged locomotion(Murphy, 2013). extensive research was done japan project called DDT, this research addresses the search and rescue, the impact of natural and artificial disasters and the role robot can play to reduce the consequence. It gives a deep intuition on the disaster information gathering, the material used, design methods. The design varies with the type of robot models and some background perceptions were discussed (Tadokoro, 2009)

2.2 Why Autonomous Search and Rescue Robot

The first applicable proposal for utilization of small robotic model in USAR performed over victim rehabilitation mission in the repercussions of the Oklahoma town disaster (Blitch, 1996). Generally, USAR is full of cluttered area’s and every robot that is implemented in this condition don’t have data beforehand about obstacles in the surrounding. Therefore, the task is deeply challenging for a vehicle to autonomously move around the environment and detect casualties. Therefore, presently most implementations of robots in USAR task demand

human interventions in the system to assist a robot manually(Denker and Işeri, 2017). Mobile communication issues and fatigue problem can arrive. Moreover, an operating person can turn out to be easily pressurized and exhausted in USAR due to surroundings, making mistakes in decision making (Burke et al. 2004). Semi-autonomous control strategy has been put forward that allows the system to be separated between a vehicle and an operator(Denker and Işeri, 2017).But to get a robust solution fully Autonomous USAR is very crucial.

2.3 SLAM For Search and Rescue Robot

With the development of robotics research in these late years, autonomous robotic systems are being designed to aid rescue personnel in USAR operations. Starting 2000's Robocop rescue is playing an important role in search and rescue research. Recently several modules of an algorithm for achieving greatly capable autonomous USAR robot algorithm is presented in ROS(Kohlbrecher et al. 2017). They incorporate hector slam for implementing SLAM in clustered urban environmental. They show a new algorithm for SLAM problem which needs to be settled to produce adequately exact metric maps valuable for navigation of models or a robot framework by name called hector slam(Kohlbrecher et al. 2017). As odometry is highly untrusted in USAR situations, the framework is intended to not require odometry information, rather absolutely depending on quick LIDAR information scan matching at full LIDAR update rate. In combination with altitude prediction framework and a discretionary pitch/yaw part to balance out the laser scanner, the framework can give a map of the area regardless of whether the movement plane is non-level as experienced in the Robocop scenarios. This can be considered one of the best advantages of the strategy-making hector slam to turn into the accepted standard Slam framework utilized by numerous groups with extraordinary achievement in Robocop tournament. Additionally, hector exploration planner and simulation with gazebo environment is briefly explained(Kohlbrecher et al. 2017).

2.4 Motion Planning

In addition to slam, search and rescue robotics require motion planning to move secure and systematically in the heavenly clustered world to achieve their assignments. This problem brings the principle challenge to consider navigation strategies that can be implemented to

various robot kinematics, environments, and targets. A fundamental comprehension of the analysis problems and kind of outlooks to get solution of the basic motion planning problem had been given here (LaValle, 2011). Despite the fact that the motion planning problem lives in continuous C-space, is an analysis done digitally. To have a realistic or applicable solution we need to perform digitalization of our system first to solve the problem. This has prompted two fundamental ways of thinking combinatorial planning, creating a model in the C-space that digitally and totally include entire data necessary to implement planning. Sampling-based planning utilizes collision detection computations to test and updatly find the C-space for the result, as opposed to totally describing the majority of the Cfree structure. The subsequent methodology is most broadly utilized but, the first is far best with numerous qualities.

2.5 ROS Navigation System

ROS initially created in 2007 at the Stanford Artificial Intelligence, today used by many robots, universities and companies and it been the thesaurus standard for robot programming. It has an integral role in Process management, Inter-process communication, Device drivers, Simulation, Visualization, Graphical user interface, Data logging, Control, Planning, Perception, Mapping and Manipulation of the robotic system. Especially for the navigation system, ROS provides varies motion planning algorithms separately for global and local planners. The principal Global Planer in ROS is navfn gives a quickly added path planning which will be utilized to make plans for robotic platform, path planning is processed with the Dijkstra's Algorithm (Jihoonl, 2014). The global_planner package provided as a progressively adaptable substitution to navfn providing an application of a higher, interpolated global planner for movement of the robot (Reinzor, 2019). The carrot_planner is a basic global planner, the planner acquires any goal coordinate from an outer client, checks if the user-determined objective is an obstacle, and on the off chance that it will be, it strolls around the vector of the user indicated objective and the robot base (NickLamprianidis, 2014).

The default local planer for ROS is base_local_planner. It is the implementation of the Trajectory Rollout and Dynamic-Window algorithms to deal with robot local planning

problem. Utilizing slam, the planner makes a kinematic calculation for the platform to arrive from a beginning to an objective area. In the path, the planner finds free space, away around for the robot base by solving the cost function, which utilizes a binary map of the environment. The cost function detects the expenses of crossing the entire binary grids. The planner main responsibility is to utilize the analysis of cost function to decide linear and angular speeds given to the robot (NicolasVaras, 2014). The dwa_local_planner gives a similar capacity as the DWA however is diverse in the method environmental inspection of robot performed. (AchmadFathoni, 2018). Eband_local_planner actualizes a module to the moving base local planner (FelixWidmaier, 2018). It permits advancing a global plan arrangement locally, limiting the length of the way and fending off it from obstacles while in the meantime considering moving obstacles. Favorable position of this methodology is that they optimize the plan incrementally. That is, the longer the robot moves, the better the trajectory path will be. It is intended for use with omnidirectional and differential drive robots (Quinlan and Khatib, 2002). The teb_local_planner algorithm actualizes a module to the local planner of the 2D move base architecture (ChristophRoesmann, 2019). The teb_local_planner or Timed Elastic Band locally enhances the created global arrangement optimizing the direction execution time. It is consistency with Kino dynamic requirements, including, satisfying most extreme speeds and increasing velocities at runtime. It is intended for robots that have a turning circle some of which are a vehicle or articulated robot (Christoph Rösmann, 2017).

Research aiming to incorporate and investigate the implementation of a navigation system of ROS based on the time elastic band (TEB) in real robot model based on Ackerman kinematics' analyzed here (Marin-Plaza et al. 2018). They show due to the continuous update property of the planner TEB is a comparatively convenient preference over the addition of kinematic model change and if the vehicle misses the path. Experimenting by real vehicle platform takes time and consume resources as the availability of a suitable environment and robot dynamic properties have to be accounted for(Mcleod, 2018). On this paper, the authors demonstrate the simulation of two steering methods namely differential and skid steering using the navigation stack of ROS for autonomous navigation. As tuning of the navigation stack in a real robot is costly and dangerous the paper analyzes how to perform SLAM and

move_base of navigation stack using ROS platform and gazebo environment (Mcleod, 2018).

2.6 Victim Detection

Finding victim in USAR environment is a challenging problem as the robot is subjected to maneuver in the undiscovered environment and find victims lying or trapped under rubble whose body may be partially visible. Classical learning approaches developed for USAR environment focused on first extracting set handcrafted features or histogram of oriented gradients then trained supervised learning model(Louie and Nejat, 2013). Doing manual design requires empirical selection and validation, which is time-consuming and require expert knowledge. Additionally, it requires a predefined rule to analyze the grouping of human body parts. As deep learning approach take sensory data as its input instead of handcrafted features it can reduce the above classical learning problems. There are two main approaches to design the architecture of person recognition with deep learning(Tsung-Yi Lin et al. 2017). The first is a two-stage approach which is comprised of a first stage that generates a set of region proposals indicating where target objects might be located, and the second stage classifies each proposed region as an object class or as background. Design of FPN (feature pyramid network) with fast R-CNN (recursive convolution neural network) is a typical example of this approach. The other approach is one stage detector which performs object localization and classification together. This approach has fast detecting capability but compromising with an accuracy of recall ability. Most popular one stage detector SSD, YOLO v2, YOLO v3, and retina Net are well recognized for this method. The result shows that the YOLO algorithm has the highest precision-recall rates for both partially occluded body parts and full visible body parts demonstrating robustness for occlusion (Identification et al. 2019)

CHAPTER 3
AUTONOMOUS USAR WITH ROS

3.1 Introduction

This chapter discusses how the research is done by using certain methods. first, a brief explanation about ROS is given. After that, on the subsection of the chapter, we explain how to create a robot model description, sensors, mechanisms and software tools used for this project. Finally, we extensively analyze the algorithm used to develop an autonomous system in the ROS platform.

3.2 Principle of Robot Operating System

Before diving into analyzing the implementation of ROS for our system here is a brief explanation about how ROS works in general. As shown in figure 3.1 ROS Master manages the inter-process interaction between nodes (processes). Every node register at startup with the master. ROS master Enable ROS nodes to find each other, think of it as a ROS index administration, kind of DNS Provides naming and enrollment services for nodes, topics, services, and so forth. ROS Nodes organized in package ROS nodes is executable c++ or python program which can be individually compiled, executed and managed for a specific competition. ROS messages data structure defining the type of a topic Comprised of a nested structure of integers, floats, Booleans, strings, etc. and arrays of objects Defined in *.msg files. Strictly-typed data structures for the communication between nodes. float64 x, float64 y, float64 z, Vector3 linear, Vector3 angular can be some example of Ros messages. ROS topic is a name for a flood of messages, nodes communicate over topics, nodes can publisher subscribe to a topic typically, 1 publisher and n subscribers. ROS Services using one to one request-response as service /client mode Synchronous the transactions between nodes / RPC (remote procedure call) the two main roles of Ros service are performing computation and triggering functionality/behavior. As map_server/static_map recovers the present grid map utilized by the robot for navigation. Parameter Server operating inside the ROS master, it is a shared, multi-variate data set that is steamed via network APIs. It very well may be Best

utilized for static, non-binary information, such as configuration parameters. ROS packages program in ROS is sorted out using packages. A package contains at least one node and gives a ROS interface, most of ROS packages are facilitated in GitHub.

The launch is a tool for launching multiple nodes (as well as setting parameters) Are written in XML as *. launch files If not yet running, launch automatically starts a rescore.

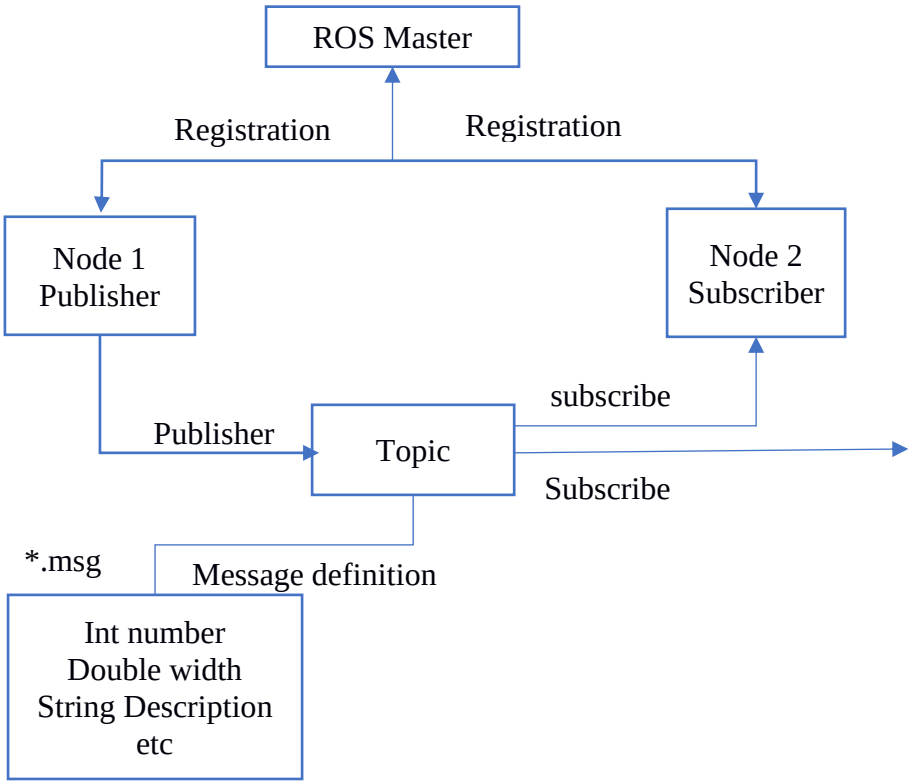

Figure 3.1: Robot operating system component

3.3 ROS Navigation

The objective of the navigation stack is to move a robot starting with one position then onto the next position securely (without smashing or getting lost). It learns from the odometry and sensors, and an objective posture and yields safe speed directions that are sent to the robot. The navigation stack expects that the robot is arranged in a specific way so as to run. The diagram underneath demonstrates a review of this design. The white parts are required segments that are as of now actualized, the gray segments are discretionary segments that are as of now executed, and the blue segments must be made for every robot stage. The pre-essentials of the route stack, alongside directions on the best way to satisfy every prerequisite, are given in the areas beneath.

Table 3.1: Ros packages utilized in this research

Packages	**Application**
TF	Maintaining the relationship between multiple coordinates frames overtime
HECTOR MAPPING	Provides a laser _based slam (simultaneous _localization and mapping) using a grid map
AMCL	Probabilistic localization system for a robot moving in 2D
MOBE_BASE	Implement the action of moving to the destination location
RVIZ	A 2d multi-robotic simulator
GAZEBO	A 3d multi-robotic simulator

The navigation stack isn't a piece of the standard ROS kinetic installation. To use navigation stack we should install it first by typing be command below.

$ sudo apt-get install ros-melodic-navigation

3.3.1 Requirement for using navigation stack

To take advantage of navigation stack of ROS, one must ensure the robot is utilizing ROS and the platform is publishing data about the transformation b/n coordinate frame utilizing TF (transformation frame of Ros). The stack utilizes data acquired from sensors to evade obstacles on the surrounding, it expects data obtained are distributing in according to messages declaration over ROS. We use Hokuyo Laser for our purpose as a range sensor. Also, for estimation of the movement of the robot ROS navigation acquires that odometry data be communicated utilizing TF and the ROS messages. In spite of the fact that navigation stack doesn't require odometry and map to work, we make a guide by utilizing hector slam and spare it on map server so as to reuse for our analysis.

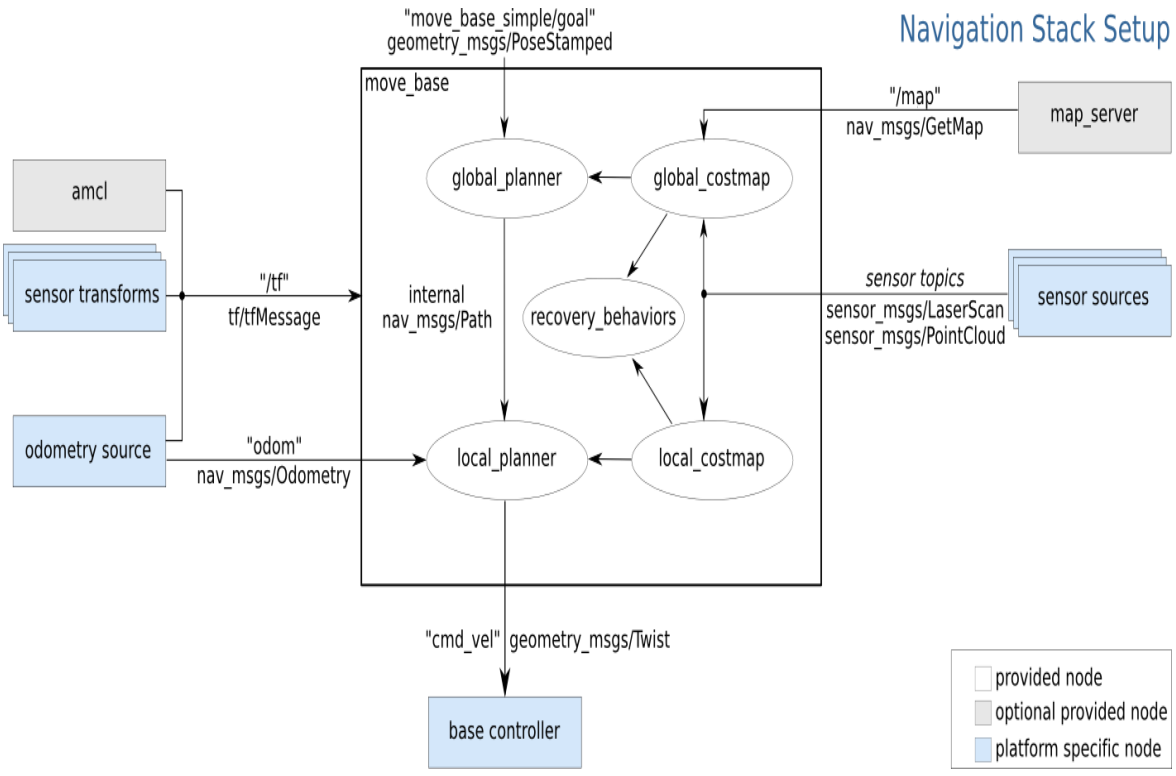

Figure 3.2: The overall architecture of Ros navigation stack(Gill ,2014.)

velocity command in Ros navigation sent by using geometric and twist messages transformed from the base frame of the platform communicated though "cmd_vel" topic. This means that the have to be anode that provides odometry and other node subscribed to it that takes angular and leaner velocity messages and change them to motor commanded to give to the mobile base.

3.3.2 Coast Map

Are a set of values (matrix) that show an area that is convenient for the vehicle inside a grid map. It interprets and uses the occupied and unoccupied cell of the surrounding and the provided robot inflation radius for effective navigation. The cost map is utilized to keep away from obstacles as per the estimation of the parameter given to it. There are two kinds of cost maps in ROS which are Managed by the costmap_2d package Global cost map collection occupied and unoccupied cells for creating a global plane of the robot and Local cost map utilized for the local planer. The global planner just uses the present guide and the footprint of the robot to design away from indicating A to B. It couldn't care less about how the robot moves so as to achieve this direction. The local planner utilizes a kinematic model of the robot to locate the robust arrangement of directions that achieve globally create a path. skid steer robot, for instance, the planner 'recreates' each linear and rotational speed suitable for performing and picking the optimal arrangement of speeds that achieve the objective.

Table 3.2: Necessary package for utilized navigation stack

Packages	**Application**
Map server	map database of ROS
HECTOR MAPPING	Offer SLAM using scan matching in occupancy grid
AMCL	Offers the localization system for the robot platform

Table 3.2 continued..

MOBE_BASE	Implement the action of moving to the destination location
Global_planner	Implement fast global planer for navigation
Local_planner	Implement trajectory rollout and TEB_local_planner for local robot navigation

3.4 Sensors

Basically, our robot contains two sensors hocuyo lidar sensor and camera sensor. Hokuyo scanning LiDAR is a range sensor which can create Realtime information on the surrounding for the purpose of detection, navigation, and measurement. It is responsible for providing continuous time-stamped digital mapping coordinates which input for our SLAM system. Most modern lidars applied for indoor and outdoor application range from 20mm-300mm, 240°- 270° scanning window and their update rate is very high. Their high update rate has significant importance when used in hector SLAM. Camera sensor provides a sequence of RGB image for the victim detection system. For simulation purpose, the gazebo provides a plugin for hokuyo lidar and camera sensor included in our robot description.

3.5 Robot Kinematic Model

The kinematics model of robot differs according to the steering mechanism which then determines the mechanical construction of the robot. In ROS the are several robot kinematic models we can operate on in order to implement our algorism the well-known are differential drive, skid and Ackerman(carlike) steering mechanisms additionally it is possible to create our own kinematic model. To implement simulation autonomous system in ROS and gazebo there is gazebo plugin which code that implements skid steering include the plugin to our robot description code (URDF) this true for lidar and camera plugin also. The steering control takes Front and rear-wheel joint static transform and applies appropriate force to move, turn the robot when command velocity id is given to it. Odometry calculation is also obtained and to obtain the robot need to go form the starting position. The static frame

transformation of the robot is taken by a robot state publisher. A brief introduction on how to create a robot model (robot description) is given in the next section.

3.6 Unified Robot Description Format (URDF)

Unified robot description format specifies varies information contained in tag depending on the robot model. In general sensor/proposal depicts a sensor, for example, a camera, beam sensor, and so on. Link contains the Kino dynamic characteristics of a robot platform. Transmissions interface mechanical interconnection of joints and corresponding actuators. Joint Describes the Kino dynamic character of the joint element. model _state depicts the condition of robot structure at a specific moment, further gazebo depicts simulation characters such as damping and friction properties. The model depicts the kinematic and dynamic properties of a robot structure. Can be incorporated as the plan prerequisite need. I talked about some of the critical traits I utilized for my purpose on the following points.

3.6.1 Link

The link element expresses a rigid body with inertia, visual and collision tags. There is a specific tag that defines a link in URDF file. I will go through some tag and show what they represent name give the nomenclature for the link. Other tags represent the visual, collusion, inertia, geometry and texture property of the link element. The initial tag is used specifically for gazebo simulator in order for the simulator to understand the inertia of the link defined. Inertia can be represented by a 3x3 rotational matrix in the inertial coordinate. Due to the symmetric property of inertia matrix, we can specify using only the following 6-attribute’s ixx, ixy, ixz, iyy, iyz, izz .visial tag can be written in order to show the link geometry of the part on graphical user Interface.

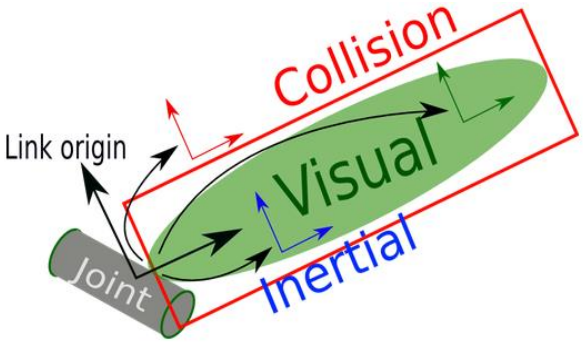

Figure 3.3: Diagram showing the relationship between frames in the link component(FabioCapasso ,2018)

Collusion tag which can be written similar to visual tag it represents the region at which the link can't be touched by other environmental or other links of robots. All the inertia, visual and collision tag's name, <origin> which contain attributes XYZ specifying offset from x,y, z, rpy. <geometry> tag representing the structure of the corresponding object. It can be a box, cylinder, sphere or a mesh. Mesh specified by corresponding file path with a specific file name, the mash can be created cad software like solid work, MeshLab, blander or other recommended format is Collada .dea file because of color flexibility ability, although STL file is supported.

3.6.2 Joints

The joint element portrays the kinematics and dynamics of the joint and furthermore indicates the safety limits of the joint.

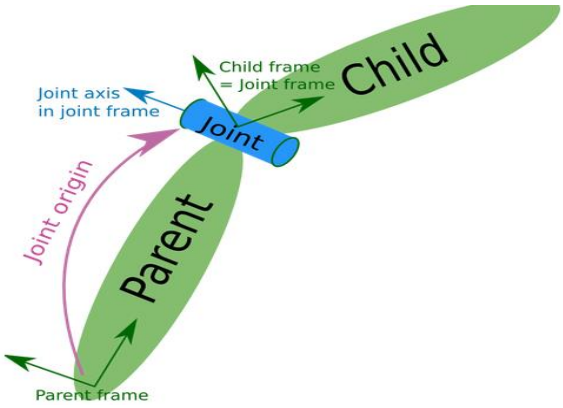

Figure 3.4: show the relationship between frames in a joint component(RicardoAngeli 2018)

The joint component has dual characters <**name**> (required) expressing a novel numeniculture of the joint. <**Type**> being necessary tag it expresses the kind of joint, which can be one of the following: revolute, Continuous and Prismatic, the first one rotating in some axis at the fixed angle specified, the subsequent one rotates continuously around an axis without limit the last one slide on some specific axes. Floating joint permits movement for each of the 6 axes on the coordinate frame. Finally, Planar this connection enable movement in a plane opposite to the hub. Joint element is a very important part of URDF as it defines the relationship b/n links. This tag contains varies elements inside, <origin> set at origin of the child link and define the transformation b/n child and parent link. We can see the figure below to understand about parent and child link and relationship between frames. XYZ, rpy attributes must be specified to set linear and angular offsets. <parent> <child> tags by their link attribute define which is parent and child link for the robot tree structure respectively. <axis> (discretionary: defaults to (1,0,0)) place inside joint tag it specifies the

rotation link with respect to another link on the robot. Additionally, <dynamics> <mimic>, <calibration> and <safety_controller> Tay's can be used to took advantage of different attribute they offer.

3.7 TF Transformation System:

Tool for monitoring coordinate frames after some time. Facilitate the connection between coordinate frames in a tree structure supported in time, letting the users transform points, vectors, and so on between coordinate frames. At wanted time Implemented as publisher/subscriber model on the topics /tf and /tf_static.TF listeners utilize a cushion to tune in to all broadcasted transforms Query for explicit changes from the change tree.

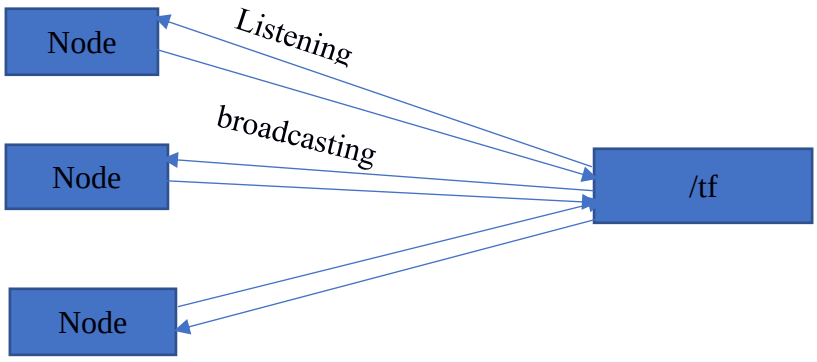

Figure 3.5: Representing TF node architecture

3.8 Simultaneous Localization And Mapping

In Probabilistic formulation of slam, a robot is presumed to maneuver in an unknown environment, over the trajectory expressed by the sequence of random variables

$$X1{:}\,t = \{x_1, \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots, x_t\} \quad (3.1)$$

while moving it acquires sequences of odometry measurement

$$U1{:}\,t = \{u_1, \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots, u_t\} \quad (3.2)$$

perceptions of environments

$$Z1{:}\,t = \{z_1, \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots, z_t\} \quad (3.3)$$

A map can be specified as a set of spatially located landmarks, by dense representations like occupancy grid's, surface map, or by row sensor measurement. Solving slam problem requires computing in high dimensional state space Considering static world and Markov assumption. A convenient way to explain this structure is with the dynamic Bayesian network (DBN) (Grisetti et al. 2010). Bayesian network is a graphical structure that expresses the stochastic process as directed graph connectivity of DBN follows a repetitive pattern expressed by the state transition model and by the observation model.

In Figure 3.5, blue nodes represent the observed variables (this means z1: environment T and u1: T), white nodes represent required variables (x1: T) and m describing the robot's trajectory and the grid map of the environment. The structure of the DBN obeys a recurrent pattern expressed by the state transition model and by the observation model. The transition model p(xt | xt−1, ut), tries to find the position of robot xt given the past position of robot xt−1 and odometry measurement ut. The observation model p(z t | xt, mt) models also try to obtain observation zt given the present robot position xt from within observation map m. To address slam problem standing on a probabilistic point of view we can see two equally important solutions which are online slam and full slam explained below.

Online slam: This problem structure addresses the estimation of momentary pose (position and orientation) along with the map, this removes old measurements and controls the present data's as they become available equation 3.4 (THRUN 2000)

$$p(x_t, m \vee z_{1:t}, u_{1:t}) = \int\int \ldots \int p(x_{1:t}, m \vee z_{1:t}, u_{1:t}) \mathrm{dx}_1 \mathrm{dx}_2 \ldots \mathrm{dx}_{t-1} \quad (3.4)$$

Full slam: analyzing the full slam problem includes estimating the posterior probability of the robot's trajectory x1:t and the map of the environment all the measurements plus an initial position provided equation 3.5.

$$p(x1{:}\,t, m | z1{:}\,t, u1{:}\,t, x_0). \quad (3.5)$$

3.8.1 Overview of existing SLAM solutions

Generally, there are about four basic techniques of generating consistent maps in slam which are EFK slam, fast slam, scan matching, and graph slam. Slam solution can be achieved by different realization the oldest and comparatively more exact estimation algorithm is Kalman filter (THRUN, 2000). Kalman filter method uses Gaussian distribution with the motion model linear or some linearization implementation made before the execution of estimation. Extended Kalman filter (EKF) is one of the most popular algorithms which categorized in this category as the name indicates EKF implements an extension of linearization step before Gaussian estimation of the pos (translation and rotation) analyzed. Considering the computational cost of EKF researchers focus to its twine extended information filter (EIF) which represent the Gaussian posterior by information vector b information matrix H instead of using mean μ and covariance matrix Σ.

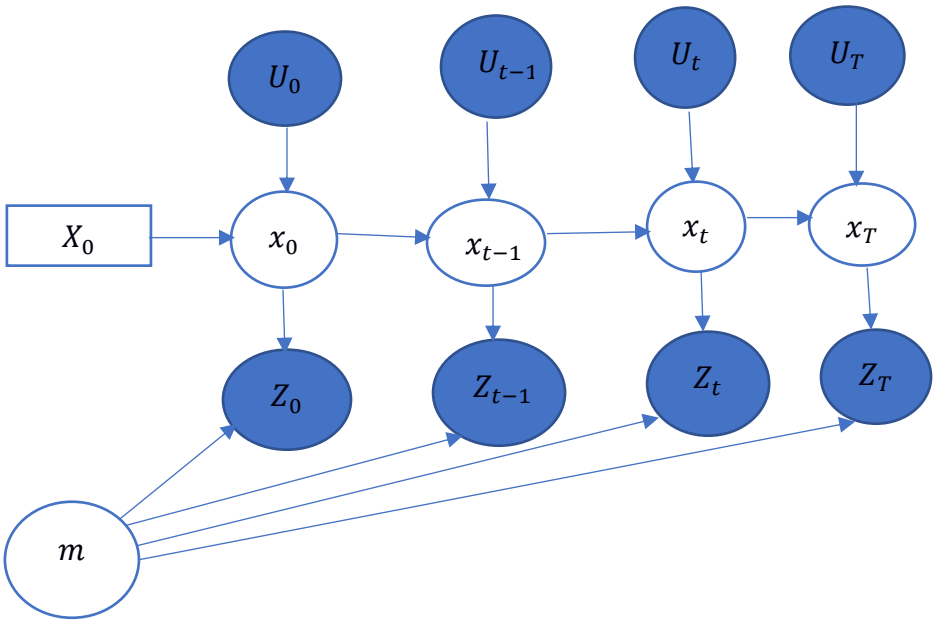

Figure 3.6: Dynamic Bayesian Network (DBN) of the simultaneous localization and mapping process (Grisetti et al. 2010)

The two parameterizations are similar can be interrelated by $H = \Sigma^{-1}$ and $b = \mu^{T}H$ the sparse pattern is drastically different (Chen 2013).EIF algorithm then extended to SEIF (which is sparse extended information filter) to optimize the storage system to linear with the number of landmarks and execute all updates in constant time (Chen, 2013).

As I mention above Kalman filter suffer when it comes to the nonlinear (arbitrary) distribution of motion model, we can use linearization method if our model is not too nonlinear but for highly arbitrary models a most promising method for estimation is particle filter (Doucett, Freitast, and Russent, 2000). Particle filters use samples of arbitrary distribution these samples are assigned with a distinct amount of weight, the higher the weight the higher the probability of the area this means we can find the posterior from the sample representation. To achieve this particle filter algorithm, implement three main steps which are sampling the particle using the proposal distribution, compute the importance of weight, resampling for better estimation. As the particles are generated and resumed, they automatically concentrate on the more probable region of the distribution. Graph-based slam build by node s which represent robot pose (position and orientations) or landmarks, and edges reflecting the sensor measurements that define the relationship between those poses or landmarks this construction stage is called front end (Grisetti et al. 2010). This stage only responsible for the data association and bounce the search for correspondence. In graph slam, we need to find the structure of anode that highly consistent with the measured data this is achieved in the optimization stage. This stage involves solving a large error minimization implementing mostly list square method, or related algorithms.

The other widely used slam algorithm is using scan matching approach which is implements scan registration with or without odometry (Stefan Kohlbrecher, Oskar v, Johannes M, 2011). Scan matching organizes or aligns scans as they become available with each other or with existing map. Due to modern laser scanners low distance measurements noise and high scan rate this method yields Avery accurate in creating maps. For most robots, this method performs much better than that of odometry data if it is available at all. Hector slam is the most known algorithm for implementing this method which demonstrated in UGV,

underwater, and quadcopters. In our case also we take advantage of this method as our environment for search and rescue mission is full of rubs and uncertainties.

3.8.2 Hector SLAM

Hector slam is one of well-known slam algorithm which uses scan matching for generating a map(Stefan Kohlbrecher, Oskar v, Johannes M 2011). This approach relies on optimization the alignment of beam endpoints with map registered before.

In HECTOR, we seek to obtain rigid transformation ξ = (px, py, ψ) T that minimizes

$$\boldsymbol{\xi}^* = \underset{\boldsymbol{\xi}}{\operatorname{argmin}} \sum_{i=1}^{n} \left[1 - M(\mathbf{S}_i(\boldsymbol{\xi}))\right]^2 \tag{3.6}$$

We compute the bast alignment laser scan with the map equation 3.7. where Si (ξ) is the world coordinate of scan endpoints. M (Si (ξ)) return the map value at the coordinate given by …S i (ξ). In other words, we compute the best alignment of the laser scan with the map.

$$\sum_{i=1}^{n} \left[1 - M(\mathbf{S}_i(\boldsymbol{\xi} + \Delta\boldsymbol{\xi}))\right]^2 \rightarrow 0\,. \tag{3.7}$$

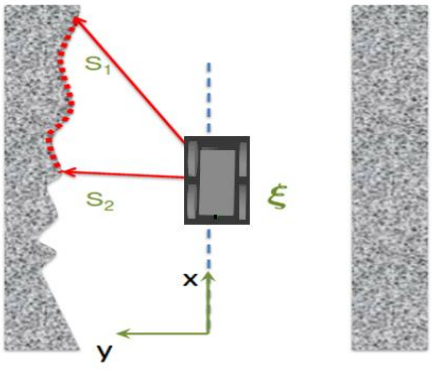

Figure 3.7: Demonstrates hector slam scan matching problem

Given that starting pos ξ. of, we can evaluate Δξ which minimize the error measurement according to by using first-order Tyler expansion and partial derivative with Δξ to zero and solving we can obtain Δξ according to equation 3.8. which is the change in pose (Stefan Kohlbrecher, Oskar v, Johannes M 2011).

$$\Delta\boldsymbol{\xi} = \mathbf{H}^{-1} \sum_{i=1}^{n} \left[\nabla M(\mathbf{S}_i(\boldsymbol{\xi})) \frac{\partial \mathbf{S}_i(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}} \right]^T [1 - M(\mathbf{S}_i(\boldsymbol{\xi}))] \tag{3.8}$$

3.8.3. Adaptive monte carlo localization (AMCL)

Two key problems in mobile robot localization are global position prediction and local position tracing. We express global position estimation as the capability of the robot to find its position with a previously known map and without any other information other than the robot could be somewhere on the map. If the robot doesn't have prior information about the map, the map should be created online at the time of exploration. Once we obtain the position of the robot on the map, local tracing localizes the robot throughout the movement of the robot over time. Both these abilities are important to empower a robot to execute helpful errands, for example, office conveyance, transportation and search and rescue mission (Sebastian Thrun et al. 1999). AMCL is a probabilistic localization system for a vehicle moving in 2D space. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach, which uses a particle filter to trace the pose of a vehicle within a known map (SteveMacenski n.d.). AMCL takes an input of a laser-based grid map, laser scans, and frame transform messages, and outputs pose prediction of the robot. On startup, AMCL initializes its particle filter by using the parameters given by launch file. Remember that, due to defaults setup, if no parameters are given, the initial filter state will be a moderately sized particle cloud centered about (0,0,0).

3.9 Motion Planning

Path planning research for autonomous navigation is started since in the 1970', in 1971 Lazano, Parez, and Wesely introduces the configuration space which then starts the in-depth research in modern motion planning (LOZANO-PEREZ, 1983). now a day's automation of different machineries being studied, robot automation being the frontier for this since robot's

role in people's day to day life is being significantly growing. In this area most researches are focused on obtaining optimal solution such as path planning (figuring out the path the vehicle must follow to arrive at specific goal and motion control (analyzes the control input that must be given to the robot to keep the collision-free path. Specifically considering motion control, varies algorithms has been used in keeping the path. Here we define the problem motion planning and try to see TAB_local_algorithim intuitions.

3.9.1 Obstacles And The Configuration Space

The configuration space, or C-space, of the robot framework, is the space of every conceivable setup of the framework (LaValle 2011). The number of degrees of freedom of a robot configuration is the dimension of the configuration space or the minimum number of parameters needed to specify the configuration. 3-parameter representation: q = (x, y, θ). In 3D, q would be of the form (x, y, z, α, β, γ) . Standing on our understanding of configurations and of configuration spaces, we can characterize the path-planning problem to be determining a continuous mapping, c: [0, 1] →Q, to such an extent that no configuration in the way causes a crash between the robot and an obstacle (LaValle, 2011). With $W = R^m$ being the work space, $O \epsilon W$ the set of obstacles, $A(q)$ the robot in configuration $q\epsilon$

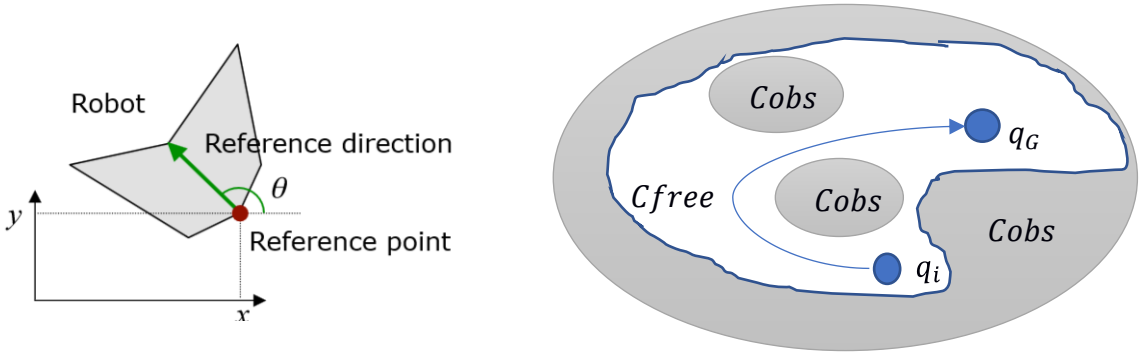

Figure 3.8: Three degree of freedom robot in c-space on the left, the right figure shows the question of path planning in c space connects q_i to q_g while remaining in Cfree

$$\boldsymbol{C_{free}}=\{\boldsymbol{q}\epsilon\boldsymbol{C} \mid \boldsymbol{A}(\boldsymbol{q})\cap\boldsymbol{O}=\emptyset\} \quad (3.9)$$

$$c_{obs} = c/c_{free} \quad (3.10)$$

The motion planning amounts to finding a continuous path. This setting provided; we can analyze motion planning with the robot assumed as a point in C-space.

$$\tau\,[0,1] \rightarrow\ c_{free} \quad (3.11)$$

With $\tau\,(0) = q_i$, $\tau\,(1) = q_g$

3.9.3 Local planer solutions in ROS

It is mandatory to compare algorithms to achieve a higher standard result. Ebad local planer is a force base path deformation and path following controller. Use in plan short path for the local solution without considering Kino dynamic constraints. Tab local planner depends on optimization resp with a predictive control strategy. using multiple local solutions perform parallel optimization for the time-optimal result or ref, path fidelity considering Kino dynamic constraints. DWA uses sample-based trajectory generation with predictive control to achieve the globally planned goal. One of the downsides, the algorithm takes samples that exhibit constant curvature. Showing time suboptimal result obtained from multiple local solutions by considering Kino dynamic constraints. There difference lie in the computational burden of the algorithms, TAB local planner possesses high Burdon compared to that the other two. DWA planner being the list as constraint competition is less. All this algorithm can be implemented for Omnidirectional and differential drive robots, but Tab local planner can be used for car-like robots in addition.

3.9.4 TEb_local Planer

Online planning is favored over offline approaches because of its quick reaction to changes in a dynamic environment or disrupts the robot movement at runtime. Besides producing an impact freeway towards the objective trajectory optimization considers auxiliary objective

such as control error, trajectory length, control effort, travel time and clearance from obstacles. Practically, because of limited computational resources most of the time online optimization depends upon local optimization strategies for which convergence into the global optimal trajectory is not pledged. In the navigation system of mobile robots, optimal trajectories are necessary due to the existence of obstacles.

Time Elastic Bands (TEB) create a sequence of intermediate vehicle poses $p = (xi, yi, oi)T$ modifying the initial global plan. It requires the speed and acceleration limits of the robot, the safe separation of the obstacles and the geometric, and kinematic and dynamic limitations of the vehicle (Marin-Plaza et al. 2018). The original TEB planner is optimized extended in (Christoph Rösmann,, 2017) to a fully integrated online trajectory planning methodology that combines the exploration and simultaneous optimization of multiple admissible topologically individual trajectories during runtime. A discrete trajectory $\boldsymbol{b} = [s1, \Delta T1, \Delta T2, \ldots\ldots\ldots, \Delta TN - 1, Sn]^T$ is expressed by an ordered sequence of poses incremental sequence with time stamps. $s_k = [x_k, y_k, \beta_k]^T \in R^2 \times S^1$ with k=1, 2……., N represents the position and orientation of the robot and $\Delta T_k \in R > 0$ with k=1,2……..., N−1 denote the time interval related to the movement between two respective poses S_K and S_{K+1} consecutively.

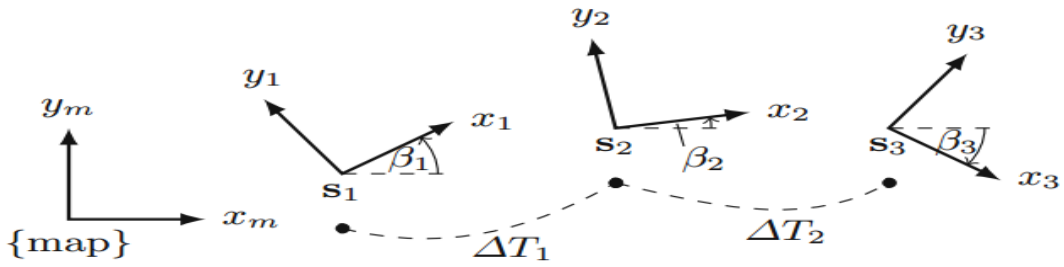

Figure 3.9: Demonstrate Discrete path with n=3 poses (Christoph Rösmann 2017)

The TEB optimization inquiry is defined in such a way that b∗ minimizes a weighted and clustered nonlinear least-squares cost function:

$$b^* = \arg \min_{b\{S_1, S_N\}} \sum_i \sigma_i f_i^2(b),\ i \in \{\mathcal{I}, \mathcal{P}\} \quad (3.12)$$

The terms fi: B→R ≥ 0 represent conflicting objectives and penalty functions. The set of indices related to objectives is denoted by $\mathcal{I}$ and the set of indices that allotted to penalty functions by Ƥ. The change among individual variables is obtained by weights σ_i. The term b\{S_1, S_N} indicates that starting pose S_1 =S_s and final pose S_N =S_g are fixed and therefore can't be optimized. on the cost function, S_1 and S_N are replaced by the present robot pose S_s and the required goal pose S_g.

3.10 YOLO V3

Deep learning has been around for long but it has massively dominated computer vision after graphic card capacity is upgraded in recent years overcoming customarily used methodologies, achieving the top score, on several tasks and their associated competitions. Deep learning is achieved by forming a hierarchy of a convolutional neural network, pooling, and softmax structure. One strand convolutional network can concurrently predict numerous bounding boxes and class probabilities for those boxes.

YOLO implements deep learning algorithm in order to accomplish real-time object detection. Unlike that of sliding window and region proposal-based methods, YOLO views the entire image once during training and test moment so it completely encodes contextual data about classes as well as their appearance information. From the image, YOLO creates a grid of 13*13 grid cells each grid cell is responsible for predicting 5 bounding boxes which totally brings 845 bounding boxes. The confidence result for the bounding box and the class estimates are integrated into a final score that gives us the probability that this bounding box is a particular type of object. YOLO V3 Further improve upon YOLO V2 by incorporating some elements used in other states of the art detection algorithms such as residual blocks and feature pyramids as shown on the network Figure 3.1

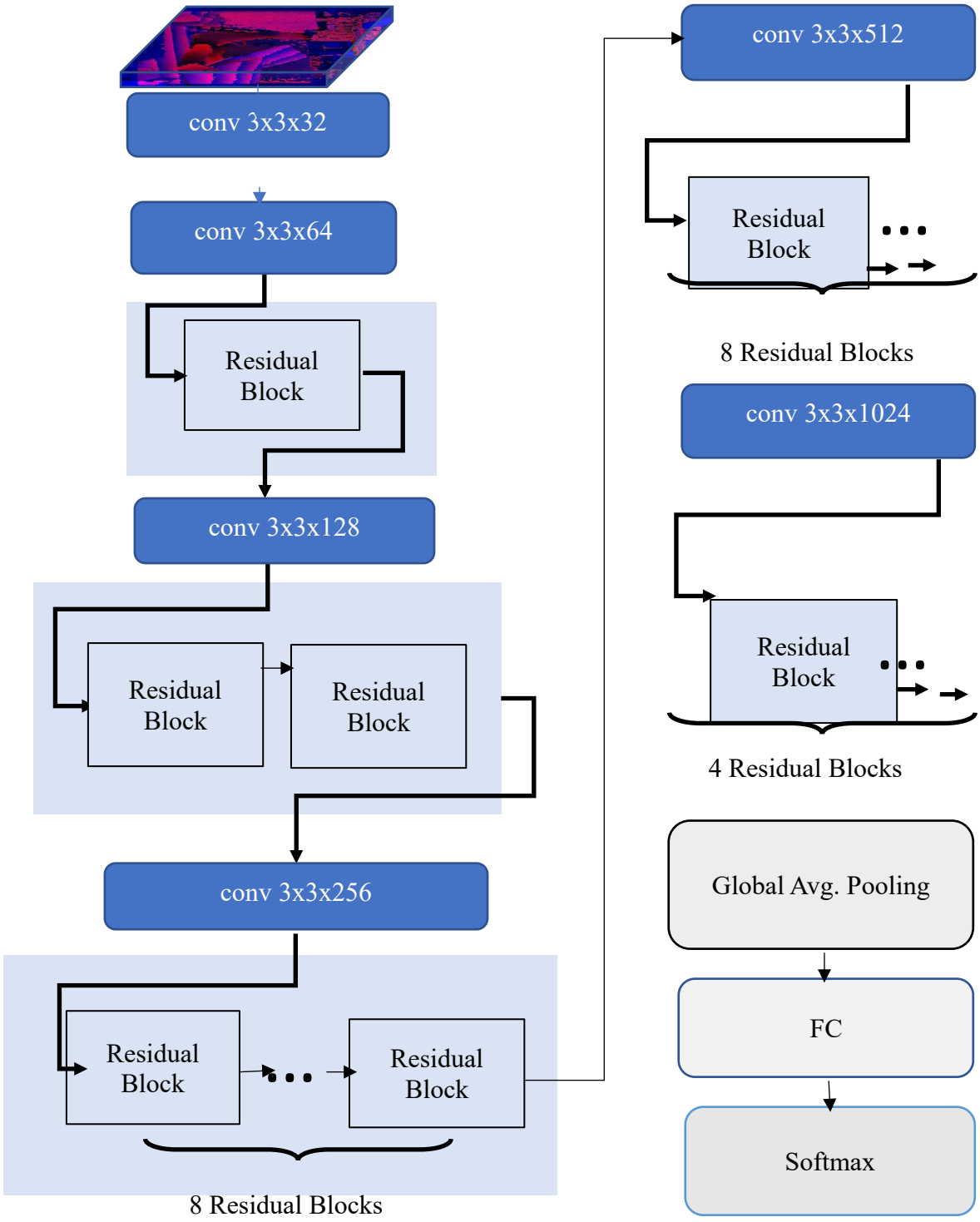

Figure 3.10: Darknet-53, the feature extraction layers used in YOLOv3, (Adopted from (Identification et al. 2019))

On the present stage, there is ROS package developed for object detection in-camera images called YOLO (V3) (Bjelonic, 2018). We can use this ROS package YOLO (V3) on GPU and CPU. The pre-trained model of the convolutional neural system is proficient to identify pre-trained classes including the informational collection from VOC and COCO, or you can likewise make a system with your very own recognition objects.

CHAPTER 4
RESULT AND DISCUSSION

4.1 Overview

For the simulation of robot, we created a robot description and gazebo environment using URDF and SDF as explained on the corresponding topic. Hector mapping, AMCL, map_sever, move_base packages of ROS are parametrized according to our need's. The parametrization is done using a launch file for SLAM and move_base packages. For local and global planer packages we use a YAML file to set the corresponding tuning values Appendix 3. Frame transformation b/n links are done by using robot state publisher and TF frame state. Skid steering control used as odometry source.

4.2 Skid-Steer Model

As expressed previously our model is a skid-steer platform. This platform is used as it provides the same movement model, Figure 4.1 but requires different control and model dynamics. This will enable the skid-steer of system parameters between model types. For simplicity, the model contains only laser scanner placed on the middle of the front base link of the robot. The model's parameters are 500 x 750 x 375mm (LxWxH), weight is 8kg, and wheel radius is 500 mm. Fig shows the final models design.

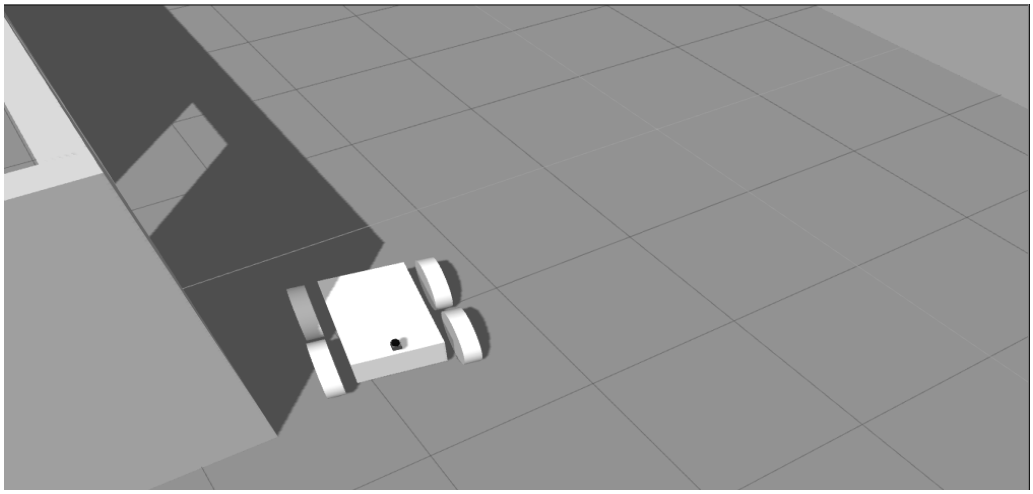

Figure 4.1: Skid steering robot model used in experimentation

4.3 Simulation Environment

The simulator is very essential tools in robotic researches for their fast and robust test of new concepts, methods, and algorithms. As we are taking advantage of ROS platform to simulate search and rescue robot environment, gazebo simulator is very suitable, as it's capable of creating complex 3d sonorous that our robot experience additional it offers data visualization and simulation of the remote environment. Gazebo uses SDF format for creating a world which is an XML file. In this SDF file, several components are included to address the root property of the environment. For the simulation of the autonomous navigation system, we created a simple cross-shaped floor with four rooms using build editor of gazebo simulator, and included fallen and standing person structure for implementing our victim detection algorithm as shown figure 4.2.

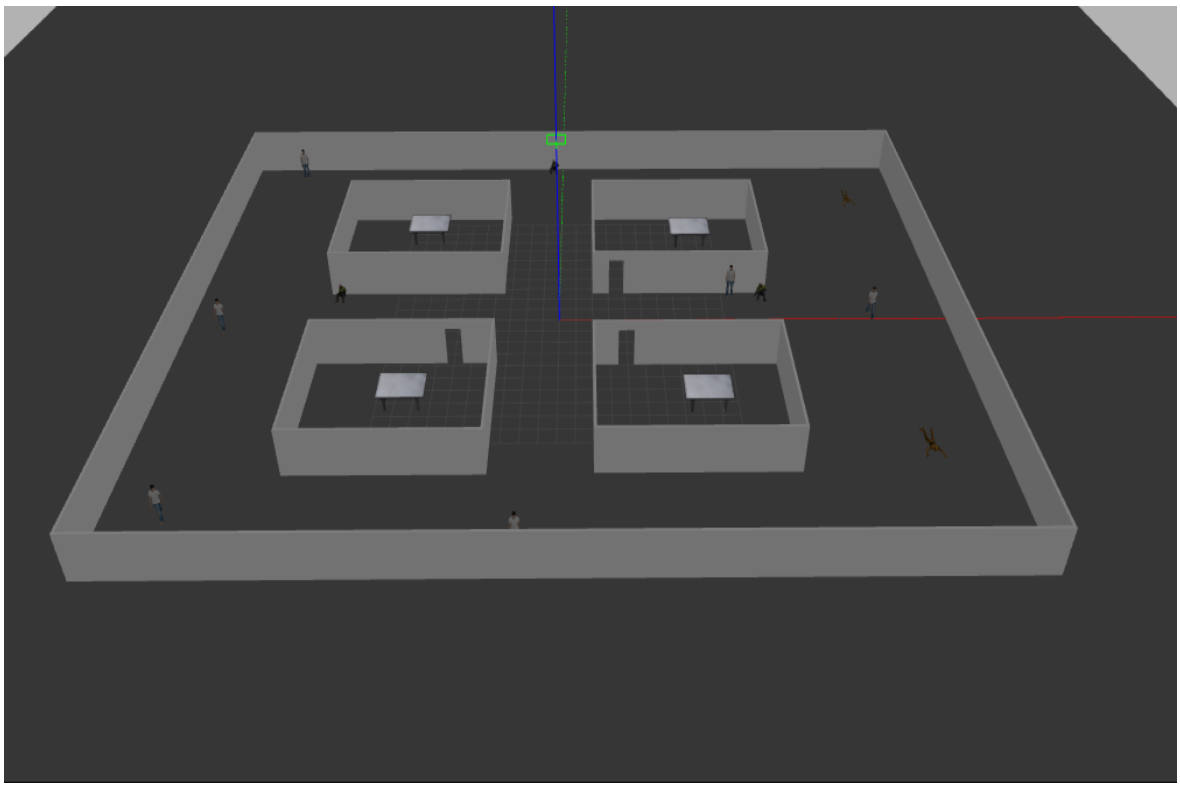

Figure 4.2: Environment design for experimentation in gazebo simulator

4.4 Packages Used

For the model, we use the AMCL package for localization, global_planner for global planning and TEB_local_planner as the primary local path control. These three provided the best results through testing detail parameter settings on appendix 3. The drive controller uses the skid_steering_controller with an update rate of 10, a torque of 10 N.m, receives /Odom topics and publishes motor commands to the /cmd_vel topic. The localization parameters and laser parameters were set from the skid-steering-drive model defined above with odom_model_type adjusted to diff. The controller frequency of the path planners move_base node is also set to 50Hz. This is to help reduce the impact of the rotation bug mentioned above. The global path planner used is global_planner. It is defined with default values. The local planner uses the Time elastic band (TEB) method, the odom_model_type set to deff in move_base launch file appendix 2, show the skid steering model is implemented by the assumption of the bicycle model, meaning that the four well of the robot model using two virtual weels placed for the simplicity of analysis. All parameters used for packages are shown in the appendix. The corresponding values are in meter, meter per second, radians per se cond and in corresponding SI unit

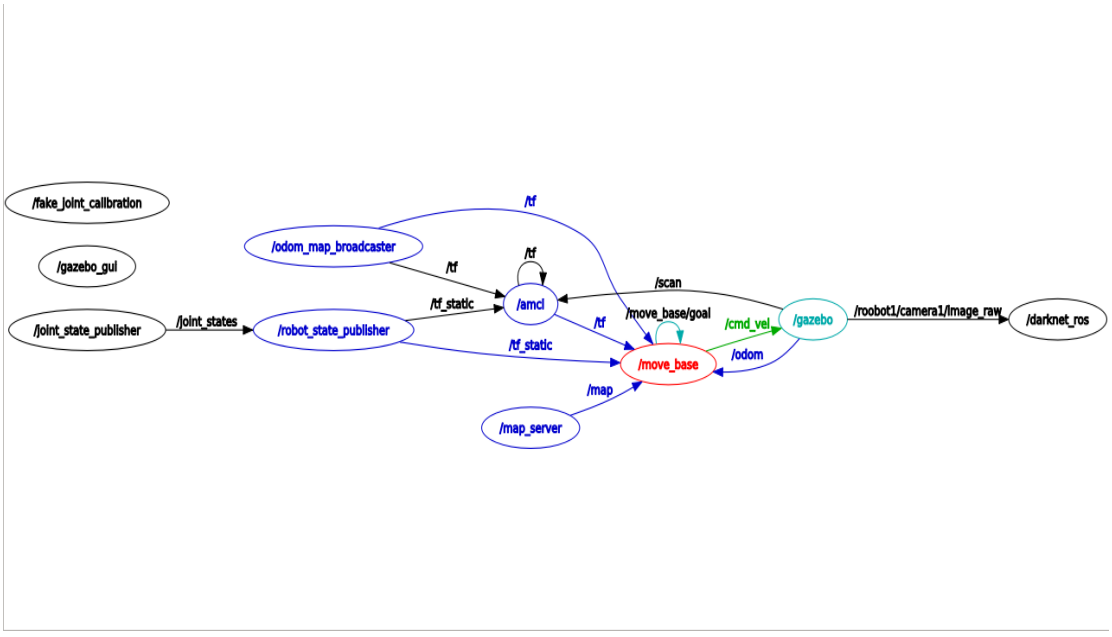

Figure 4.3: RQT graph showing the node of the control system from ROS. Representing all the node used in the system and their corresponding topics registered for communication

4.5 Internode Communication

The frame transformation graph of our simulated robot is shown in Figure 4.4. The parent of the frame on hierarchal is map frame. This frame is a virtual frame that offers orientation and position of the environment for the system. The relative position and orientation of the robot in the simulated environment are advertised from Odom frame the broadcaster is /gazebo topic. Robot state publisher is responsible for transforming frame pose between fixed robot frames which are wheels and chassis (base_link). RQT graph of the robot shows the overall system of our robot. Several nodes of the system operated by registering the corresponding topic to plan, move, to position the robot autonomously. Figure 4.3 shows each node of the system with their corresponding subscribed and publishing topics.

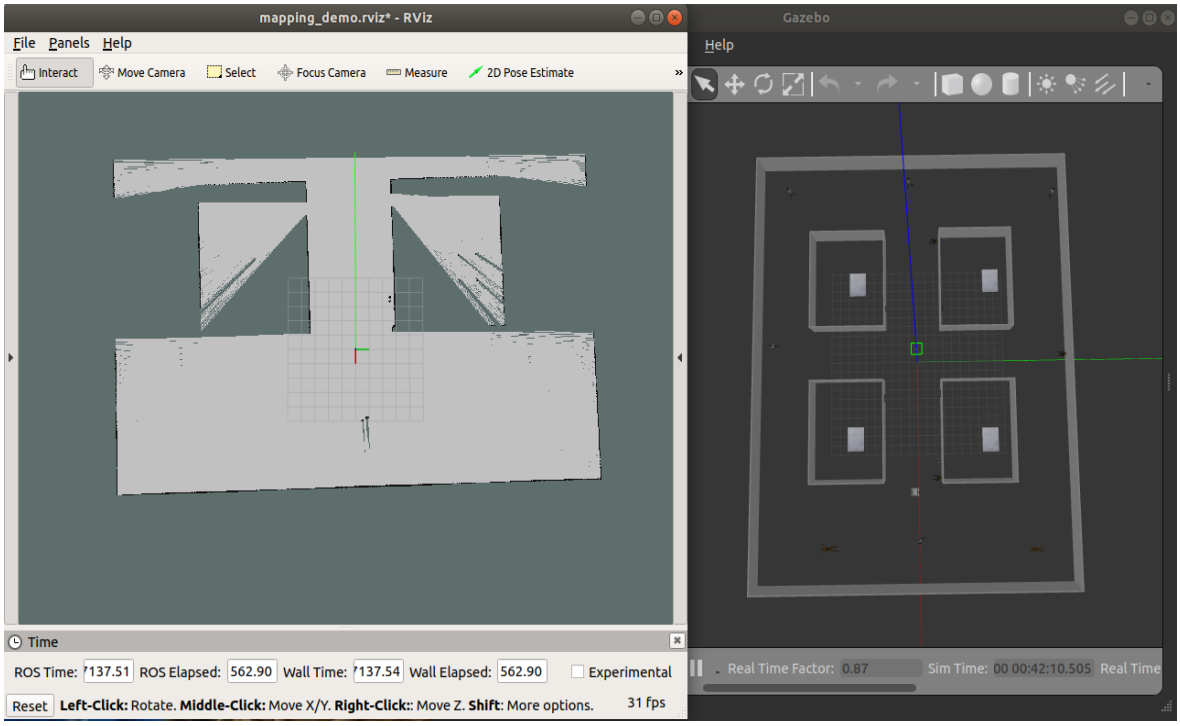

Figure 4.3: implementing hector mapping on the enviroment, left side showing map of half of the enviroment ,right side robot moving in the environment

Figure 4.4: Robot frame transformation graph, show the hierarchy of frames with there corresponding publishers.

4.6 Simultaneous Localization and Mapping

The mapping performed using hector SLAM on the simulating environment is shown in Figure 4.5. Hector slam implements hector mapping node for building a map of a world and concurrently predicts the pose of the environment at scan frame rate. The linear and angular speed of the robot is 0.02 and 0.16 (meter/second rad/second) were set when mapping respectively. AMCL package is used for localization of the robot. The result for the mapping and localization is pretty much consistent to the environment and robot position as shown below in figure 4.5. The environment as represents on Gazebo Figure 4.2: the two figures below show the occupancy grid map created using hector slam.

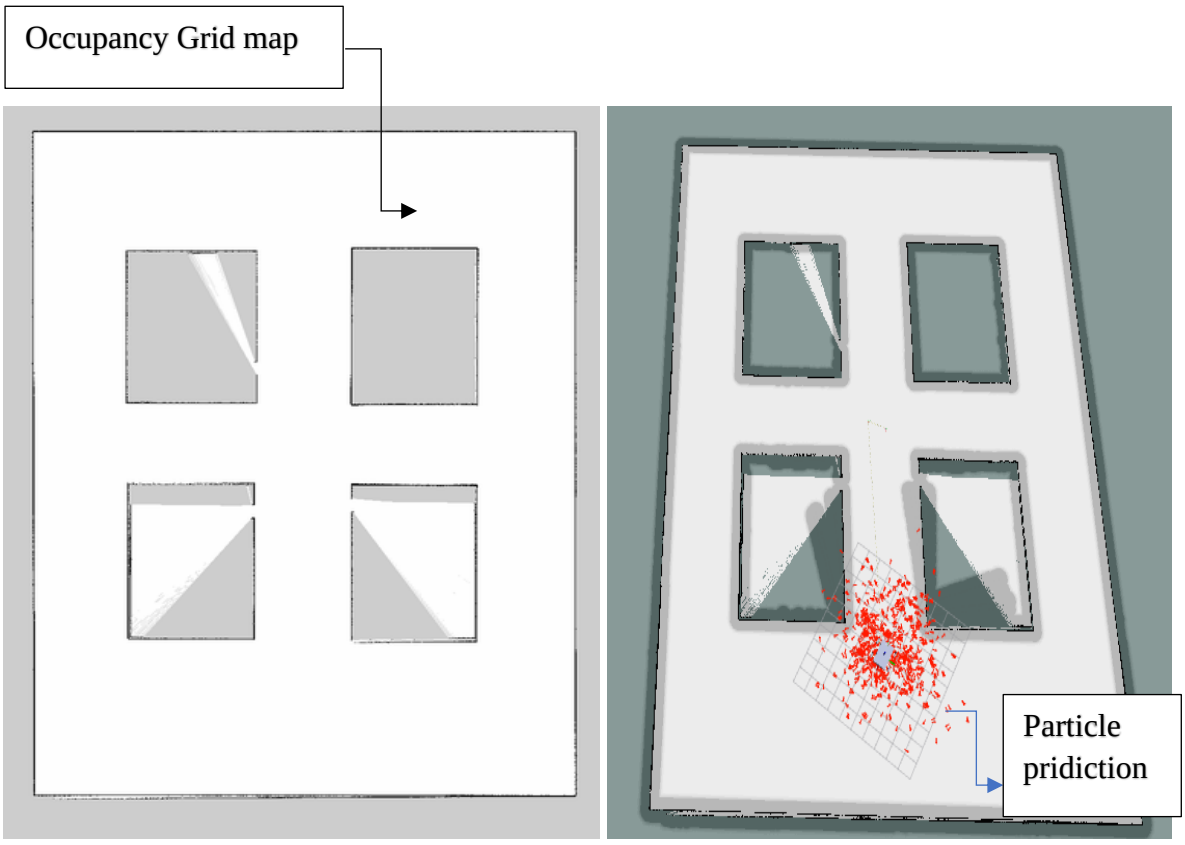

Figure 4.5: Occupancy grid map of the gazebo environment on the left side of the figure, global cost map, robot localization, and particle prediction shown on the right.

4.7 Motion Planner

The motion planner uses default ROS global planer and tab planer as a local planer. Both local cost map and glocal cost map are shown in figure 4.6. The path created by global plan considerably accurate but sometimes can be very near to obstacle. The record of velocity necessary for the robot to achieve a goal is shown on the graphical on figure 4.8,4.9,4.10 on some momentary time interval.

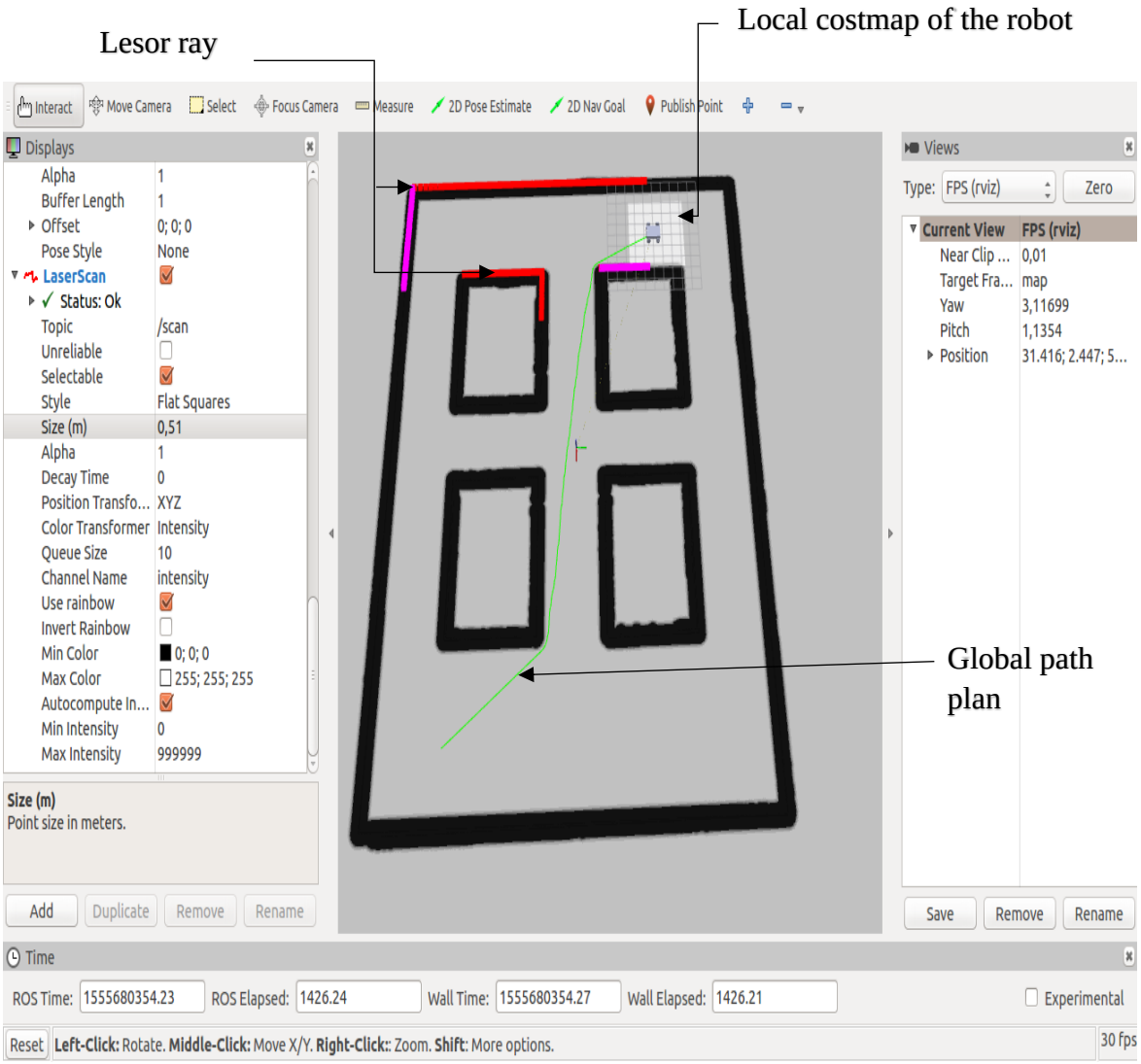

Figure 4.6: Global and local cost map of the system, the red and pink line representing laser ray and global plan represented by the green line, local cost map show with a white box the robot moving on the top

4.8 Victim Detection

The robot is in front of two people in the gazebo environment the camera integrated with the robot record data and send to YOLO. On Run time stream of images are given to YOLO, YOLO V3 node is register to the camera topic in our case Camera/raw. As the result show the pre-trained YOLO algorithms process the data and obtain two victims in this case person classify and create a bounding box around the image shown figure 4.11. We used the default trained YOLO algorithm because creating clustered USAR environment is still not easy to achieve is gazebo platform to implement complicated victim detection.

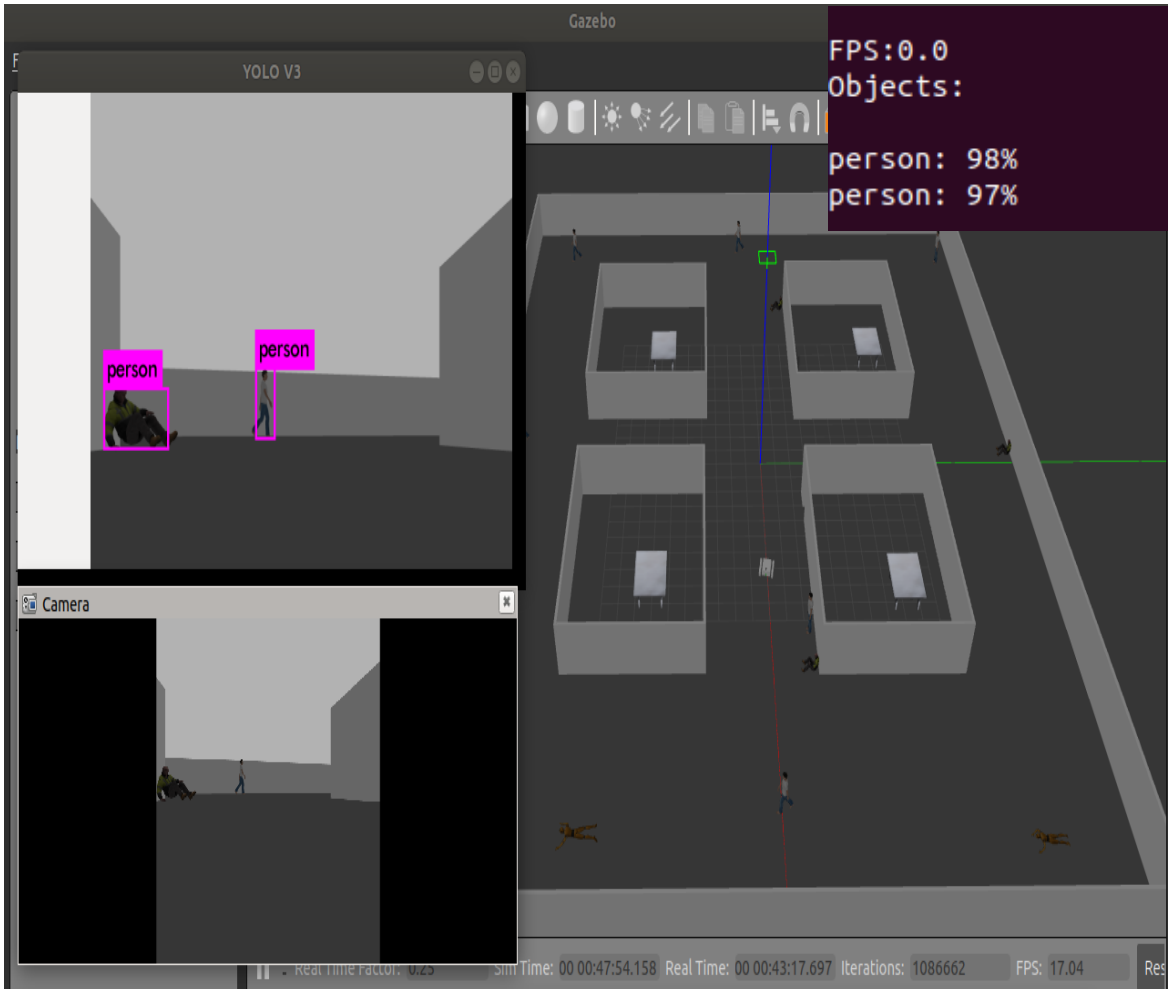

Figure 4.7: Wider right-side representing robot facing the person in gazebo environment, below to the left output of the camera sensor in RVIZ, on the top right YOLO object detection result bounding box on the person

4.9 Discussions

As we learn from the simulation at high speed the mapping built is inconsistent because of the rate at which the scan match of hector slam. This makes the mapping inconsistent with the environment. In order to fix this, the mapping is performed at a lower speed of 0.02 meter/sec and 0.16 rad/second linear and angular respectively. The simulation shows Amcl works better for short distance traveled. The orientation of the pose array of the AMCL is not consistent when the robot rotates but give accurate enough for overall performance. For the uncertainty of Amcl value alpha have been increased of default value because the skid steering gazebo introduces much more uncertainty. As we observed from the simulation the gazebo simulator update rate plays an essential role succeeding on navigation stack because the update of the odometry from gazebo must match the update rate of frame transform of robot description.

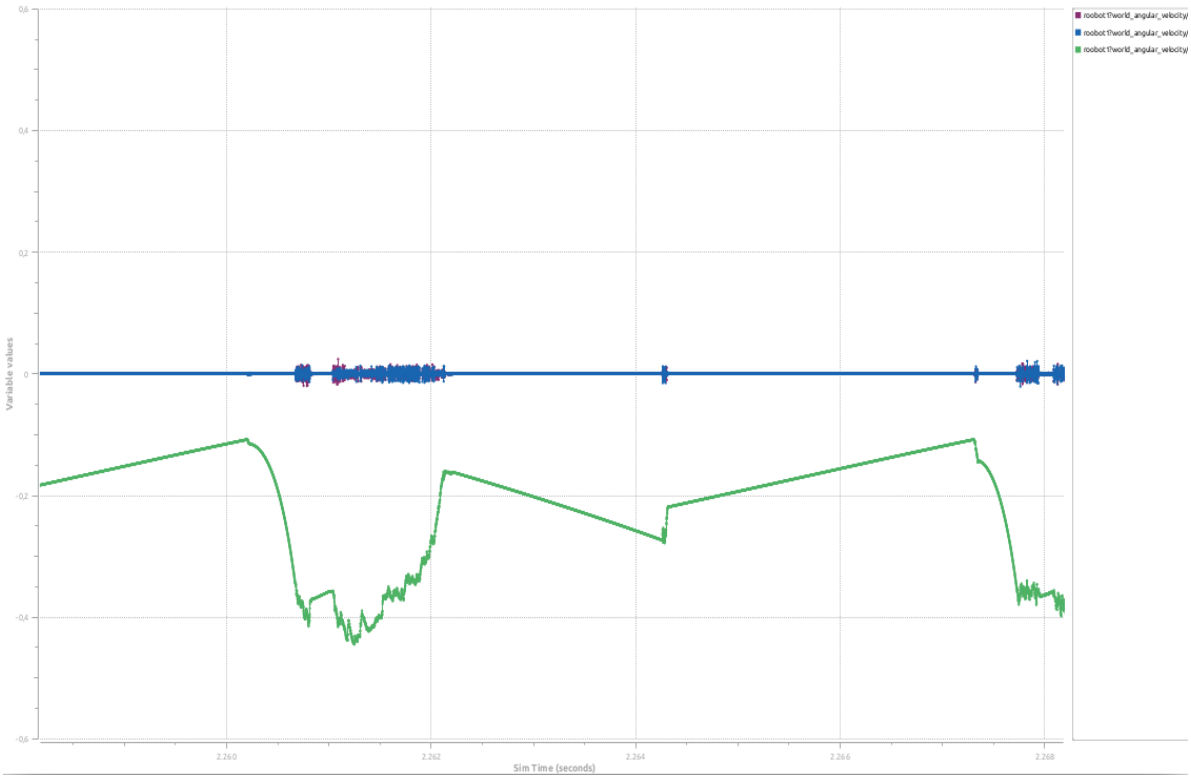

Figure 4.8: Angular velocity of the robot recorded on some specific time. Blue representing Angular velocity around the X-axis rad curve representing Angular velocity around the Y-axis green curve representing Angular velocity around the Z-axis

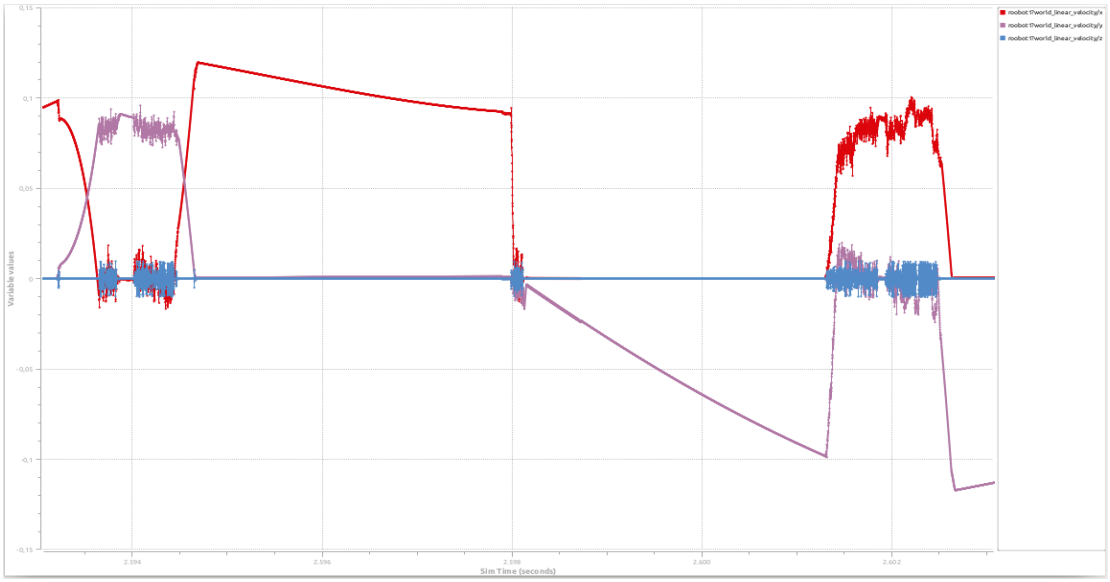

Figure 4.9: Linear velocity of robot recorded on some specific time. The blue curve represents linear velocity in x-direction pink curve representing linear velocity in y and the red curve represents linear velocity in z

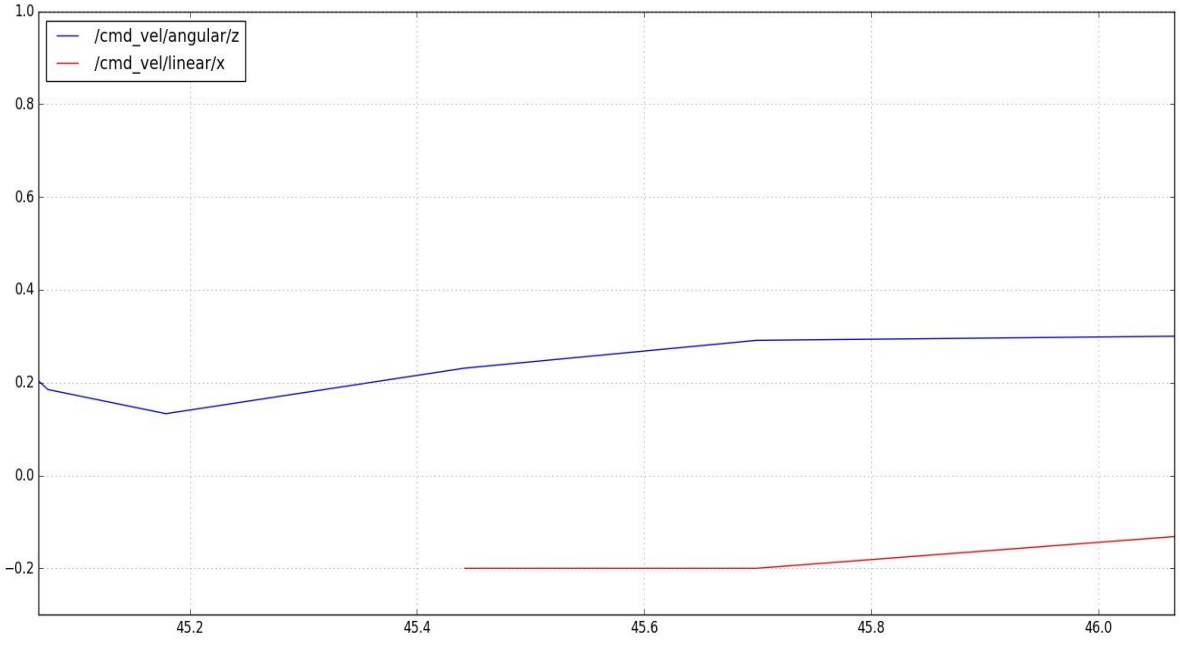

Figure 4.10: A graph showing angular and linear cmd_vel of the move_base node for the local planer in some momentary time interval

CHAPTER 5
CONCLUSION AND RECOMMENDATION

5.1 Conclusions

On this research, we demonstrated the simulation of autonomous search and rescue robotics using a robot operating system and gazebo simulator. Analyzing ROS we show that hector mapping, AMCL, Ros global planer, and TEB local in combination can give robust result comparative from other existing slam and motion planning algorithms on the platform. Victim detection is achieved successfully using YOLO V3 which is the present state-of-the-art deep learning algorithm. The test environment and robot model have been created using universal robot description formant (URDF) and solid dynamic format (SDF). The system takes range data of the surrounding using hucuyo lidar sensor interpreted it as occupancy grid map using hector slam. Using the map created from the map server move base generate obstacle-free path and velocity command to reach a goal. Our result shows that comparatively teb_local planner best performance for local planning utilizing multiple local solutions perform parallel optimization for its time-optimal result. Default Ros global planer is used for path planning and the parameters used for move base global planer and local planer is demonstrated in the appendix. The result has been demonstrated by using GUI of YOLO V3, RVIZ and gazebo platform.

5.2 Recommendations and Future Work

The analysis is done in this research shows that we can successfully achieve an autonomous search and rescue robot by using Ros platform. We recommend to realize and use this system for real autonomous research and rescue mission or computation. The method we used in this research is can be easily perceived and adapted for any kind of robot system for any objective. Therefore, we recommend a robot designer to use Ros for their robot because it is flexible, full of resources and capability. As the objective of this work is to analyzing simulating of Ros platform for achieving autonomous navigation system for search and rescue mission we are limited. On the feature work, we can work on several issues such as realizing this system in a real environment. We can analyze the dynamic obstacle which is obvious in search and environment. On this thesis we use a predefined map for our system store on map server in the future we may work on using online slam to achieve exploration capability for our system.

REFERENCES

Achmad Fathoni.“Dwa_local_planner.”*2018, june*. http://wiki.ros.org/dwa_local_planner?distro=kinetic.

Bjelonic, Marko. “Real-Time Object Detection for {ROS}.” *2018 , january*. https://github.com/leggedrobotics/darknet_ros.

Burke, Jennifer L, Robin R Murphy, Dawn R Riddle, and Thomas Fincannon. 2004. “Task Performance Metrics in Human-Robot Interaction : Taking a Systems Approach.” *Center for Robot-Assisted Search and Rescue*.

Chen, Yuncong. 2013. “Algorithms for Simultaneous Localization and Mapping.” http://cseweb.ucsd.edu/~yuc007/documents/re_report.pdf.

Christoph Rösmann, Frank Hoffmann and Torsten Bertram. 2017. 2 springer *Online Trajectory Planning in ROS Under Kinodynamic Constraints with Timed-Elastic-Bands*. https://www.tandfonline.com/doi/full/10.1179/bac.2003.28.1.020.

ChristophRoesmann.“Teb_local_planner.” *2019, july*. http://wiki.ros.org/teb_local_planner.

Denker, Ahmet, and Mehmet Can Işeri. 2017. “Design and Implementation of a Semi-Autonomous Mobile Search and Rescue Robot: SALVOR.” *IDAP 2017 - International Artificial Intelligence and Data Processing Symposium*.

Doucett, Arnaud, Nando De Freitast, and Stuart Russent. 2000. “1301.3853.Pd.”

FabioCapasso. “Link.” *2018, october*. http://wiki.ros.org/urdf/XML/link.

Felix Widmaier. “Eband_local_planner.”*2018, january*. http://wiki.ros.org/eband_local_planner.

Gill, JaspritS.“Navigation.”*2018,october*. http://wiki.ros.org/navigation/Tutorials/RobotSetup.

Grisetti, Giorgio, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. 2010. “A Tutorial on Graph-Based SLAM.” *IEEE Intelligent Transportation Systems Magazine* 2(4): 31–43.

GvdHoorn. “Gmapping.” *2019, february*. http://wiki.ros.org/gmapping.

Identification, Victim et al. “Using Deep Learning to Find Victims in Unknown Cluttered Urban Search and Rescue Environments.”

Jihoonl. “Navfn.” *2014, october*. http://wiki.ros.org/navfn?distro=kinetic.

Kohlbrecher, Stefan, Johannes Meyer, Thorsten Graber, and Karen Petersen. 2017. “Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots.” (April): 1–8.

LaValle, S M. 2011. “Motion Planning: The Essentials.Pdf.” *IEEE Robotics and Automation Society Magazine*.

Louie, Wing Yue Geoffrey, and Goldie Nejat. 2013. “A Victim Identification Methodology for Rescue Robots Operating in Cluttered USAR Environments.” *Advanced Robotics* 27(5): 373–84.

LOZANO-PEREZ, TOMAS. 1983. “Spatial Planning: A Configuration Space Approach.” *Pulp and Paper Canada*: 109–20.

Marin-Plaza, Pablo, Ahmed Hussein, David Martin, and Arturo De La Escalera. 2018. “Global and Local Path Planning Study in a ROS-Based Research Platform for Autonomous Vehicles.” *Journal of Advanced Transportation* 2018.

Mcleod, Haidyn. 2018. “ROS Localisation and Navigation Using Gazebo.” : 1–8.

Murphy, Robin R.et al. 2013. “Springer Handbook of Robotics.” *Choice Reviews Online* 46(06): 46-3272-46–3272.

Nick Lamprianidis.“Carrot_planner.”*2014,january*. http://wiki.ros.org/carrot_planner?distro=melodic.

Nicolas Varas.“Base_local_planner.”*2019 april*. http://wiki.ros.org/base_local_planner?distro=kinetic.

Quinlan, S., and O. Khatib. 2002. “Elastic Bands: Connecting Path Planning and Control.” : 802–7.

Reinzor. “Global_planner.” *2019 , july*. http://wiki.ros.org/global_planner?distro=kinetic.

RicardoAngeli. “Joit.” *2018 ,November*. http://wiki.ros.org/urdf/XML/joint.

Santos, Machado, and Rui P Rocha. 2011. “An Evaluation of 2D SLAM Techniques Available in Robot Operating System.” *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*: 1–6.

Scaramuzza, Davide et al. 2016. “Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age.” *IEEE Transactions on Robotics* 32(6): 1309–32.

Sebastian Thrun, Dieter Fox, Wolfram Burgar, and Frank Dellaert. 1999. “Monte Carlo Localization for Mobile Robots Frank.” *Icra* 2: 1322–28.

Stefan Kohlbrecher, Oskar v, Johannes M, Uwe K. 2011. “A Flexible and Scalable SLAM System with Full 3D Motion Estimation.” http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.302.2579&rep=rep1&type=pdf.

Stefan Kohlbrecher. “Hector_slam.” *2014 , april*. http://wiki.ros.org/hector_slam.

Steve Macenski. “Amcl.” *2019 , june*. http://wiki.ros.org/amcl.

Tadokoro, Satoshi. 2009. Rescue Robotics *Rescue Robotics*.

Thrun, S. “Learning Occupancy Grids with Forward Models.” *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)* 3: 1676–81. http://ieeexplore.ieee.org/document/977219/.

THRUN, Sebastian. 1 MIT press *PROBABILISTIC ROBOTICS*.

Tsung-Yi Lin et al. 2017. “Focal Loss for Dense Object Detection.” (Aug 7): 12980.

APPENDICES

APPENDIX 1

Universal robot description of the rescue robot

```
<robot name="roobot1">
<link name="base_footprint"></link>
  <joint name="base_footprint_joint" type="fixed">
    <origin xyz="0 0 0" rpy="0 0 0" />
    <parent link="base_footprint"/>
    <child link="base_link" />
  </joint>
<link name="base_link">
      <inertial>
        <origin xyz="0 0.1 0" rpy="0 0 0" />
        <mass value="3" />
        <inertia ixx="0.2656" ixy="0" ixz="0" iyy="0.57810" iyz="0" izz="0.8124" />
      </inertial>
      <visual>
        <origin xyz="0 0 0.25" rpy="0 0 0" />
        <geometry>
          <box size="0.75 0.5 0.25" />
        </geometry>
        <material name="">
          <color rgba="0.79216 0.81961 0.93333 1" />
        </material>
      </visual>
      <collision>
        <origin xyz="0 0 0.25" rpy="0 0 0" />
        <geometry>
          <box size="1.5 1 0.25" />
        </geometry>
      </collision>
   </link>
   <link name="Empty_Link1">
      <inertial>
        <origin xyz="0 0 0" rpy="0 0 0" />
        <mass value="1" />
        <inertia ixx="0.0108333" ixy="0" ixz="0" iyy="0.0108333" iyz="0" izz="0.01" />
      </inertial>
      <visual>
        <origin xyz="0 0 0" rpy="1.570795 0 0" />
        <geometry>
          <cylinder length="0.1" radius="0.2" />
        </geometry>
```

```
      <material name="">
        <color rgba="0.79216 0.81961 0.93333 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="1.570795 0 0" />
      <geometry>
        <cylinder length="0.1" radius="0.2" />
      </geometry>
    </collision>
  </link>
  <joint name="front1" type="continuous">
    <origin xyz="-0.25 -0.375 0.2" rpy="0 0 0" />
    <parent link="base_link" />
    <child link="Empty_Link1" />
    <axis xyz="0 1 0" />
  </joint>
  <link name="Empty_Link2">
    <inertial>
      <origin xyz="0 0 0 " rpy="0 0 0" />
      <mass value="1" />
      <inertia ixx="0.0108333" ixy="0" ixz="0" iyy="0.0108333" iyz="0" izz="0.01" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="1.570795 0 0" />
      <geometry>
        <cylinder length="0.1" radius="0.2" />
      </geometry>
      <material name="">
        <color rgba="0.79216 0.81961 0.93333 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="1.570795 0 0" />
      <geometry>
        <cylinder length="0.1" radius="0.2" />
      </geometry>
    </collision>
  </link>
  <joint name="front2" type="continuous">
    <origin xyz="-0.25 0.375 0.2" rpy="0 0 0" />
    <parent link="base_link" />
    <child link="Empty_Link2" />
    <axis xyz="0 1 0" />
  </joint>
```

```
<link name="Empty_Link3">
  <inertial>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <mass value="1" />
    <inertia ixx="0.0108333" ixy="0" ixz="0" iyy="0.0108333" iyz="0" izz="0.01" />
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="1.570795 0 0" />
    <geometry>
      <cylinder length="0.1" radius="0.2" />
    </geometry>
    <material name="">
      <color rgba="0.79216 0.81961 0.93333 1" />
    </material>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="1.570795 0 0" />
    <geometry>
      <cylinder length="0.1" radius="0.2" />
    </geometry>
  </collision>
</link>
<joint name="rear1" type="continuous">
  <origin xyz="0.25 -0.375 0.2" rpy="0 0 0" />
  <parent link="base_link" />
  <child link="Empty_Link3" />
  <axis xyz="0 1 0" />
</joint>
<link name="Empty_Link4">
  <inertial>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <mass value="1" />
    <inertia ixx="0.0108333" ixy="0" ixz="0" iyy="0.0108333" iyz="0" izz="0.01" />
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="1.570795 0 0" />
    <geometry>
      <cylinder length="0.1" radius="0.2" />
    </geometry>
    <material name="">
      <color rgba="0.79216 0.81961 0.93333 1" />
    </material>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="1.570795 0 0" />
```

```
      <geometry>
        <cylinder length="0.1" radius="0.2" />
      </geometry>
    </collision>
  </link>
  <joint name="rear2" type="continuous">
    <origin xyz="0.25 0.375 0.2" rpy="0 0 0" />
    <parent link="base_link" />
    <child link="Empty_Link4" />
    <axis xyz="0 1 0" />
  </joint>
  <link name="Empty_lidar">
    <inertial>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <mass value="0.1" />
      <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy=" 0 0 0" />
      <geometry>
        <mesh filename="package://robot_1/meshes/hokuyo.dae" />
      </geometry>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <box size="0.1 0.1 0.1" />
      </geometry>
    </collision>
  </link>
  <joint name="lidar1" type="fixed">
    <origin xyz="0.375 0 0.5" rpy="0 0 0" />
    <parent link="base_link" />
    <child link="Empty_lidar" />
    <axis xyz="0 1 0" />
  </joint>
 <link name='camera'>
  <inertial>
    <mass value="0.1"/>
    <origin xyz="0.0 0 0" rpy=" 0 0 0"/>
    <inertia
        ixx="1e-6" ixy="0" ixz="0"
        iyy="1e-6" iyz="0"
        izz="1e-6"/>
  </inertial>
```

```
  <collision name='camera_collision'>
    <origin xyz="0 0 0" rpy=" 0 0 0"/>
    <geometry>
      <box size=".05 .05 .05"/>
    </geometry>
  </collision>
  <visual name='camera_visual'>
    <origin xyz="0 0 0" rpy=" 0 0 0"/>
    <geometry>
      <box size=".05 .05 .05"/>
    </geometry>
  </visual>
</link>

<joint type="fixed" name="camera_joint">
  <origin xyz="0.4 0 0.2" rpy="0 0 0"/>
  <child link="camera"/>
  <parent link="base_link"/>
  <axis xyz="0 1 0" rpy="0 0 0"/>
  <limit effort="10000" velocity="1000"/>
  <joint_properties damping="1.0" friction="1.0"/>
</joint>
<gazebo reference="base_link">
  <material>Gazebo/white</material>
</gazebo>
<gazebo reference="Empty_Link1">
  <material>Gazebo/white</material>
</gazebo>
<gazebo reference="Empty_Link2">
  <material>Gazebo/white</material>
</gazebo>
<gazebo reference="Empty_Link3">
  <material>Gazebo/white</material>
</gazebo>
<gazebo reference="Empty_Link4">
  <material>Gazebo/white</material>
</gazebo>
 <gazebo reference="base_footprint">
    <material>Gazebo/Blue</material>
 </gazebo>
 <!-- hokuyo -->
 <gazebo reference="Empty_lidar">
    <sensor type="ray" name="head_hokuyo_sensor">
      <pose>0 0 0 0 0 0</pose>
      <visualize>false</visualize>
```

```
        <update_rate>40</update_rate>
        <ray>
          <scan>
            <horizontal>
              <samples>720</samples>
              <resolution>1</resolution>
              <min_angle>-1.570796</min_angle>
              <max_angle>1.570796</max_angle>
            </horizontal>
          </scan>
          <range>
            <min>0.10</min>
            <max>30.0</max>
            <resolution>0.01</resolution>
          </range>
          <noise>
            <type>gaussian</type>
            <!-- Noise parameters based on published spec for Hokuyo laser
            achieving "+-30mm" accuracy at range < 10m.  A mean of 0.0m and
            stddev of 0.01m will put 99.7% of samples within 0.03m of the true
            reading. -->
            <mean>0.0</mean>
            <stddev>0.01</stddev>
          </noise>
        </ray>
        <plugin name="gazebo_ros_head_hokuyo_controller"
filename="libgazebo_ros_laser.so">
          <topicName>/scan</topicName>
          <frameName>Empty_lidar</frameName>
        </plugin>
      </sensor>
    </gazebo>
    <gazebo>
      <plugin name="skid_steer_drive_controller"
filename="libgazebo_ros_skid_steer_drive.so">
        <updateRate>10.0</updateRate>
        <robotNamespace>/</robotNamespace>
        <leftFrontJoint>front1</leftFrontJoint>
        <rightFrontJoint>front2</rightFrontJoint>
        <leftRearJoint>rear1</leftRearJoint>
        <rightRearJoint>rear2</rightRearJoint>
        <wheelSeparation>0.46</wheelSeparation>
        <wheelDiameter>0.1</wheelDiameter>
        <robotBaseFrame>base_footprint</robotBaseFrame>
        <torque>10</torque>
```

```
      <MaxForce>5</MaxForce>
      <commandTopic>cmd_vel</commandTopic>
      <odometryTopic>odom</odometryTopic>
      <odometryFrame>odom</odometryFrame>
      <broadcastTF>true</broadcastTF>
      <covariance_x>0.001</covariance_x>
      <!-- 0.0001 -->
      <covariance_y>0.001</covariance_y>
      <!-- 0.0001 -->
      <covariance_yaw>0.01</covariance_yaw>
      <!-- 0.01 -->
    </plugin>
  </gazebo>
  <gazebo>
    <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
      <robotNamespace>/roobot1</robotNamespace>
    </plugin>
  </gazebo>
  <gazebo reference="camera">
   <material>Gazebo/Green</material>
   <sensor type="camera" name="camera1">
    <update_rate>30.0</update_rate>
    <camera name="head">
      <horizontal_fov>1.3962634</horizontal_fov>
      <image>
        <width>800</width>
        <height>800</height>
        <format>R8G8B8</format>
      </image>
      <clip>
        <near>0.02</near>
        <far>300</far>
      </clip>
    </camera>
    <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
      <alwaysOn>true</alwaysOn>
      <updateRate>0.0</updateRate>
      <cameraName>roobot1/camera1</cameraName>
      <imageTopicName>image_raw</imageTopicName>
      <cameraInfoTopicName>camera_info</cameraInfoTopicName>
      <frameName>camera</frameName>
      <hackBaseline>0.07</hackBaseline>
      <distortionK1>0.0</distortionK1>
      <distortionK2>0.0</distortionK2>
      <distortionK3>0.0</distortionK3>
```

```
        <distortionT1>0.0</distortionT1>
        <distortionT2>0.0</distortionT2>
      </plugin>
    </sensor>
  </gazebo>
</robot>
```

APPENDIX 2

LAUNCH FILES OF THE NODES OF ROBOT SYSTEM

1. Hector slam launch file

```
<launch>
  <arg name="tf_map_scanmatch_transform_frame_name"
default="scanmatcher_frame"/>
  <arg name="base_frame" default="base_link"/>
  <arg name="odom_frame" default="odom"/>
  <arg name="pub_map_odom_transform" default="true"/>
  <arg name="scan_subscriber_queue_size" default="5"/>
  <arg name="scan_topic" default="scan"/>
  <arg name="map_size" default="2048"/>

  <node pkg="hector_mapping" type="hector_mapping" name="hector_mapping"
output="screen">

    <!-- Frame names -->
    <param name="map_frame" value="map" />
    <param name="base_frame" value="base_link" />
    <param name="odom_frame" value="odom" />

    <!-- Tf use -->
    <param name="use_tf_scan_transformation" value="true"/>
    <param name="use_tf_pose_start_estimate" value="false"/>
    <param name="pub_map_odom_transform" value="$(arg
pub_map_odom_transform)"/>

    <!-- Map size / start point -->
    <param name="map_resolution" value="0.050"/>
    <param name="map_size" value="$(arg map_size)"/>
    <param name="map_start_x" value="0.5"/>
    <param name="map_start_y" value="0.5" />
    <param name="map_multi_res_levels" value="2" />

    <!-- Map update parameters -->
    <param name="update_factor_free" value="0.4"/>
    <param name="update_factor_occupied" value="0.9" />
    <param name="map_update_distance_thresh" value="0.4"/>
    <param name="map_update_angle_thresh" value="0.06" />
    <param name="laser_z_min_value" value = "-1.0" />
    <param name="laser_z_max_value" value = "1.0" />
```

```
    <!-- Advertising config -->
    <param name="advertise_map_service" value="true"/>

    <param name="scan_subscriber_queue_size" value="$(arg
scan_subscriber_queue_size)"/>
    <param name="scan_topic" value="$(arg scan_topic)"/>

    <!-- Debug parameters -->
    <!--
      <param name="output_timing" value="false"/>
      <param name="pub_drawings" value="true"/>
      <param name="pub_debug_output" value="true"/>
    -->
    <param name="tf_map_scanmatch_transform_frame_name" value="$(arg
tf_map_scanmatch_transform_frame_name)" />
  </node>

  <!--<node pkg="tf" type="static_transform_publisher" name="map_nav_broadcaster"
args="0 0 0 0 0 0 map nav 100"/>-->
</launch>
```

2. Move_base launch file

```
<launch>
  <arg name="map_file" default="$(find my_mapping_launcher)/config/my_map.yaml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(arg
map_file)" />

   <param name="use_sim_time" value="true"/>
   <arg name="use_map_topic" default="false" />
   <arg name="scan" default="scan" />
   <arg name="initial_pose_x" default="0.0" />
   <arg name="initial_pose_y" default="0.0" />
   <arg name="initial_pose_a" default="0.0" />
   <arg name="odom_frame_id" default="odom" />
   <arg name="base_frame_id" default="base_footprint" />
   <arg name="global_frame_id" default="map" />
   <node pkg="amcl" type="amcl" name="amcl">
     <param name="use_map_topic" value="$(arg use_map_topic)" />
     <!-- Publish scans from best pose at a max of 10 Hz -->
     <param name="odom_model_type" value="diff" />
     <param name="odom_alpha5" value="0.1" />
     <param name="gui_publish_rate" value="10.0" />
     <param name="laser_max_beams" value="60" />
     <param name="laser_max_range" value="12.0" />
```

```
    <param name="min_particles" value="500" />
    <param name="max_particles" value="2000" />
    <param name="kld_err" value="0.05" />
    <param name="kld_z" value="0.99" />
    <param name="odom_alpha1" value="0.5" />
    <param name="odom_alpha2" value="0.4" />
    <!-- translation std dev, m -->
    <param name="odom_alpha3" value="0.1" />
    <param name="odom_alpha4" value="0.4" />
    <param name="laser_z_hit" value="0.5" />
    <param name="laser_z_short" value="0.05" />
    <param name="laser_z_max" value="0.05" />
    <param name="laser_z_rand" value="0.5" />
    <param name="laser_sigma_hit" value="0.2" />
    <param name="laser_lambda_short" value="0.1" />
    <param name="laser_model_type" value="likelihood_field" />
    <!-- <param name="laser_model_type" value="beam"/> -->
    <param name="laser_likelihood_max_dist" value="2.0" />
    <param name="update_min_d" value="0.25" />
    <param name="update_min_a" value="0.2" />
    <param name="odom_frame_id" value="$(arg odom_frame_id)" />
    <param name="base_frame_id" value="$(arg base_frame_id)" />
    <param name="global_frame_id" value="$(arg global_frame_id)" />
    <param name="resample_interval" value="1" />
    <!-- Increase tolerance because the computer can get quite busy -->
    <param name="transform_tolerance" value="0.3" />
    <param name="recovery_alpha_slow" value="0.0" />
    <param name="recovery_alpha_fast" value="0.0" />
    <param name="initial_pose_x" value="$(arg initial_pose_x)" />
    <param name="initial_pose_y" value="$(arg initial_pose_y)" />
    <param name="initial_pose_a" value="$(arg initial_pose_a)" />
    <remap from="scan" to="$(arg scan)" />
  </node>


  <!-- Move base -->
  <node pkg="tf" type="static_transform_publisher" name="odom_map_broadcaster"
args="0 0 0 0 0 0 /map /odom 100" />
  <node pkg="move_base" type="move_base" respawn="false" name="move_base"
output="screen">
    <param name="use_sim_time" value="true"/>
    <rosparam file="$(find roboot1_move_base)/config/costmap_common_params.yaml"
command="load" ns="global_costmap" />
    <rosparam file="$(find roboot1_move_base)/config/costmap_common_params.yaml"
command="load" ns="global_costmap" />
```

```
   <rosparam file="$(find roboot1_move_base)/config/costmap_common_params.yaml"
command="load" ns="local_costmap" />
   <rosparam file="$(find roboot1_move_base)/config/local_costmap_params.yaml"
command="load" />
   <rosparam file="$(find roboot1_move_base)/config/global_costmap_params.yaml"
command="load" />
  <!--  <rosparam file="$(find
roboot1_move_base)/config/base_local_planner_params.yaml" command="load" /> -->
  <!-- <rosparam file="$(find roboot1_move_base)/config/move_base_params.yaml"
command="load" /> -->
  <rosparam file="$(find roboot1_move_base)/config/ Teb_planner_params.yaml "
command="load" />
    <remap from="cmd_vel" to="cmd_vel"/>
    <remap from="odom" to="odom"/>
    <remap from="scan" to="scan"/>

  <!-- <param name="base_local_planner"
value="base_local_planner/TrajectoryPlannerROS"/> -->
     <param name="base_local_planner" value="teb_local_planner/TebLocalPlannerROS"
/>
  <!-- <param name="base_local_planner"
value="eband_local_planner/EBandPlannerROS"/>  -->
    <param name="controller_frequency" value="30.0" />
    <param name="planner_frequency" value="10.0" />


  </node>
```

2. Darknet ROS launch file

```
<launch>
  <!-- Console launch prefix -->
  <arg name="launch_prefix" default=""/>

  <!-- Config and weights folder. -->
  <arg name="yolo_weights_path"          default="$(find
darknet_ros)/yolo_network_config/weights"/>
  <arg name="yolo_config_path"           default="$(find
darknet_ros)/yolo_network_config/cfg"/>

  <!-- ROS and network parameter files -->
  <arg name="ros_param_file"             default="$(find darknet_ros)/config/ros.yaml"/>
  <arg name="network_param_file"         default="$(find darknet_ros)/config/yolov2-
tiny.yaml"/>
```

```
  <!-- Load parameters -->
  <rosparam command="load" ns="darknet_ros" file="$(arg ros_param_file)"/>
  <rosparam command="load" ns="darknet_ros" file="$(arg network_param_file)"/>

  <!-- Start darknet and ros wrapper -->
  <node pkg="darknet_ros" type="darknet_ros" name="darknet_ros" output="screen"
launch-prefix="$(arg launch_prefix)">
    <param name="weights_path"          value="$(arg yolo_weights_path)" />
    <param name="config_path"          value="$(arg yolo_config_path)" />
  </node>

 <!--<node name="republish" type="republish" pkg="image_transport" output="screen"
                                                                    args="compressed
in:=roobot1/camera1/image_raw raw out:=/camera/image_raw" /> -->
</launch>
```

3. YOLO v3 launch file

```
<launch>

  <!-- Use YOLOv3 -->
  <arg name="network_param_file"         default="$(find
darknet_ros)/config/yolov3.yaml"/>


  <!-- Include main launch file -->
  <include file="$(find darknet_ros)/launch/darknet_ros.launch">
    <arg name="network_param_file"    value="$(arg network_param_file)"/>
  </include>

</launch>
```

APPENDIX 3

YAML FILES OF THE ROBOT SYSTEM

1. Base_local_planner_params.yaml

```
TrajectoryPlannerROS:
  max_vel_x: -0.2
  min_vel_x: -0.1
  max_vel_theta: -0.02
  min_in_place_vel_theta: 0.10
  escape_vel: 0.02

  acc_lim_theta: 0.5
  acc_lim_x: 0.5
  acc_lim_y: 0.5
  goal_distance_bias: 0.8
  path_distance_bias: 1.0
  gdist_scale: 0.8
  pdist_scale: 5.0
  occdist_scale: 0.01
  heading_lookahead: 0.325

  sim_time: 4

  meter_scoring: true
  holonomic_robot: false

TebLocalPlannerROS:

  odom_topic: odom
  map_frame: /odom

  teb_autosize: True
  dt_ref: 0.3
  dt_hysteresis: 0.03
  global_plan_overwrite_orientation: True
  max_global_plan_lookahead_dist: 1.0
  feasibility_check_no_poses: 1

  max_vel_x: -0.8
  max_vel_x_backwards: -0.2
  max_vel_y: 0.0
  max_vel_theta: 0.4
```

```
wheelbase: 0.0
 acc_lim_x: 0.5
 acc_lim_y: 0.00
 acc_lim_theta: 0.5

 footprint_model: # types: "point", "circular", "two_circles", "line", "polygon"
 type: "polygon"
 radius: 0.85 #
 vertices: [ [-0.6, -0.85], [-0.6, 0.85], [0.6, 0.85], [0.6, -0.85] ]

 # GoalTolerance
 xy_goal_tolerance: 0.2
 yaw_goal_tolerance: 0.1
 free_goal_vel: False

 # Obstacles
 min_obstacle_dist: 0.5
 include_costmap_obstacles: True
 costmap_obstacles_behind_robot_dist: 1.0
 obstacle_poses_affected: 10
 costmap_converter_spin_thread: True
 costmap_converter_rate: 10
 inflation_dist: 0.8
 dynamic_obstacle_inflation_dist: 0.6
 obstacle_association_force_inclusion_factor: 2.0
 obstacle_association_cutoff_factor: 4.0

 costmap_converter_plugin: "costmap_converter::CostmapToPolygonsDBSMCCH"
 costmap_converter_spin_thread: True
 costmap_converter_rate: 5
 costmap_converter/CostmapToPolygonsDBSMCCH:
 cluster_max_distance: 0.4
 cluster_min_pts: 2
 cluster_max_pts: 30
 convex_hull_min_pt_separation: 0.1
```

2. Costmap_common_params.yaml

```
map_type: costmap
global_frame: map
robot_base_frame: base_footprint
obstacle_range: 10
raytrace_range: 12

footprint: [ [-0.6, -0.85], [-0.6, 0.85], [0.6, 0.85], [0.6, -0.85] ]
```

transform_tolerance: 0.3

robot_radius: 0.85
inflation_radius: 0.85

observation_sources: laser_scan_sensor
laser_scan_sensor: {sensor_frame: Empty_lidar, data_type: LaserScan, topic: scan, marking: true, clearing: true}

3. Teb_planner_params.yaml

```
TebLocalPlannerROS:

  #odom_topic: odom
  #map_frame: /odom

  teb_autosize: True
  dt_ref: 0.3
  dt_hysteresis: 0.03
  global_plan_overwrite_orientation: True
  max_global_plan_lookahead_dist: 5.0
  feasibility_check_no_poses: 1

  wheelbase: 0.0
  acc_lim_x: 0.25
  acc_lim_y: 0.25
  acc_lim_theta: 1.5
  footprint_model: # types: "point", "circular", "two_circles", "line", "polygon"
    type: "circular"
    radius: 0.9 #
 #  vertices: [ [-0.6, -0.85], [-0.6, 0.85], [0.6, 0.85], [0.6, -0.85] ]

  # GoalTolerance
  xy_goal_tolerance: 0.2
  yaw_goal_tolerance: 0.1
  free_goal_vel: False

  # Obstacles
  min_obstacle_dist: 0.4
  include_costmap_obstacles: True
  costmap_obstacles_behind_robot_dist: 1.0
  obstacle_poses_affected: 10
  costmap_converter_spin_thread: True
  costmap_converter_rate: 5
```

```
  min_obstacle_dist: 0.35
  inflation_dist: 0.45
  dynamic_obstacle_inflation_dist: 0.6
  obstacle_association_force_inclusion_factor: 2.0
  obstacle_association_cutoff_factor: 4.0
```

4. Global_costmap_params.yaml

```
global_costmap:
   global_frame: map
   robot_base_frame: base_footprint
   update_frequency: 10
   publish_frequency: 10
   resolution: 0.06
   static_map: true
   rolling_window: false
```

5. Local_costmap_params.yaml

```
local_costmap:
   global_frame: odom
   robot_base_frame: base_footprint
   update_frequency: 5
   publish_frequency: 2
   width: 10
   height: 10
   origin_x: 0.0
   origin_y: 0.0
   resolution: 0.02
   static_map: false
   rolling_window: true
```

4. Move_base_params.yaml

```
shutdown_costmaps: false

controller_frequency: 30.0
controller_patience: 3.0

planner_frequency: 30.0
planner_patience: 5.0

oscillation_timeout: 10.0
oscillation_distance: 0.2
base_local_planner: " teb_local_planner/TebLocalPlannerROS "
```