ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Ph.D. Dissertation

on the topic:

"Robot Navigation in Unknown and Dynamic Environment"

---

by:
Neerendra Kumar
(Ph.D. Student)

Supervisor:
Dr. Zoltán Vámossy
(Associate Professor)

Doctoral School of Applied Informatics and Applied Mathematics

Budapest, 2020

Declaration of Authorship

I, **Neerendra Kumar**, declare that the thesis entitled, '**Robot Navigation in Unknown and Dynamic Environment**' and the content presented in it, is entirely my own research work in the able supervision of **Dr. Zoltán Vámossy** (Associate Professor).

It is hereby assured that:

- This work has been completed as a candidate for PhD degree at Óbuda University.
- Wherever in the thesis the published work of other authors is consulted, this work is clearly referenced.
- Wherever in the thesis quotations from other authors are used, then this fact is clearly mentioned. The thesis is absolutely my own work with the exception of such quotations.
- All the main sources of help have been properly acknowledged .

Budapest

September 3, 2020.

[illegible]
....

Neerendra Kumar
(PhD Candidate)

"We will either find a way or make one"

General Hannibal Barca

Acknowledgements

Above all, I am extremely grateful to my Ph.D. supervisor, **Dr. Zoltán Vámossy**, for his continuous expert guidance, support and motivation during my Ph.D. research work. He always responded my queries, emails and paper corrections very quickly. Further, my supervisor, always supported me, promptly, about the document requirements of the Universities pertaining to this study. I have always got positive response from him for our meetings to discuss the research problems and the possible solutions.

I am greatly thankful to **Prof. Aurél Galántai**, the head of the *Doctoral School of Applied Informatics and Applied Mathematics* of the *Óbuda University*, for his immense support and leadership to improve the results and to achieve the final goal of the work. I am highly thankful to **Prof. László Horváth** for his valuable and motivational comments on my oral presentations of end semester research reports and on my research paper presentations in various internal conferences. I am grateful to **Prof. Levente Kovács** for his interesting and easy to grasp subject lectures and his continuous inspiration to attend the international conferences related to my work. From bottom of my heart, I am thankful to **Prof. József K. Tar** for his knowledgeable subject lectures and his provocation to use open source software.

I am wholeheartedly thankful to **Mr. Zsolt Miklós Szabó-Resch** for providing me his technical expertise and guidance on *Linux, robot operating system, Gazebo simulator and robot handling.* My profound thanks to **Dr. György Eigner** for sharing the important information related Ph.D. degree procedure and formats of required documents and research reports. My sincere thanks to **Ms. Zsuzsanna Bácskai** and **Ms. Katalin Hersics** for their administrative support throughout my study at *Óbuda University.*

I am greatly thankful to the governments of **Hungary** and **India** for selecting me for the *Tempus Public Foundation's* prestigious *Stipendium Hungaricum Scholarship Programme* to get admission in this Ph.D. programme.

A lot of thanks to **Prof. Ashok Aima**, Vice Chancellor of Central University of Jammu, India, for granting me the study leave to pursue this Ph.D. study. I am grateful to **Prof. Lokesh Verma**, **Prof. Devanand** and my fellow colleagues of *Central University of Jammu* for their support in the study leave.

I would also like to thank to all of those who are not namely listed here, like, my professors, many of my friends and colleagues from whom I got inspired and learnt a lot.

Finally, I give my heartily thanks to my loving family members for their support.

Abstract

This dissertation presents the research work carried out on robot navigation. The *introduction* and the *conclusion* of the study are provided in Chapters 1 and 6, respectively. Chapter 2 presents scientific methods applied in the research work. Chapter 3 is dedicated to the robot navigation in known and static environment. The robot navigation in unknown and dynamic environment has been considered in Chapter 4. In Chapter 5, advanced algorithms are proposed for the obstacle recognition and obstacle avoidance in unknown and static environments.

The dissertation has covered the robot-navigation-problem in the three main categories of the environments in which the robot may navigate.

Firstly, the robot navigation in known and static environment is considered in Chapter 3. In Chapter 3, the A* algorithm is implemented in the global path planner of the robot. The result obtained from the different heuristic functions has been compared. Further, the Probabilistic Roadmaps (PRM) technique is used to find an obstacle-free-path from start to goal.

Secondly, the robot navigation in unknown and dynamic environment is taken into consideration in Chapter 4. In Chapter 4, an algorithm for obstacle avoidance has been evolved using the bumper hit events of the robot. Further, robot navigation models are presented using MATLAB-Simulink. The robot navigation models use fuzzy controllers for obstacle avoidance during robot navigation. For the fuzzy controllers, appropriate Fuzzy Inference Systems (FISs) are developed and implemented by using Mamdani and Sugeno types of FISs.

Thirdly, the robot navigation in unknown and static environment is presented in Chapter 5. In Chapter 5, the main results are presented by a theses group. The theses group contains three thesis points (5.4.1−5.4.3). The thesis point in 5.4.1 is based on the proposed algorithm for obstacle recognition. The thesis point 5.4.2 is obtained from the proposed advanced algorithm possessing obstacle recognition and obstacle avoidance features. Finally, the "two sample t-test" is applied, for obstacle recognition and avoidance in robot navigation, to present the thesis point 5.4.3.

The newly proposed concepts, methods and algorithms are tested on simulated *Turtlebot* robot in *Gazebo* simulator. The experimental results for the real *Turtlebot* robot has also been included wherever it was possible. As background information, the related appendices are provided at the end of the dissertation.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

ANFIS	Adaptive Neuro-Fuzzy Inference System
ANN	Artificial Neural Network
ANNs	Artificial Neural Networks
FIS	Fuzzy Inference System
FISs	Fuzzy Inference Systems
MF	Membership Function
NDT	Normal Distribution Transform
PRM	Probabilistic Roadmaps
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping

1. Introduction

1.1. Background of the research

Mobile robots have a wide range of applications like space missions, household and office work, receiving-delivering orders of clients in restaurants, transportation of logistics in inventories, inspection and maintenance, agriculture, security and defence, operations in radioactive areas etc. Mobile robots are very useful in the area where either the task is boring due to the repetitiveness of the same operations or the working environment is hazardous for the human being. For mobile robots, the navigation from one location to the other is one of the most desirable operations [1–4]. On the basis of the available prior information, the navigation environments can be classified into two major types: known environment and unknown environment. Further, the navigation environments may or may not change during the navigation task. Generally, for the navigation point of view, the navigation environments get changes due to the moving obstacles. Therefore, a navigation environment is considered as static if the obstacles in the path are static. On contrast, a navigation environment is dynamic if the obstacles are dynamic. Consequently, the applicable navigation strategies depend on the type of the navigation environment [5–7].

The navigation process can be subdivided into two parts: global navigation and local navigation. The global navigation method is used to find globally optimal path on the basis of prior information like map of the environment. However, in the absence of sufficient prior information the global navigation strategy is not applicable. Instead of the prior information, the local navigation is based on the on-line sensory data received from the sensors mounted on the robot. Therefore, the local navigation methods can also be applied when the navigation environment is unknown or partially known. In real-world scenarios, *support vector machines* based local planner can be efficiently integrated into an autonomous navigation system [8]. Global path planner and local path planner are the two main parts of the navigation system of autonomous robot. These two path planners, in cooperation, make the robot motion optimal and collision free. The global planner generates a feasible route from the robot's current location and orientation to the goal location and orientation. The local path planner moves the robot as per the global plan and dynamically adapts to dynamic environment state [9–12].

In known environments, path planning is the process of finding the best feasible path from start to goal location [13–16]. In addition, self localization and mapping are among the challenging tasks [NK-17]. Simultaneous Localization and Mapping (SLAM) yields a map of the environment and keeps track of the robot in the navigation environment with various objects [18–20]. SLAM and path planning are necessary for the autonomous navigation process [21]. A path planning algorithm to calculate convenient motions of the robot by taking the different optimization criteria like time, energy and distance into account to overcome unknown and challenging obstacles is proposed in [22]. In known and static environment, the robot navigation process can have four major steps: making the map of surrounding world, store the map in computer readable form, find a feasible path from start to goal in the stored map and then drive the robot on the observed path. The process of map building includes the following three steps:

(i) Select the appropriate two-dimensional coordinate points in the environment world covering the whole working area.

(ii) Drive the robot following these selected coordinates, and receive the sensor readings.

(iii) In the occupancy grid framework, mark the free and occupied spaces in the world by setting the probabilities of occupied spaces as one and the probabilities of free spaces as zero.

The occupancy grid can be saved and used directly in robot path planning in a convenient way [23]. Representation of navigation environment as a grid-based space is a necessary task for robust and accurate motion planning of robot. A two-dimensional occupancy grid can be constructed using the data of LASER sensor before the execution of the motion planning function [24]. Many researchers used the occupancy grid representation of the environment world map in the robot navigation related work as in [25–28]. The occupancy grid map is computer readable and this map can be supplied to the computer program to find an obstacle free path. Recently, these concepts are used in [29, 30]. A minimum cost path can be acquired by applying a classical approach [31]. For a mobile robot, bacterial potential field method can be used to compute the feasible, optimal and safe paths in environments with static and dynamic obstacles [32]. For real-time path planning in a complex and dynamic environment, a biologically inspired two level method gives the plus of both global and local path finding procedures [33].

Graph-based search algorithms may take larger computation time to find the globally optimal path because these algorithms exhaustively explore the search space. In this case, a heuristic function increases the speed of exploration process by providing the estimate of the cost of reaching the goal node [34]. Useful heuristic functions for A* algorithm can be obtained using the various distances given in [35]. In a hierarchical path planning approach, the A* algorithm finds a geometric path speedily and various path points can be chosen as sub-goals for the second level. The *least-squares policy iteration* algorithm can be used to acquire a near-optimal local planning strategy to generate smooth trajectories in the second level. Consequently, an optimized path can be found by sequentially achieving the sub-goals generated in the first level [36, 37]. The global map should only consider the obstacles which are persistent and larger than a definite area. Inclusion of many small obstacles in the global map may lead to the larger complexity of the topological space. Homotopic A* algorithm, as path planner, is presented and compared with other path planners in [38]. However, the different heuristic functions are not taken into consideration in the implementation of A* algorithm as global path planner for the autonomous navigation of mobile robots.

Probabilistic roadmaps technique can be used to find a path from start to the goal point in occupancy grid map [39]. In PRM technique, the given number of nodes are created in the free space of occupancy grid and then these nodes are connected to each other within some threshold connection distance. Further, one path from the available connected paths from start to goal is selected. The process of sampling and node addition in PRM is given in [40]. Various path planning strategies can be found in recent research work [41, 42]. To drive the robot on the given path, a path following algorithm is required. An implementation of pure pursuit path tracking algorithm with some limitations is presented in [43].

In unknown environments, the robot navigation becomes more challenging because no satisfactory information on the environment is available before starting the navigation process. Therefore, a global navigation path may not be generated. In this case, a local path from the current position to the goal position may provide a solution. Further, in the case of robot navigation in known environment, there may be situations where the obstacles are dynamic. So, the actual positions of the dynamic obstacles may not be measured accurately in global path planning. Furthermore, the partially known environment consists of known and unknown obstacles. In the case of unknown and dynamic environment, the robot needs to avoid the obstacles by using its inbuilt sensors. A robot can not only avoid the obstacles but also find a globally optimal path by dynamically altering the locally optimal paths [44, 45]. In these cases, the local path planning is a suitable option for the navigation purpose [46]. In case of unknown environment, navigation task needs an approach that can work in uncertain situation [47].

The information about velocities of the dynamic obstacles is not necessarily required for the obstacle avoidance in the navigation path [48].

Modern control theory and robotics are advancing greatly due the development of new technologies [49]. Commonly, there are two steps in the design of the controllers for mobile robots. Firstly, the data from sensors are computed into high level and meaningful values of variables. Secondly, some machine learning method is used to produce a controller. Taking these high level variables as input, the controller outputs the control commands for the robot [50].

Humans perform the navigation task without any exact computation and mathematical modelling. The ability of humans to deal with uncertainties can be followed in robot navigation using the neural network and fuzzy logic. Fuzzy logic gives an abstract behaviour of the system in an intuitive form even in the absence of precise mathematical and logical model. In practice, the sensor information is noisy and unreliable. The sensor inputs to the robot can be mapped into the control actions using fuzzy rules. Therefore, fuzzy logic is a suitable tool to achieve robust robot behaviour [51]. The obstacles can be treated as integrated part of the environment. Human-like approaches to avoid the obstacles can be used in robot navigation [52]. In the environments with moving and deforming obstacles, a purely reactive algorithm to navigate a mobile robot mathematically guarantees collisions avoidance [53]. A fuzzy logic based navigation approach which uses grid based map and a behaviour based navigation method is given in [54]. However, this approach requires the environmental information in grid map in advance. In different conditions and uncertainty, an optimal and robust fuzzy controller for path tracking is presented in [55]. A Mamdani-type fuzzy controller [56] for wheeled robot can be optimal for trajectory tracking to deal with parametric and non-parametric uncertainties [57]. Neural networks have the ability to work with imprecise information and are excellent tools applicable in obstacle avoidance by mobile robots. Reliable and fault-tolerant control can be obtained with neural control systems [58]. In robot navigation, neural network can be employed to map the relationships between inputs and outputs for interpreting the sensory data, obstacle avoidance and path planning. Although, the neural network method is slow and the learning algorithm used may not lead to an optimal solution. Thus, the integrated approaches like neuro-fuzzy technique are much more suitable for the robot navigation task [16]. In complex and unknown environments, simulation experiments indicate better navigation performance using neuro-fuzzy approach. Using neuro-fuzzy approach, the parameters of the controller can be optimized and the structure of the controller can be self-adaptive. To improve automatic learning and adaptation, ANFIS combines fuzzy logic and neural network. ANFIS can be used to predict and model several engineering systems. ANFIS is a fuzzy based model which is trained using some data set. Consequently, ANFIS computes the best suitable parameters of membership functions involved in FIS [59, 60].

LASER scanner is one of the most common sensors used to execute SLAM [61–63]. Using a LASER scan, the Normal Distribution Transform (NDT) can model the distribution of all two dimensional points around the robot. Two successive LASER scans can be aligned using NDT to recover the translation and rotational parameters between the two scan positions. A map can be defined as the collection of LASER scans along with their global poses [64]. In the process of alignment of two LASER scans, the point of the LASER scan sample of the second scan in its coordinate frame is mapped into the first scan coordinate frame. Importantly, this mapping can be said optimal if the sum of the normal distributions of mapped points of second scan into first one using the mean and covariance of the NDT of the first scan is maximum. For autonomous robots [65, 66], a survey on heuristic approaches in robot path planning is given in [16]. Artificial Neural Network (ANN) can compensate erroneous sensor data. Moreover, the ANN can be trained during the SLAM [67]. Regardless of the uncertainties in the sensors observations, robot may navigate safely from start to goal [47]. Using range finder sensors (e.g. LASER scan sensors), the robot can navigate from start to goal with obstacle avoidance in unknown and dynamic environments environment [68]. The obstacle size and velocity vector of the unmanned surface vehicles can be used to achieve the obstacle avoidance. Moreover, fusion of more than one navigation algorithms can result more accurate outcomes [69]. The issue of obstacle avoidance is treated as an optimization problem in [70]. Previous experiences during the robot navigation are helpful to predict the path with obstacle avoidance of static and dynamic obstacles [71]. Various requirements of obstacle avoidance can be satisfied for the changing distance between robot and the obstacles [72–74].

The combination of several sensor systems is used in mobile robots. For making navigation decision, sensor fusion is the task of combining the information into a usable form [75]. The complex task of programming generally prevents the use of industrial robots to a large extent [76]. In addition, there may be situations in the autonomous navigation in an unknown environment [77] that certain sensors of the robot do not work properly or they are removed to minimize programming and system complexity.

The present study begins with the robot navigation task in known environment and then advances to the robot navigation in unknown environment.

1.1.1. Research gaps in the existing solutions

Considering the background of the research (Section 1.1), the research gaps for the dissertation can be given by the following points ((i) – (iii)):

(i) It is evident from the existing research work that the robot navigation task in case of known-static environment is less-complex than in the case of unknown-dynamic environment. Therefore, to begin with, classical path planning strategies such as A* algorithm and PRM can be studied in known-static environment. As yet, no significant comparison of application of various heuristic functions in A* algorithm has been done for path planning.

(ii) The existing research has established that the *fuzzy logic* based techniques are best suitable for uncertain data. So, in case of unknown-dynamic environment, *fuzzy logic* based robot navigation model becomes an exciting research point to be considered. Thus far, Robot navigation model using Mamdani-type FIS, Sugeno-types FIS, and ANFIS has not been presented to obtain obstacle avoidance in simulated as well as real world unknown-dynamic environment.

(iii) A search robot may need to navigate through all around the working area. Hence, to avoid the repetitive paths in unknown environment, a search robot should recognise the obstacles visited earlier. Till present, for obstacle recognition and avoidance, "standard deviation" and the "t-test" have not been applied for a search robot in unknown environment.

1.2. Dissertation outline

The current chapter (Chapter 1) presents the basis and the state of the art of the study.

Chapter 2 presents the scientific methods applied in the research work. The Section 2.1 gives the details of hardware and software used in the study. In Section 2.2 of the chapter, mathematical foundations, of applied soft-computing techniques, has been described.

Chapter 3 is about the robot navigation in known and static environment. In this chapter, two approaches have been implemented for robot navigation. Firstly, A* algorithm has been applied and the results are compared for different heuristic functions. Secondly, the robot navigation is performed using the probabilistic road-maps and finding an optimal path from start to goal.

Chapter 4 is concerned with the robot navigation in unknown and dynamic environment. This chapter contains three Sections. In the Section 4.1 , an algorithm using the bumper events of the robot has been proposed for the obstacle avoidance. A model for robot navigation using Mamdani-type FIS is presented in Section 4.2. Section 4.3 is about the robot navigation with obstacle avoidance using fuzzy controller based on Sugeno-type FIS.

Chapter 5 is based on the robot navigation in unknown and static environments. There are three sections in this chapter. In Section 5.1, obstacle recognition, during the robot navigation, is achieved by comparing the standard deviations of the LASER scans' distance range vectors. By combining the standard deviations of the distance range vectors, robot positions and time of scans, an advanced algorithm, for obstacle avoidance and breaking of the repetitive paths loops, is presented in Section 5.2. Section 5.3 considers t-test to check out the similarity of two readings LASER scan. Consequently, the outcome of the t-test has been used for obstacle recognition and avoidance during the robot navigation.

Finally, in Chapter 6, conclusion of the study, applicability of the obtained results, and future research scope based on the dissertation are provided.

The organisational structure of the dissertation is shown by Fig. 1.1.

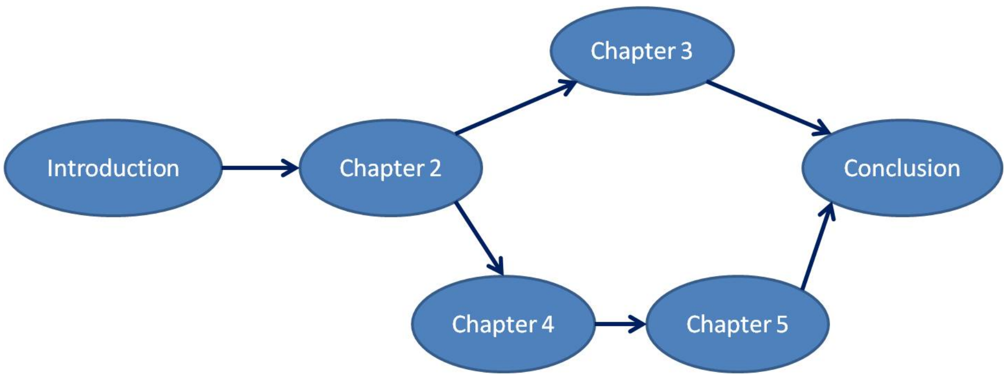


Figure 1.1.: Structure of the dissertation.

2. Hardware, software, and scientific methods applied in the research work

During the research work efficient combination of the already known algorithms and scientific methodologies such as A*, probabilistic roadmap search are applied to achieve the new research results.

2.1. Hardware and software used for experimental set-up

In case of hardware used, the experimental work is carried out on a general purpose laptop computer containing 4x Intel core i3- 2350M CPU @ 250-GHz and 3952 MB memory. A real Turtlebot robot is used to for testing the proposed methods. Turtlebot is a wheeled mobile robot. This robot kit is useful for the researcher due to its low cost and open source. The main hardware components of Turtlebot are *Turtlebot Structure*, *Turtlebot Module Plate*, *Kinect*, *Robot Operating System (ROS) compatible Netbook*, *Asus Xion Pro Live*, and *Kobuki Base*. In addition to the software development kit (SDK), the robotic software development environment of the Turtlebot provides libraries for the several useful tools like visualization, control and error handling. To bring up the Turtlebot in the real system, the ROS command *$ roslaunch turtlebot_bringup minimal.launch* can be used in the Ubuntu (Operating System) terminal.

The main software used in the work are as follows:

- The operating system installed on the computer is Ubuntu, version 14.04.5 LTS. The operating system has been updated to *Ubuntu 16.04.4 LTS* in Section 5.2.
- As ROS, Indigo distribution with version 1.11.20 is considered.
- To simulate the navigation task, Gazebo simulator version 2.2.6 is used till the Section 5.1. Thereafter, the Gazebo version is updated to 7.0.
- For computer programming purpose, **C++** language is used in the Sections 3.1 and 4.1. In the remaining Sections, MATLAB is considered for the programming task. Section-wise, MATLAB release version related Information is given in Table 2.1.

Table 2.1.: MATLAB releases and Toolboxes used.

Section	MATLAB release used	Toolboxes used
3.2	R2016b	Robotics System Toolbox
4.2	R2016b	Simulink, Robotics System Toolbox, Fuzzy Logic Toolbox
4.3	R2016b	Simulink, Robotics System Toolbox
5.1	R2018a	Robotics System Toolbox
5.2	R2018a	Simulink, Robotics System Toolbox
5.3	R2019a	Robotics System Toolbox

2.2. Mathematical foundations of applied soft computing techniques

Modern soft computing tools, fuzzy logic, artificial neural networks, and neuro-fuzzy inference systems are also used in our work. The soft computing tools serve as approximate technical realization of the universal approximators (e.g. [78–80]).

2.2.1. Artificial neural network (ANN)

An artificial neuron (perceptron) mimics the computational function of biological neuron. Fig. 2.1 shows a model of artificial neuron presented by [81].

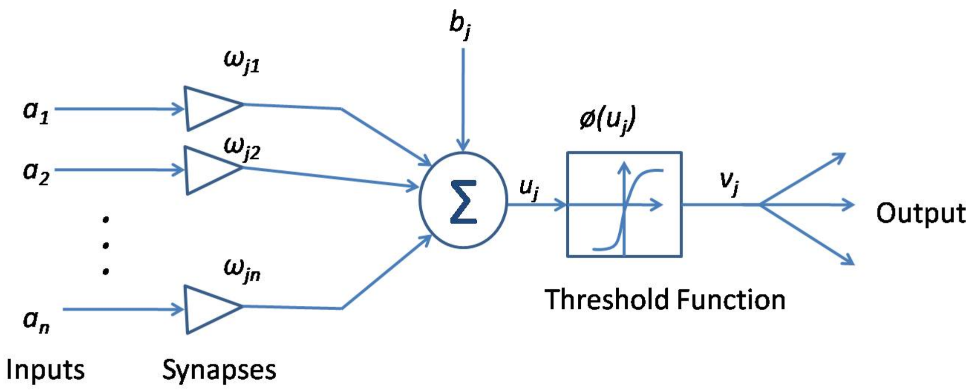


Figure 2.1.: Artificial neuron model given by [81].

The output (v_j) of an artificial neuron can be defined using (2.1).

$$\begin{aligned} u_j &= \sum_{i=1}^{n} \omega_{ji} a_i + b_j \\ v_j &= \phi\left(u_j\right) \end{aligned} \tag{2.1}$$

where,

b_j = bias or offset for j^{th} neuron in the layer.

ω_{ji} = synaptic weights for j^{th} neuron in the layer.

a_i = input to the neuron

Multilayer perceptron (a feedforward neural network) is layered network of artificial neurons. To obtain parallel computation, a layer of the ANN consists several artificial neurons. In each layer of this type of Artificial Neural Networks (ANNs), functions of the artificial neurons are similar. The output of the neurons in a layer is passed to the neurons of the next layer. This network type is applicable for the approximation of non-linear multiple variable functions by supervised training that means setting the number of neurons in the layers, and following that, setting the connection weights and biases by minimizing the mapping error for the samples used for training. In *error backpropagation* [82] learning, the *gradient descent* method is used for training the network. In our work, *multilayer perceptron*, with *error backpropagation* learning, has been applied.

In other types of ANNs, *recurrent neural networks* (e.g. Hopfield's and Elman's networks [83, 84]) that can model system's dynamics, *convolutional neural networks* that have come into fashion recently for recognizing typical patterns in various scales in strongly non-linear processes e.g. in fluid dynamics [85] and work on the basis of the approximation abilities of the Volterra Polynomials [86], or *self-organizing maps* as Kohonen's network (e.g. [87]) that can learn without supervision. However, these type of ANNs are not applied in our work.

2.2.2. Fuzzy logic

2.2.2.1. Fuzzy set

Ordinary sets have a limited scope of applicability. In real life problems, the ordinary set are not well suited to define the membership of objects in case of human thinking. An ordinary set can also be defined using MF. The membership function of an ordinary subset Y of the set U (universe of discourse) can be defined as follows:

$\chi_Y : U \to \{0,1\}$

$$\chi_Y(x) = \begin{cases} 0, & x \notin Y \\ 1, & x \in Y \end{cases} \tag{2.2}$$

This definition of ordinary sets is extended for the fuzzy sets using the membership function. According to [88], in a fuzzy set, objects have continuum membership function values. Fuzzy set $\left(\widetilde{Y}\right)$ can be defined by (2.3), as follows:

$$\widetilde{Y} = \left\{ \left(x, \mu_{\widetilde{Y}}(x)\right) : x \in U \right\}. \tag{2.3}$$

where, $\left(x, \mu_{\widetilde{Y}}(x)\right)$ is ordered pair of an element (x) and the actual value of its corresponding membership function $\left(\mu_{\widetilde{Y}}(x)\right)$. The membership function values range between zero and one $\left(\mu_{\widetilde{Y}}(x) : U \to [0,1]\right)$.

Support $\left(supp\left(\widetilde{Y}\right)\right)$ and core $\left(core\left(\widetilde{Y}\right)\right)$ of $\widetilde{Y}$ can be expressed by (2.4) as follows:

$$\begin{aligned} supp\left(\widetilde{Y}\right) &= \left\{x \in U \mid \mu_{\widetilde{Y}}(x) > 0\right\}, \\ core\left(\widetilde{Y}\right) &= \left\{x \in U \mid \mu_{\widetilde{Y}}(x) = 1\right\}, \end{aligned} \tag{2.4}$$

Various generalizations of the classical AND, and OR operators are developed for the fuzzy sets, on the basis of which the fuzzy inference systems can be elaborated. Also, various generalizations of the classical negation operator $(\neg)$ can be elaborated, and the statements of the De Morgan's Laws can be maintained for fuzzy sets (e.g. [89–91]).

2.2.2.2. Membership functions

In π-shaped fuzzy sets, the membership function increases from zero to a maximum (≤ 1) and after reaching maximum it decreases to zero. This type of membership functions are used to present the fuzzy concepts like: ‘around’, ‘approximately’, ‘close to’.

The well known membership functions for π-shaped fuzzy sets are *bell*, *triangle*, and *trapezoid*. A triangle membership function can be defined using (2.5).

$$\mu(x) = \begin{cases} 0, & \text{for } (x \leq p) \text{ or } (x \geq q) \\ \frac{x-p}{a-p}, & \text{for } p \leq x \leq a \\ \frac{q-x}{q-a}, & \text{for } a \leq x \leq q \end{cases} \tag{2.5}$$

where, the interval $[p, q]$ and the point a are support and core of the MF, respectively.

Fig. 2.2 shows a triangle MF by taking the values of parameters as indicated in its caption. If the membership occurs at a point then triangle membership function is widely used. When the membership occurs in domain then the trapezoid membership function is applicable. Fig. 2.3 represents a trapezoid MF including the values of its parameters.

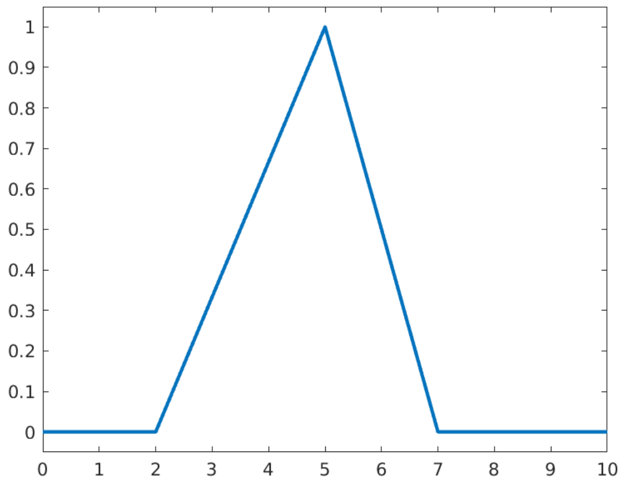


Figure 2.2.: Triangle MF having support and core as $[2, 7]$ and 5, respectively.

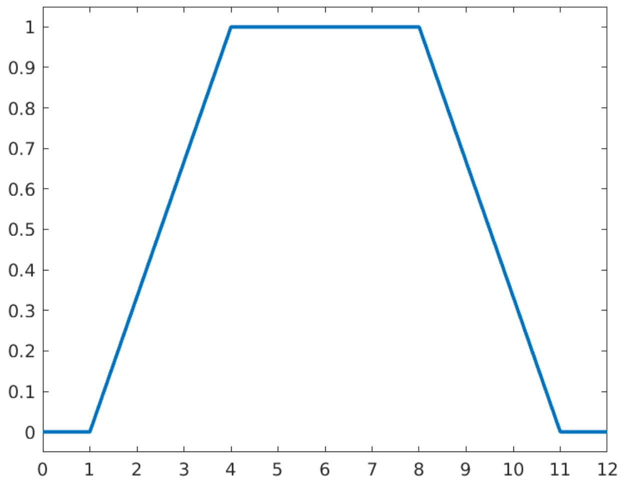


Figure 2.3.: Trapezoid MF by taking support and core intervals as $[1, 11]$ and $[4, 8]$, respectively.

The trapezoid membership function can be defined by (2.6).

$$\mu(x) = \begin{cases} 0, & \text{for } (x \leq p) \text{ or } (x \geq q) \\ \frac{x-p}{a-p}, & \text{for } p \leq x \leq a \\ 1, & \text{for } a \leq x \leq b \\ \frac{q-x}{q-b}, & \text{for } b \leq x \leq q \end{cases} \tag{2.6}$$

where, the intervals $[p, q]$ and $[a, b]$ are support and core of the MF, respectively.

As non-linear membership functions, bell MF and Gaussian MF are widely used.

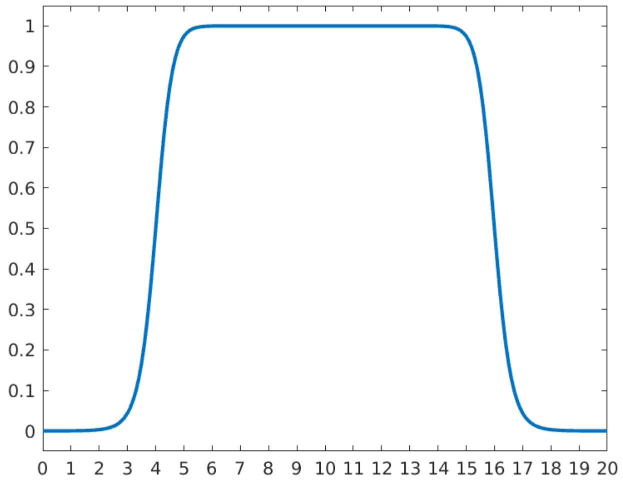


Figure 2.4.: Generalized bell MF by taking $w = 6$, $s = 8$, $c = 10$.

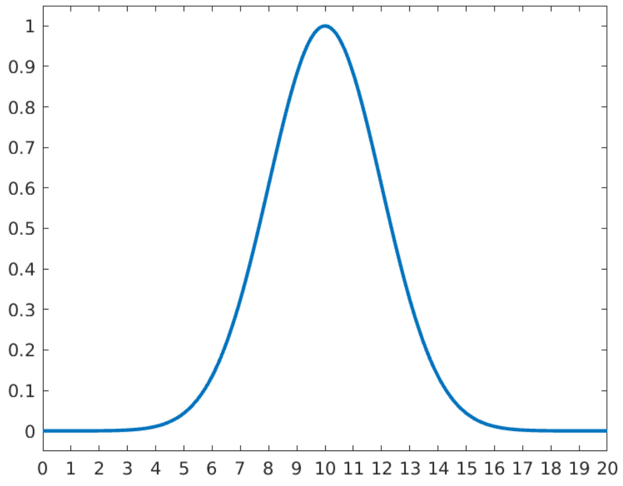


Figure 2.5.: Gaussian MF having values of the parameters as $\sigma = 2$, $m = 10$.

The generalised bell membership function is defined in (2.7) as follows:

$$\mu(x) = \frac{1}{1 + \left|\frac{x-c}{w}\right|^{2s}} \tag{2.7}$$

where,

w defines the width of the MF.

s defines the curve-shape on both sides of the plateau.

$c =$ center of the MF.

Shape of a generalized bell MF, with the values of parameters as $w = 6, s = 8, c = 10$, is given in Fig. 2.4.

Gaussian membership function can be defined as given in (2.8).

$$\mu(x) = e^{\frac{-(x-m)^2}{2\sigma^2}} \tag{2.8}$$

where, m and σ are the mean and the standard deviation, respectively.

2.2.3. Fuzzy inference system

A model of FIS is presented in Fig. 2.6. In Fig. 2.6, the *input* and the *output* are crisp values.

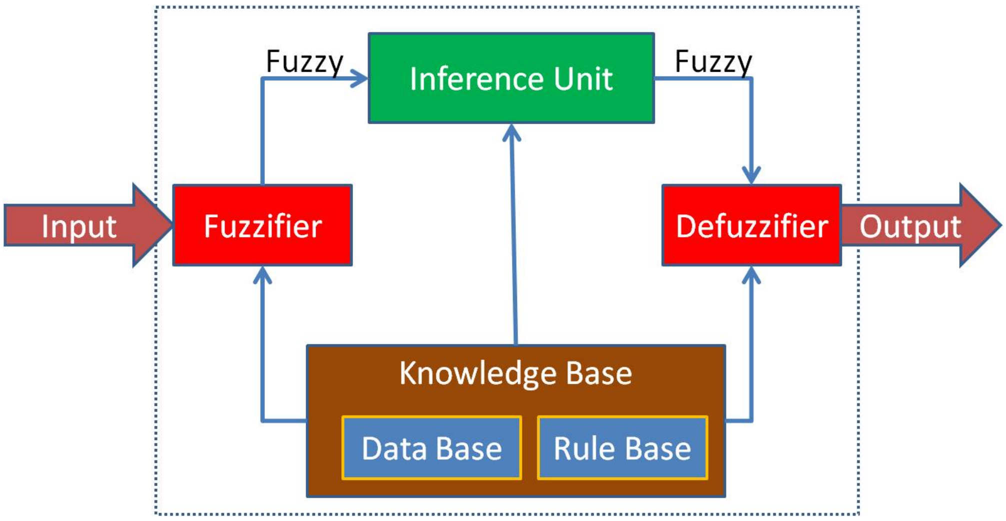


Figure 2.6.: A model of fuzzy inference system presented in [92].

The *inference unit* of the fuzzy inference system takes decisions by using the *IF. . . THEN* rules provided by the *knowledge base*. There are the following two main methods of FIS:

(i) Mamdani FIS [93].

(ii) Takagi-Sugeno FIS (also known as Sugeno FIS) [94].

2.2.3.1. Fuzzification

Fuzzification is a mapping of a point (real-valued) $x^* \in U \subset \mathbb{R}$ to a fuzzy set $\widetilde{Y} \in U$.

The widely used fuzzification methods are singleton fuzzifier, Gaussian fuzzifier, and triangular fuzzifier.

(i) **Singleton fuzzifier:**

$$\mu_{\widetilde{Y}}(x) = \begin{cases} 1, & \text{if } x = x^* \\ 0, & \text{otherwise} \end{cases} \tag{2.9}$$

(ii) **Gaussian fuzzifier:**

$$\mu_{\widetilde{Y}}(x) = e^{-\left(\frac{x_1 - x_1^*}{c_1}\right)^2} \star e^{-\left(\frac{x_2 - x_2^*}{c_2}\right)^2} \star \ldots \star e^{-\left(\frac{x_n - x_n^*}{c_n}\right)^2} \tag{2.10}$$

where, $\{c_i, i = 1, \ldots, n\}$ are positive constants and the operator $\star$ corresponds to some "generalized AND operator". Usually, $\star$ is min or algebraic product.

(iii) **Triangular fuzzifier:**

$$\mu_{\widetilde{Y}}(x) = \begin{cases} \left(1 - \frac{|x_1 - x_1^*|}{c_1}\right) \star \ldots \star \left(1 - \frac{|x_n - x_n^*|}{c_n}\right), & \text{if } |x_i - x_i^*| \geq c_i, \text{ for } i = 1 \text{ to } n. \\ 0, & \text{otherwise} \end{cases} \tag{2.11}$$

where, c_i and $\star$ have the same meaning as in (2.10).

2.2.3.2. Defuzzification

Let $\widetilde{Y}$ is a fuzzy set defined over U with its membership function $\mu_{\widetilde{Y}}(x), x \in U$, and let x^* denotes the defuzzification of $\mu_{\widetilde{Y}}(x)$. Main methods to obtain x^* are *centroid of gravity*, *center of sum* method, *mean of max* method, *height* method, *bisector* defuzzification, *smallest of maximum* defuzzification, *largest of maximum* defuzzification, and *weighted average* defuzzification method.

The *center of gravity* method is given in (2.12).

$$x^* = \frac{\int x\mu_{\widetilde{Y}}(x)dx}{\int \mu_{\widetilde{Y}}(x)dx} \tag{2.12}$$

where, $\int \mu_{\widetilde{Y}}(x)dx$ is the area bounded by $\mu_{\widetilde{Y}}$ and x^* represents x-coordinate of the center of gravity of the area.

If the membership function has n different disjoint peaks then the *center of sum* method, defined in (2.13), can be applied.

$$x^* = \frac{\sum_{i=1}^{n} x_i A_{\widetilde{Y}_i}}{\sum_{i=1}^{n} A_{\widetilde{Y}_i}} \tag{2.13}$$

here, $A_{\widetilde{Y}_i}$ represents area bounded by the $\widetilde{Y}_i$ and x_i denotes geometric center of the area.

Weighted average method is also known as *Sugeno defuzzification* method. The crisp output using this method is presented in (2.14).

$$x^* = \frac{\sum_{i=1}^{n} x_i \mu_{\widetilde{Y}_i}(x_i)}{\sum_{i=1}^{n} \mu_{\widetilde{Y}_i}(x_i)} \tag{2.14}$$

where, x_i is the middle value of the i^{th} peak, $\widetilde{Y}_i(x)$.

2.2.4. Adaptive neuro-fuzzy inference system (ANFIS)

Neuro-fuzzy systems combine:

(i) the learning and parallel computational abilities of neural networks,

(ii) human-like comprehension and knowledge representation of the fuzzy systems.

Adaptive network is a superset of feed-forward neural networks having supervised learning. Fuzzy rules of ANFIS are developed using Sugeno FIS [94]. Form of a Sugeno fuzzy rule for can be given as (2.15).

$$\textit{Rule}: \ \textbf{If } x \text{ is } X_i \text{ and/or } y \text{ is } X_j \textbf{ then } z = f(x, y) \tag{2.15}$$

The function $f(x, y)$, in (2.15), is defined by (2.16) as follows:

$$f(x, y) = \begin{cases} d, & \text{for zero-order Sugeno FIS} \\ ax + by + c, & \text{for first-order Sugeno FIS} \end{cases} \tag{2.16}$$

where, $\{a, b, c\}$ is the set of parameters. x, y are input variables and X_i, X_j are fuzzy sets for $i, j = 1$ to n. n represents total number of fuzzy sets for the input and d is a constant.

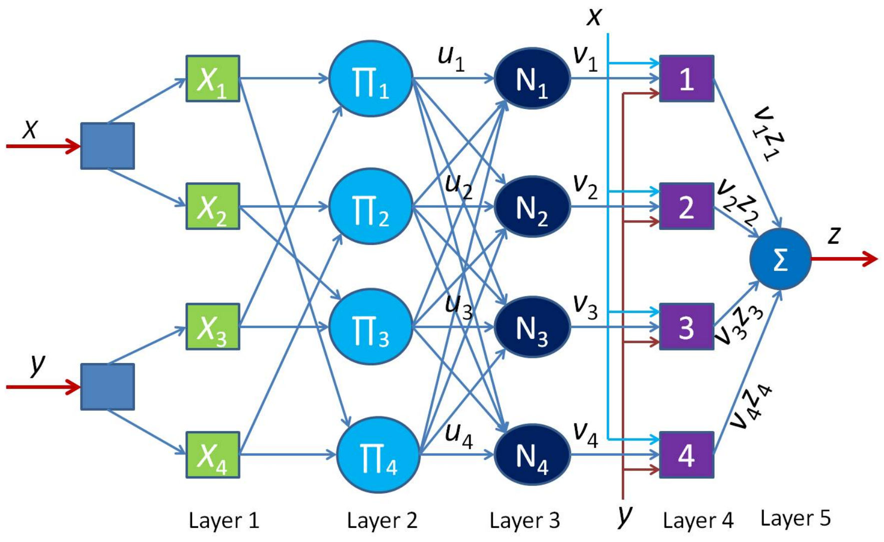


Figure 2.7.: A four-rule-ANFIS Structure using the model of [92].

Layer structure of a four-rule-ANFIS (Fig. 2.7) can be given by using ANFIS model of [92].

The functions of nodes in the *layer 1* are similar as given in (2.17):

$$O_i^{(1)} = \mu_{X_i}(x) \tag{2.17}$$

where,

x = input to node i.

X_i = corresponding linguistic label of the node for $i = 1$ to n.

n = total number of nodes in the *layer 1*.

μ_{X_i} = membership function of X_i. According to need, the membership function (μ_{X_i}) can be selected from the definitions given in (2.5), (2.6), (2.7), (2.8).

In the *layer 2*, each node multiplies its input values. The output of a node in *layer 2* is defined as (2.18).

$$u_i = \mu_{X_j}(x) \times \mu_{X_k}(y) \tag{2.18}$$

where, the operator $\times$ is generalised AND.

In case of ANFIS structure given in Fig. 2.7, $i = 1$ to 4, $j = 1, 2$, and $k = 3, 4$.

Output of each node represents rule's firing strength.

In the *layer 3*, the output of i^{th} node is ratio of firing strength of i^{th} rule to the sum of the firing strengths of all rules, as given in (2.19).

$$v_i = \frac{u_i}{\sum_{k=1}^{n} u_k} \tag{2.19}$$

where, n is the total number of rules in the ANFIS.

The output of i^{th} node in the *layer 4* is defined in (2.20).

$$O_i^{(4)} = v_i z_i, \text{ for } i = 1 \text{ to } n. \tag{2.20}$$

where, n is the total number of rules in the ANFIS structure.

The *layer 5* has single node. The output (z) of this layer can be expressed as in (2.21).

$$\begin{gathered} z = \sum_{i=1}^{n} v_i f_i, \\ f_i = a_i x + b_i y + c_i, \end{gathered} \tag{2.21}$$

where, $\{a_i, b_i, c_i\}$ is the set of parameters for i^{th} rule, and n is the total number of rules in the ANFIS structure [92, 94].

3. Robot navigation with obstacle avoidance in known and static environment

3.1. Heuristic approaches in robot navigation

This section presents the implementation of A* algorithm as a global path planner for the navigation of a Turtlebot robot. A map of the navigation environment has been developed for a Gazebo simulator's world. Manhattan distance, octile distance, and Euclidean distance heuristic functions are used to estimate the cost of moving from a cell to the goal cell of the environment. Time taken by the global planner, length of the path covered and the number of cells to visit are presented in tabular form. Mat plots for global costmap changes are also given using 'rqt' tool of ROS.

Considering the different heuristic functions, the A* algorithm has been implemented for the autonomous navigation of Turtlebot robot. For the local planner, the default dynamic window approach of ROS is used.

Remaining part of this section is organized as follows: Mathematical background of A* algorithm and the different heuristic functions used are given in Section 3.1.1. Section 3.1.2 presents the implementation and results of the study. Finally, Section 3.1.3 is for the summary of the work.

3.1.1. A* algorithm and the heuristic functions

The A* algorithm provides the solution by searching among all possible paths and selects the path that minimizes $f(x_1, y_1)$, the sum of the cost of the path from the start node to the node (x_1, y_1) and estimated cost of the cheapest path from the node (x_1, y_1) to the goal node [34].

$$f(x_1, y_1) = g(x_1, y_1) + h(x_1, y_1), \tag{3.1}$$

where, $g(x_1, y_1)$ is the cost of the path from the start node to (x_1, y_1), and $h(x_1, y_1)$ is a heuristic that estimates the cost of the cheapest path from (x_1, y_1) to the goal.

Let $|x|$ represent the absolute value of x and C represent the minimum cost of moving from one space to adjacent space. Various distance metrics and distances including the following are given in [35].
The *Manhattan distance heuristic* function is expressed in (3.2) as follows:

$$h(x_1, y_1) = C(|x_1 - x_2| + |y_1 - y_2|), \tag{3.2}$$

The *diagonal distance heuristic* is defined in (3.3) as follows:

$$h(x_1, y_1) = C(|x_1 - x_2| + |y_1 - y_2|) + (D - 2C)\ Min(|x_1 - x_2|, |y_1 - y_2|), \tag{3.3}$$

In (3.3), $Min(|x_1 - x_2|, |y_1 - y_2|)$ gives the minimum value between the two arguments and D represents the cost of moving diagonally. The *diagonal distance* is called *octile distance* when $C = 1$ and $D = \sqrt{2}$.
The *Euclidean distance heuristic* is given by (3.4).

$$h(x_1, y_1) = C\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}, \tag{3.4}$$

3.1.2. Implementation and experimental results

The A* algorithm is implemented in C++ for the three heuristic functions ((3.2) – (3.4)) given in Section 3.1.1. Methods to create user-defined Gazebo-world and to make communication between software packages using ROS are provided in Appendices A.1 and A.2, respectively. An introduction to pre-defined and user-defined ROS nodes is provided in Appendix A.3. ROS packages based method to create map of the navigation environment and robot's autonomous navigation with given map are explained in Appendix A.5.

Using *gmapping* package of ROS, map of *corridor.world* (Fig. A.2) is built. Description about the *corridor.world* is given in Appendix A.1.2. Fig. 3.1 represents the map of *corridor.world*.

Fig. 3.2 represents the global *costmap* updates for the map presented in Fig. 3.1. The *costmap* is a data structure which provides the information of the safe places for the robot navigation. The *costmap* uses sensor data from the world to produce an occupancy grid map. The global *costmap* is applicable in global navigation. In Fig. 3.2, the description of variables is as follows:

'*/move_base/global_costmap/costmap_updates/height*' = map height
'*/move_base/global_costmap/costmap_updates/width*' = map width
'*/move_base/global_costmap/costmap_updates/x*' = map's x origin (in global frame)
'*/move_base/global_costmap/costmap_updates/y*' = map's y origin (in global frame)

here, the measurement unit of the variables is meter.

Due to erroneous sensor data, the height and width of the map produced varies slightly during the map building process.

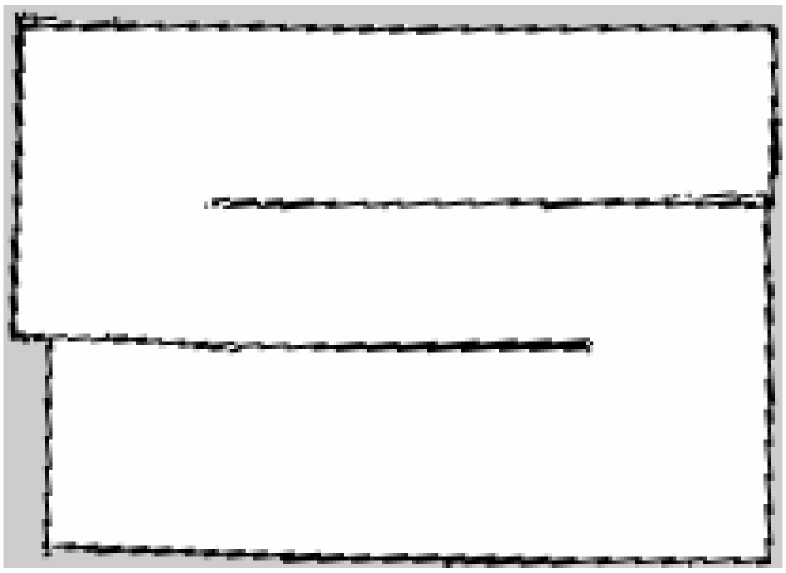


Figure 3.1.: Map of *corridor.world* using *gmapping* package of ROS.

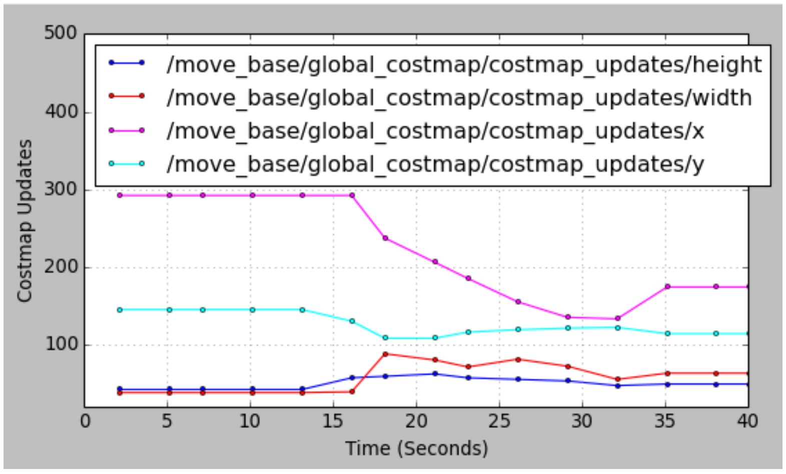


Figure 3.2.: Global costmap updates for *corridor.world*

The *costmap* allocates an unique ID (Cell ID) to each cell of the occupancy grid map. The two-dimensional coordinates of the center of a cell are considered as the *cell coordinates* of that cell. Tables 3.2–3.4 provide the details of visited cell coordinates, current and goal cell IDs, global planner time to reach the goal cell from the current cell, estimated distance from current cell to goal cell in meters and number of cells to visit using octile distance, Euclidean distance and Manhattan distance heuristics, respectively. Goal cell IDs and corresponding goal coordinates used for Tables 3.2−3.4 are given in Table 3.1.

Table 3.1.: Goal cell IDs and their coordinates used for Tables 3.2−3.4.

Table Number	**Goal Cell ID**	**Goal Cell Coordinates** (X_2, Y_2)
3.2	78511	$(-5.01013, -4.96942)$
3.3	77968	$(-4.99168, -5.00425)$
3.4	78512	$(-4.99091, -4.99587)$

Table 3.2.: Planner time, path length and number of cells to visit using octile distance heuristic.

Start Cell ID	**Start Cell Coordinates**	**Planner Time (Micro-Sec.)**	**Path Length (Meters)**	**Number of Cells to visit**
89512	$(1.04364, -3.95498)$	0.973915	6.55208	125
89513	$(1.0669, -3.95554)$	2.04848	6.57279	125
87336	$(1.03579, -4.17968)$	1.68727	6.38137	122
81344	$(0.610965, -4.70829)$	1.15402	5.75355	114
79159	$(0.164069, -4.90315)$	0.961782	5.22071	105
78605	$(-0.326765, -4.98568)$	1.75893	4.7	95
78595	$(-0.82855, -4.99297)$	0.728234	4.2	85
78585	$(-1.33259, -4.98773)$	0.782637	3.7	75
78575	$(-1.83213, -4.98842)$	0.650034	3.2	65
78565	$(-2.33186, -4.98401)$	0.93469	2.7	55
78555	$(-2.81918, -4.97933)$	0.406216	2.2	45
78545	$(-3.30742, -4.98043)$	0.688205	1.7	35
78536	$(-3.75951, -4.98735)$	0.312633	1.25	26
78527	$(-4.21368, -4.985)$	0.110972	0.8	17
78520	$(-4.5861, -4.98205)$	0.166721	0.45	10
78516	$(-4.7804, -4.98784)$	0.062063	0.25	6
78513	$(-4.93287, -4.99858)$	0.11679	0.1	3
77969	$(-4.93267, -5.00009)$	0.056646	0.120711	3
77969	$(-4.93653, -5.01733)$	0.066747	0.120711	3

It is clear from the data of the Tables 3.2−3.4 that the planner time taken to move the robot from current cell to next cell using Manhattan distance heuristic changes moderately during the navigation. In the case of octile distance heuristic, the global planner time, taken for cell transitions during the robot navigation, changes swiftly. The global planner time for robot navigation from one cell to the next cell of the path is varying largely. Own publication related to Section 3.1 is given in [NK-95].

Table 3.3.: Planner time, path length and number of cells to visit using Euclidean distance heuristic.

Start Cell ID	**Start Cell Coordinates**	**Planner Time (Micro-Sec.)**	**Path Length (Meters)**	**Number of Cells to visit**
89512	$(1.03746, -3.95512)$	5.70497	6.52279	124
89512	$(1.04253, -3.95519)$	13.2484	6.52279	124
86792	$(1.03515, -4.21765)$	12.3271	6.33137	121
82431	$(0.594576, -4.64405)$	5.09661	5.71569	112
81878	$(0.105449, -4.66552)$	18.8623	5.24498	103
81868	$(-0.379453, -4.66033)$	2.29739	4.74498	93
81858	$(-0.873481, -4.68571)$	2.07009	4.24498	83
81848	$(-1.38019, -4.69493)$	5.98157	3.74498	73
81838	$(-1.87859, -4.68794)$	2.93626	3.24498	63
81828	$(-2.39392, -4.68642)$	4.32241	2.74498	53
81818	$(-2.86297, -4.68288)$	1.70456	2.24498	43
81809	$(-3.33114, -4.68346)$	1.44749	1.79497	34
81800	$(-3.78121, -4.6896)$	2.78576	1.34497	25
80703	$(-4.22774, -4.75153)$	0.238801	0.853554	16
79607	$(-4.62291, -4.86843)$	0.285834	0.412133	8
79059	$(-4.83649, -4.94868)$	0.123107	0.191421	4
78513	$(-4.91739, -4.98285)$	0.053015	0.0707109	2
78513	$(-4.91724, -4.98161)$	0.07666	0.0707109	2
78513	$(-4.93315, -4.9917)$	0.051287	0.0707109	2

Table 3.4.: Planner time, path length and number of cells to visit using Manhattan distance heuristic.

Start Cell ID	Start Cell Coordinates	Planner Time (Micro-Sec.)	Path Length (Meters)	Number of Cells to visit
90056	$(1.03149, -3.94109)$	1.2466	6.55208	125
90057	$(1.06115, -3.94355)$	2.02882	6.57279	125
88425	$(1.0752, -4.05798)$	3.1792	6.42279	122
82433	$(0.688094, -4.61363)$	5.29837	5.79497	114
80248	$(0.239083, -4.84698)$	0.971193	5.26213	105
78606	$(-0.254425, -4.97342)$	1.19347	4.7	95
78052	$(-0.750472, -5.00026)$	6.06353	4.22071	85
78587	$(-1.24928, -4.99091)$	0.568116	3.75	76
78577	$(-1.746, -4.98364)$	0.852299	3.25	66
78567	$(-2.23996, -4.98492)$	1.33613	2.75	56
78557	$(-2.74472, -4.97907)$	0.311586	2.25	46
78547	$(-3.2144, -4.98195)$	0.586167	1.75	36
78538	$(-3.67516, -4.98319)$	0.280542	1.3	27
78529	$(-4.12344, -4.98548)$	0.204794	0.85	18
78522	$(-4.4943, -4.9849)$	0.187987	0.5	11
78518	$(-4.67007, -4.98192)$	0.196806	0.3	7
78514	$(-4.8814, -4.98692)$	0.118818	0.1	3
78514	$(-4.88038, -4.98649)$	0.061638	0.1	3
78514	$(-4.87757, -4.99671)$	0.121212	0.1	3
77970	$(-4.8891, -5.01557)$	0.561779	0.120711	3

3.1.3. Summary

The three different heuristic functions are used in the implementation of A* algorithm as a global path planner for the Turtlebot robot. It is observed that the coordinates of the cells on the path generated by each of the heuristic function do not differ significantly. Nevertheless, the time taken by global path planner, with different heuristic function used, differ significantly. Among the three, the Euclidean distance heuristic produces the most non-uniform global path planner time for the transition of the robot between the cells during navigation. On the other hand, octile distance heuristic depicts the most uniform behaviour for the global path planner time throughout the navigation path.

3.2. Robot path pursuit using probabilistic roadmaps

A collision free path can be obtained through the path planning using PRM and can be applied to related research as used in [96–99]. This section presents the implementation of pure pursuit algorithm using PRM in robot navigation. The map of the robot's environment is generated as occupancy grid. In the occupancy grid map, the PRM are obtained. A desired path from start to end location of the robot navigation is obtained from PRM.

Rest of the Section is organized as follows: Section 3.2.1 covers the required preliminary work before obtaining the PRM. Theoretical background of the PRM technique is described in Section 3.2.2. Section 3.2.3 is for the pure pursuit path tracking of the path obtained in the previous section. Section 3.2.4 summarizes the work.

3.2.1. Preliminary work

3.2.1.1. Extracting optimal way-points from the environment

In the map building process, the first step is to bring-up the robot in the environment. Consequently, the coordinates of the environment can be noted down by moving the Turtlebot using any of the Turtlebot tele-operator. The robot can be driven in the environment using the tele-operator. In this manner, the desired positional coordinates of the robot can be obtained and written in a text file for the future use.

A method to write robot positions in a text file is given in Appendix A.6. The way-points (robot positions), extracted using the method, are input to the robot. Using the input way-points, the robot visited in the *corridor.world* (Fig. A.2). Fig. 3.3 depicts the path visited by the robot during the map building process.

During the map building process, starting from the position $(0.0107, 0.0000)$, the robot travelled along the yellow coloured line then orderly followed red, green and blue coloured positions as shown in Fig. 3.3. To obtain map of the environment, the positional coordinates act as way-points of the path to be followed by the robot.

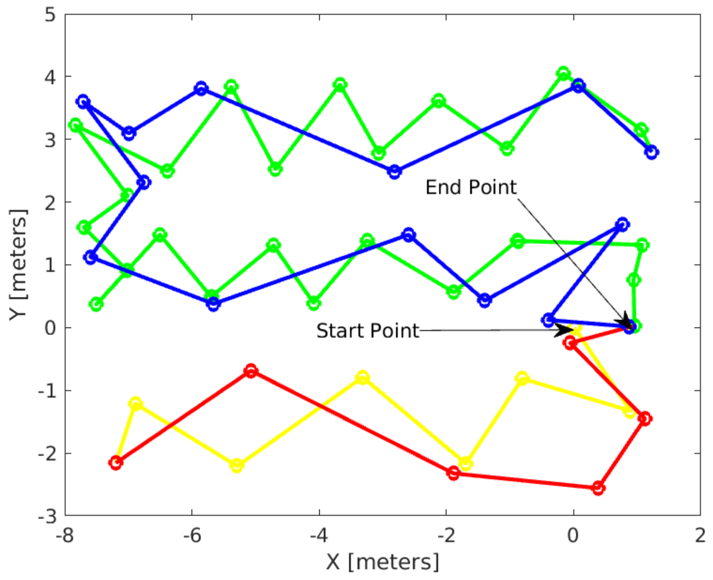


Figure 3.3.: Plot of way-points. From the start point $(0.0107, 0.0000)$ to the end point $(0.8779, 0.0150)$, the robot orderly followed the colour sequence yellow, red, green and blue.

3.2.1.2. Occupancy grid map of the environment

The map of the environment can be generated as a two-dimensional occupancy grid. Method to create occupancy grid map using MATLAB is described in Appendix A.7.

For *corridor.world* as shown in Fig. A.2, the values of width, height, and grid resolution are taken as 10.2 meter, 7.6 meter and 20, respectively. Initially, the probability values of all the grid cells are close to 1 indicating that the whole area is occupied and there is no free space to move the robot. In the process of map building, the robot can be driven following the path as shown in Fig. 3.3.

Fig. 3.4 shows the occupancy grid map of the *corridor.world* obtained by using the given way-points. Before inserting the laser scan reading into the occupancy grid, it is needed to remember that occupancy grid's x and y axes start at 0. On the other hand, the robot's environment may contain negative values for x and y. Therefore, it is required to convert the actual coordinates of the environment to the corresponding coordinates in the occupancy grid map. The corresponding occupancy grid coordinates $(X^{'}, Y^{'})$ can be obtained from robot's environment world coordinates (X, Y) by using (3.5) as follows:

$$\begin{aligned} X^{'} &= X + |L_x|\,, \\ Y^{'} &= Y + |L_y|\,, \end{aligned} \tag{3.5}$$

where, L_x and L_y are the approximate lowest values of X and Y, respectively.

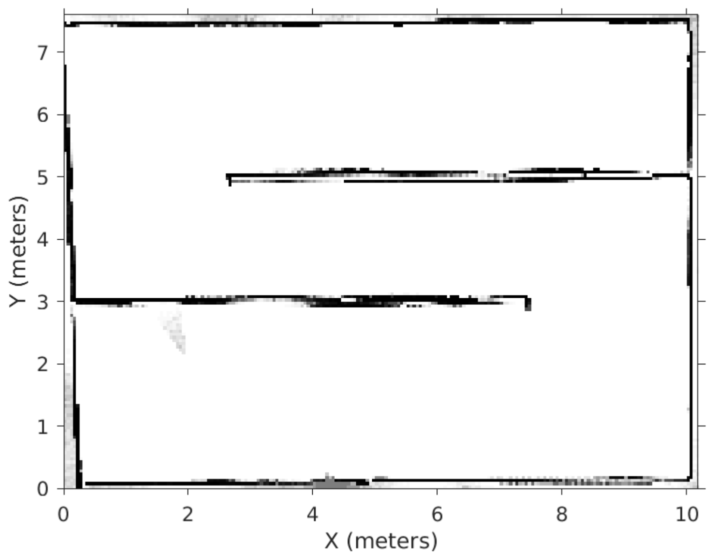


Figure 3.4.: Occupancy grid map of *corridor.world*.

In case of *corridor.world*, the values of L_x and L_y can be taken as -8.3 meter and -3 meter, respectively. For obstacle avoidance, the occupancy grid map must be inflated up to robot radius. The radius of the Turtlebot robot is taken as 0.20 meters. Fig. 3.5 presents the inflated map of the map shown in Fig. 3.4.

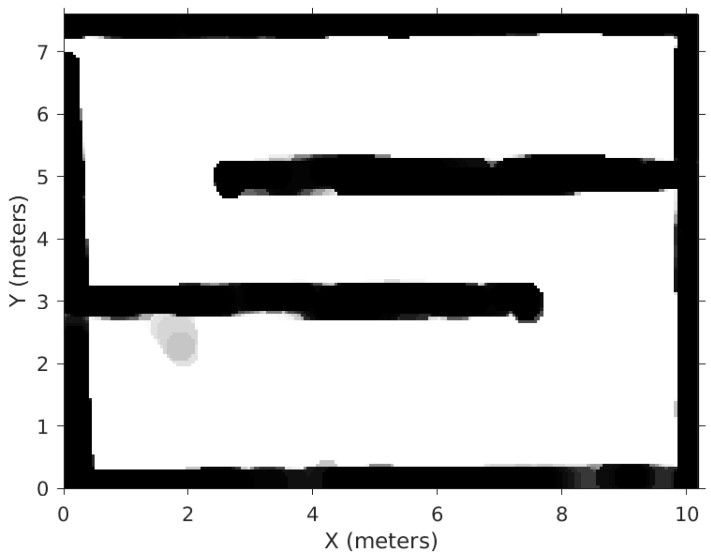


Figure 3.5.: Inflated occupancy grid map.

3.2.2. Probabilistic roadmaps

Probabilistic roadmaps path planning method has two phases, namely, *learning phase* and *query phase*. In the *learning phase*, for a given scene, a roadmap (a data structure) is generated in probabilistic manner. The generated probabilistic roadmap is saved as an undirected graph. The nodes of the graph represent collision-free configurations of the robot and the edges of the graph represent feasible paths. Local planner of the navigation system computes these paths. The *learning phase* of the PRM is summarized using Algorithm 1. Initially, the graph $G = \{N_G, E_G\}$ is empty.

During the *query phase*, a query is generated to find a path between two collision free configurations. Initially, the method generates a path from start and goal to other two nodes in the roadmap. Consequently, the graph is searched to find the connecting edges to these two nodes. Eventually, a feasible path from start to goal is received by concatenating the path segments. It is experienced that this method provides good results when larger time span is given in learning phase. Various components of the method can be tailored to increase the efficiency of this method [39]. The method is applicable to the static environments where the obstacles do not get change.

Algorithm 1 *Learning phase* of PRM.

```
 1: $N_G, E_G \leftarrow \{\}$
 2: while true do
 3:     $r \leftarrow$ a randomly selected free configuration
 4:     $N_r \leftarrow \{r' \in N_G \mid D(r, r') \leq d_m\}$
 5:     $N_G \leftarrow N_G \cup \{r\}$
 6:     for all $x \leftarrow N_r$, in ascending order of $D(r, x)$ do
 7:         if $P(r, x)$ & $\neg C(r, x)$ then
 8:             $E_G \leftarrow E_G \cup (r, x)$
 9:             Update connected components of G.
10:         end if
11:     end for
12: end while
```

where,
d_m = Maximum threshold distance between r and r'.
$P(r, x)$ returns whether a path, between r and x, is found by local planner.
$C(r, x)$ returns whether r and x belongs to the same connected component.
$D(r, x)$ is defined in (3.6).

$$D(r, x) = \max_{p \in robot} \|p(x) - p(r)\|, \tag{3.6}$$

In (3.6),

p represents a point on in the robot.

$p(r)$ = workspace position of p when the robot is at r.

$\|p(x) - p(r)\|$ = Euclidean distance between $p(r)$ and $p(x)$.

The two important properties of PRM are the number of nodes created on the map and threshold connection distance between these nodes. The PRM algorithm places the given number of nodes on the map and connects each node to every other node within the threshold distance of connections between nodes. Large number of nodes taken can provide more feasible paths. As a result, it gives more efficient path. However, increasing the number of nodes in PRM increases the computation time. Therefore, just enough node in PRM to cover the whole map may be a good solution. The threshold distance of connections between nodes has direct impact on the number of connections. Lower threshold connection distance reduces the number of available paths. Consequently, these available paths are considered to select one final obstacle free path. A well-suited combination of number of node in the PRM and threshold connection distance is required for a particular environment map. In general, large number of nodes and small threshold connection distance enhances the chance of finding an efficient path in complex environment. In case of simple environment map (i.e. with less number of obstacles), small number of nodes and large connection distance between nodes can lead to a solution efficiently.

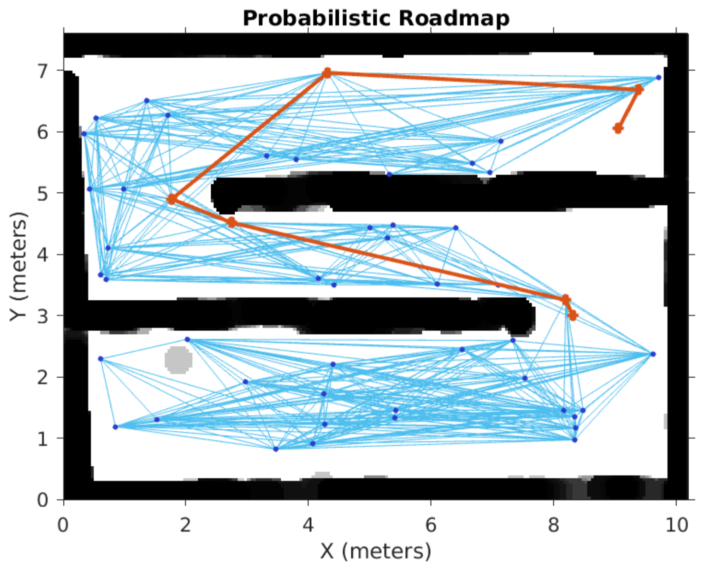


Figure 3.6.: Instance of PRM with 50 nodes.

The effect of different number of nodes in a PRM can be observed by taking a fixed connection distance between the nodes of the PRM. The system is tuned-up with 50 numbers of nodes and the connection distance as 10 meters. Fig. 3.6 presents an instance of PRM and path. Method to generate probabilistic roadmaps using MATLAB is given in Appendix A.8.

3.2.3. Path pursuit

A feasible path from starting position to the goal position is obtained using PRM. Subsequently, the robot can follow the way-points of the path using the pure-pursuit algorithm [43].

The pure pursuit algorithm calculates the required curvature that can lead the robot from its current position to the goal position.

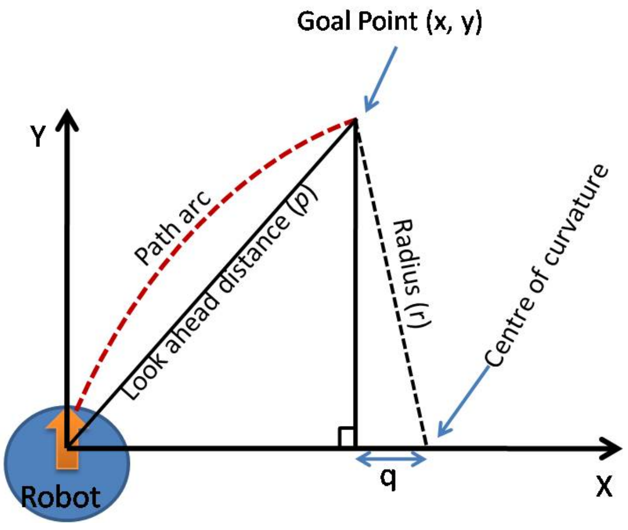


Figure 3.7.: Geometry of the pure pursuit algorithm.

The geometry of the pure pursuit algorithm is given in Fig. 3.7. Using Fig. 3.7, the relation between p, q, r, x, and y can be expressed as (3.7):

$$\begin{aligned} p^2 &= x^2 + y^2, \\ q &= r - x, \\ r^2 &= q^2 + y^2, \end{aligned} \tag{3.7}$$

By solving (3.7), the curvature (ρ) of the path arc can be given by (3.8) as follows:

$$\rho = \frac{1}{r} = \frac{2x}{p^2}, \tag{3.8}$$

The curvature of the path arc, obtained in (3.8), determines the steering wheel angle of the robot.

Following the way-points, the robot can be driven from start to goal using the Algorithm 2. Using ROS, there are two types of velocity commands: linear and angular. The linear velocity moves the robot in straight line and the angular velocity is used to turn the robot. The linear velocity of the controller is taken as 0.4 meter per second. The start and goal positions of the robot in the *corridor.world* are taken as (0.0107, 0.0000) and (0.7562, 3.0553), respectively. The initial orientation of the robot is taken as zero. Own publication related to Section 3.2 is given in [NK-100].

Algorithm 2 The path pursuit algorithm

```
1: while (Goal not found) do
2:     Receive the current pose of the robot in the actual environment:
       $robot\_Pose\_actual = [position, orientation]$;
3:     Convert the coordinates of the position in the actual world to the coordinates in
       occupancy grid using the formula given in Section 3.2.1. Let us say the position
       in occupancy grid as $position\_o$.
4:     Compute the robot pose in occupancy grid:
       $robot\_Pose = [position\_o, orientation]$;
5:     Compute the Linear and angular velocities ($v$ and $\omega$) using the controller:
       $[v, \omega] = controller(robot\_Pose)$;
6:     Send these velocities $[v, \omega]$ to the robot.
7: end while
```

where,

$position = (x, y)$ coordinate word in the actual world.

$orientation =$ Euler angle of the robot

$v =$ Linear velocity to drive the robot

$\omega =$ Angular velocity to drive the robot

3.2.4. Summary

The PRM of the map of *corridor.world* of Turtlebot-Gazebo simulator is tuned up for the number of nodes taken in PRM and the connection distance between the nodes in it. The navigation algorithm is applied to the robot in a convenient way using *Robotics System Toolbox* of MATLAB.

The limitation of this work is that it is applicable to the environment with static obstacles. For future research work, there is a scope to extend this work for the dynamic environment with moving obstacles.

4. Robot navigation with obstacle avoidance in unknown and dynamic environment

4.1. Robot obstacle avoidance using bumper event

This section presents an algorithm for robot-navigation using bumper-event of the robot. This algorithm is implemented on Turtlebot robot in the Gazebo simulator and on a real Turtlebot robot. The bumper and state fields of robot's bumper event are studied for the different command velocities. The experimental results are shown by Mat plots using ROS tool *rqt*. In this section, it is assumed that the bump sensor is the only sensor to handle the obstacles.

The rest of the Section is organized as follows: In Section 4.1.1, the proposed algorithm is given. Experimental verification and results are presented in Section 4.1.2. Section 4.1.3 is for the summary of the section.

4.1.1. Bumper-event based algorithm

The Turtlebot mobile base (Kobuki base) is equipped with a bump sensor. When the bump sensor is hit or pressed by some object, the bumper event is generated. There are two fields in bumper event: *bumper* and *state*. Bumper field can have any value from LEFT(0), CENTER(1) or RIGHT(2) depending upon the corresponding bump sensor that has been pressed. The state field can take the values RELEASED(0) or PRESSED(1). In ROS, there are two types of velocity commands: linear and angular. The linear velocity moves the robot in straight line and the angular velocity is used to turn the robot. To minimize the complexity, if the robot does not have any sensor, except the bump sensor, to avoid the obstacles then the Turtlebot will collide the obstacles in its way. In this situation, the robot must recover from the collision and move to the other possible way. For this purpose, Fig. 4.1 presents the proposed algorithm by a flowchart.

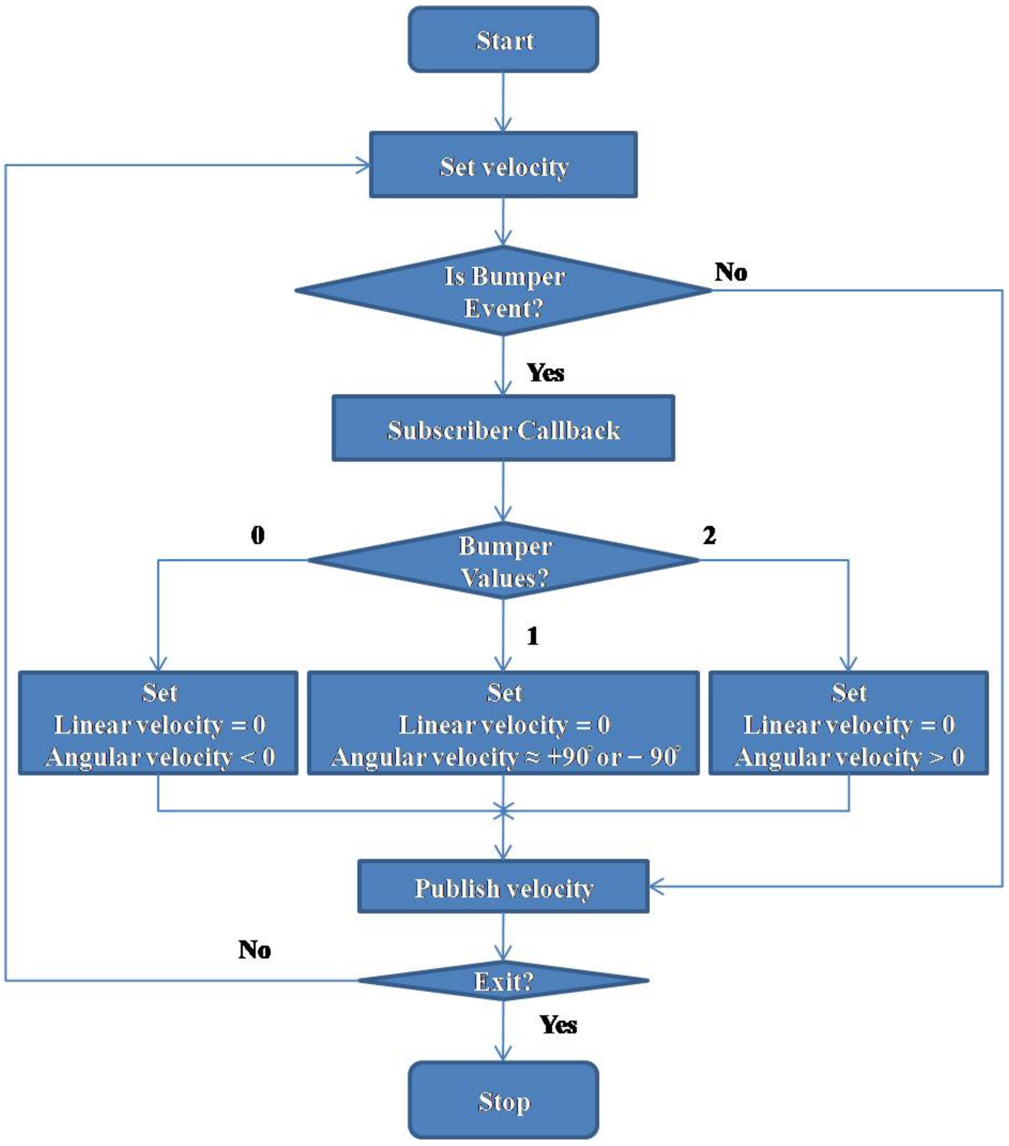

Start
Set velocity
Is Bumper Event?
No
Yes
Subscriber Callback
Bumper Values?
0
1
2
Set Linear velocity = 0 Angular velocity < 0
Set Linear velocity = 0 Angular velocity ≈ +90° or − 90°
Set Linear velocity = 0 Angular velocity > 0
Publish velocity
Exit?
No
Yes
Stop

Figure 4.1.: Flowchart of the proposed algorithm.

4.1.2. Implementation and experimental results

The x, y and z components of linear and angular velocities are taken as in Algorithm 3. In Algorithm 3, the measurement units of linear and angular velocities are meter and radian, respectively.

For Simulation in Gazebo, a simple maze (only boundaries) is constructed in Gazebo environment by inserting four jersey barriers and a Turtlebot as shown in Fig. 4.2. The proposed algorithm is tested in this maze.

Algorithm 3 The x, y and z components of linear and angular velocities

```
 1: if default velocity then
 2:     $linear.x = 0.1,\ linear.y = 0,\ linear.z = 0;$
 3:     $angular.z = 0,\ angular.x = 0,\ angular.y = 0;$
 4: else
 5:     if bumper 0 hit then
 6:         $linear.x = 0,\ linear.y = 0,\ linear.z = 0;$
 7:         $angular.z = -0.35,\ angular.x = 0,\ angular.y = 0;$
 8:     else if bumper 1 hit then
 9:         $linear.x = -0.25,\ linear.y = 0,\ linear.z = 0;$
10:         $angular.z = -1.25,\ angular.x = 0,\ angular.y = 0;$
11:     else if bumper 2 hit then
12:         $linear.x = 0,\ linear.y = 0,\ linear.z = 0;$
13:         $angular.z = 0.35,\ angular.x = 0,\ angular.y = 0;$
14:     end if
15: end if
```

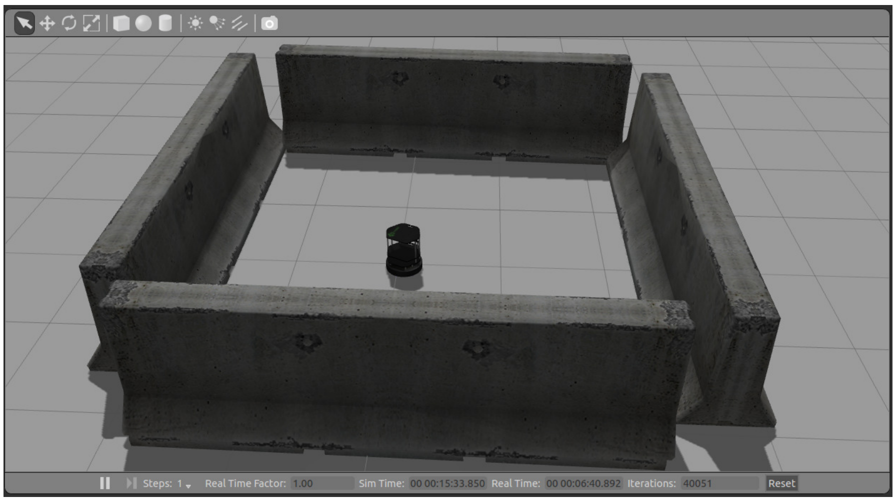


Figure 4.2.: Introductory, simple maze developed in Gazebo simulator.

The bumper hit events for right and centre bumper are shown in Fig. 4.3 . Further, Fig. 4.4 shows the left and center bumper hit events. The variables used in Fig. 4.3−4.4 are described as follows:

'*/mobile_base/events/bumper/bumper*' represents bumper field value ('0','1','2' for left, center and right bumper, respectively) of bump sensor of the robot. The information about the bumper field is available in ROS on the ROS-topic '*/mobile_base/events/bumper*'. Similarly, '*/mobile_base/events/bumper/state*' gives state field value ('0' for released and '1' for pressed) available through ROS-topic '*/mobile_base/events/bumper*'.

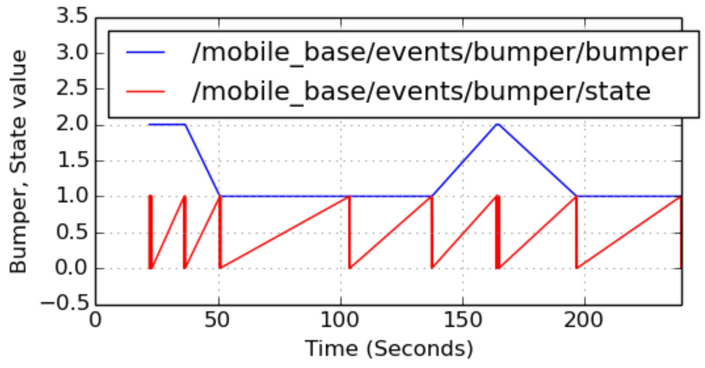


Figure 4.3.: Right and centre bumper hit events.

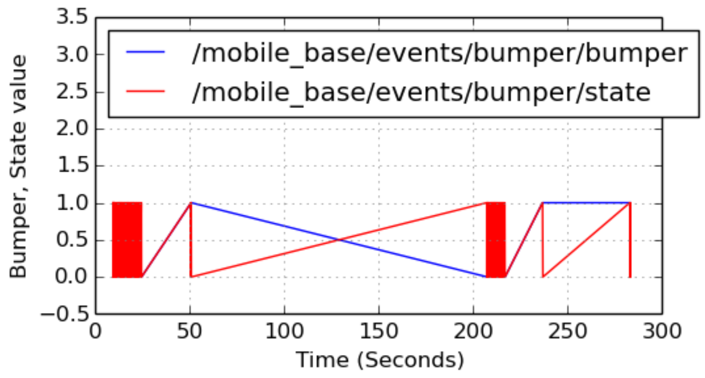


Figure 4.4.: Left and centre bumper hit events.

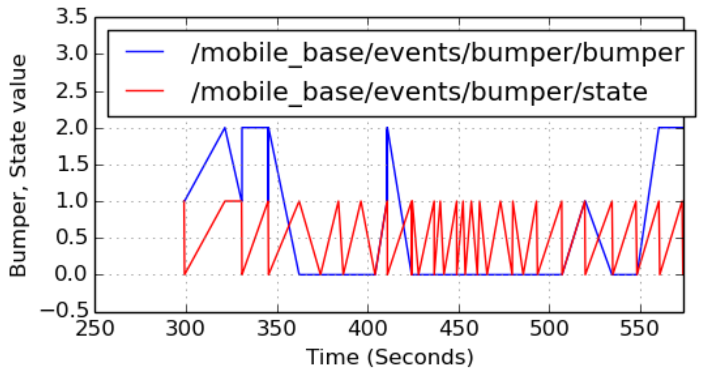


Figure 4.5.: Bumper hit events by real Turtlebot.

Graphical representation and execution of ROS nodes are explained in Appendix A.4. The implementation of the algorithm with real Turtlebot is similar to the case of Gazebo. In case of real robot, Fig. 4.5 presents the bumper events generated with left, right and center bumper values. Variables description for Fig. 4.5 is same as given for Fig. 4.3−4.4. Own publication concerned to Section 4.1 is presented in [NK-101].

4.1.3. Summary

The implementation of the proposed algorithm shows that the obstacle avoidance task can be handled by using the bumper events of the Turtlebot. The Turtlebot successfully recovers from the collisions and follows the new velocity commands. In the case of autonomous navigation in unknown environment, this algorithm is very useful when other sensors for obstacle avoidance are got damaged or removed to minimize the complexity. The main disadvantage of the proposed algorithm is that it does not lead to a collision-free navigation. Therefore, the other sensors of robot like LASER and camera may get priority over bump sensor for collision-free navigation. Consequently, the methods based on the sensors which lead to obstacle avoidance without collisions during navigation can be taken into consideration for the future work.

4.2. Robot navigation in unknown environment using fuzzy logic

In our everyday life we often solve control tasks (i.e. driving a car, controlling the heating system at home, etc.) without having exact numerical values that are replaced by qualitative concepts as “very small”, “small”, “approximately zero”, “big”, “very big”, etc. Following the early ideas on three valued logic developed by Łukasiewicz in the twenties of the past century [102, 103], it was Lotfi Zadeh who constructed the mathematically rigorous formulation of the multiple valued logic as fuzzy sets in 1965 [88]. In this formulation the “characteristic function” of a classical subset of a set was replaced with a membership function that had its value in the interval $[0, 1]$ The generalization of the “AND” and “OR” operations with classical sets were also elaborated for fuzzy sets so the control rules (i.e. the recommended actions to be done under certain conditions) formulated by the use of a common human language were “translated” to the “language” of fuzzy controllers. Such systems contain a lot of arbitrary functions (from the simple triangular and trapezoidal systems to Dombi’s “Pliant System” [91, 104]). In many applications (due to their parametric simplicity) triangular and trapezoidal membership functions are used (e.g. [105–107]).

In the practice it often occurs that for the development of the models of real physical systems no “closed analytical formulation” (i.e. the combination of finite number of certain functions that are available in the form of classical tables and integral tables) is possible. In such cases the use of “universal approximators” is expedient that means the application of special sequences and series that can be “cut” at finite number of terms to achieve some limited precision. The earliest example of function approximations was given by Weierstraß in 1885 in [108] for the approximation of continuous real valued functions by polynomials. It was generalized by Stone in 1948 in [109] for other functions than polynomials. This method has obtained application in fuzzy systems in the nineties of the past century (e.g. [110]).

Regarding the approximation of multiple variable continuous functions, as a rebuttal of one of Hilbert’s conjectures published in [111], at first Arnold gave a particular example for three variable functions in 1957 [112]. In the same year Kolmogorov published a proof on the possible approximation of continuous functions of arbitrary number of real variables with single variable real functions in [113]. Kolmogorov’s constructive example has been refined and simplified in the sixties of the past century by Sprecher [114] and Lorentz [115, 116]. The neural networks were found as technical realizations of these function approximators (e.g. [117, 118]).

The fuzzy systems were found to be universal approximators, too (e.g. [78, 79]) in which their "scalability" problems (i.e. the strong dependence of the necessary functions on the number of the independent variables and the necessary precision) were realized, too (e.g. [80, 119, 120]).

In general, their practical applicability combined with their universal approximator abilities make the fuzzy and neurofuzzy systems popular in practical applications as robot navigation.

In the modern software products as in MATLAB's Fuzzy Logic Toolbox there are conveniently usable "built in" membership function types the parameters of which easily can be edited within these systems. So the use of the available "ready" possibilities was plausible for my purposes. In the 1^{st} step, a simple, primitive, bumpers-based system was developed in Section 4.1. In the next step, the bumpers-based system is replaced with a Mamdani-type FIS. The parameters of the Mamdani-type FIS are not set by "sophisticated methods". After showing that these "preliminary solutions" are able to work, a sophisticated ANFIS system is developed with Sugeno-type output in which the parameters are set by a learning process.

In this section, a model for the robot navigation in unknown environment is presented using MATLAB-Simulink. The robot navigation is handled by two controllers: pure pursuit and fuzzy logic controller. The pure pursuit controller computes a direct path from start to goal position without considering the obstacles in the path. For obstacle avoidance in robot navigation, the fuzzy logic controller is taken. This fuzzy logic controller takes the input from the LASER sensor of the robot and gives the change in the angular velocity as output to the robot to avoid the obstacle. The navigation paths resulting from the proposed model, with and without obstacles in the paths, are shown in figures.

The remaining of this section is organized as follows: Section 4.2.1 presents an insight into using the LASER scan data for obstacle avoidance. The proposed fuzzy controller is described in Section 4.2.2. Section 4.2.3 presents description about the implementation of the model in MATLAB-Simulink. The experimental set-up and result are discussed in Section 4.2.4. Finally, Section 4.2.5 is the summary of the section.

4.2.1. LASER scan data for obstacle avoidance

LASER scan is one of the most popular method to compute the distances of obstacles in the robot navigation. The LASER sensor messages give ranges of obstacles. The obstacle ranges and corresponding angles of LASER rays give the locations of the obstacles in the robot path. A single LASER scan message may contain some hundred ranges from right to left in LASER scan area starting from the minimum scan angle in the right side to the maximum scan angle in the left side of the central LASER ray. For example, in the Turtlebot robot, a LASER scan message may contain 640 ranges from right to left. Consequently, there is equal number of scan angles for these ranges. Fig. 4.6 describes the different locations of the obstacles coming in the way of three different LASER rays.

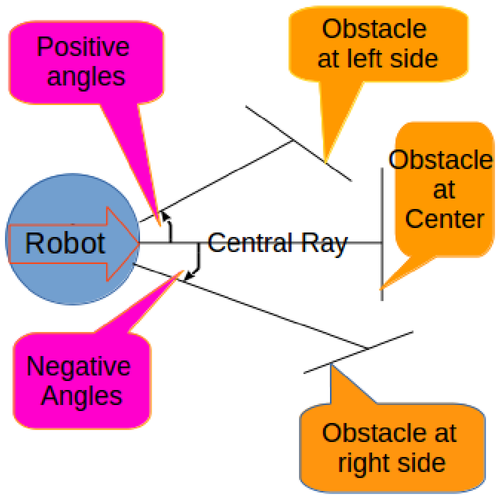


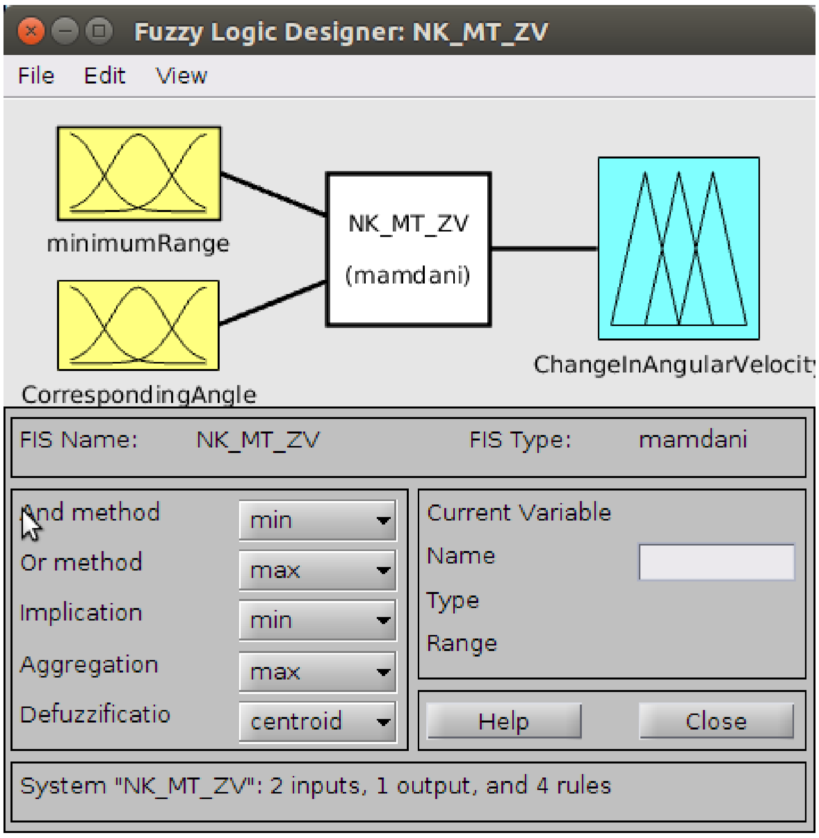
Figure 4.6.: Positions of obstacles in LASER scan area.

4.2.2. The fuzzy controller

Using the obstacle ranges and corresponding angles, the proposed fuzzy controller turns the robot in the required direction to avoid the obstacles during the navigation. The overall fuzzy logic design, membership functions for the fuzzy controller and the required fuzzy rule base are defined as follows.

4.2.2.1. Fuzzy logic design

The robot publishes LASER scan messages on a topic of ROS. The robot navigation model programmatically subscribes the LASER scan messages available on that topic of ROS. In this way, the LASER scan messages, published by the robot, are subscribed by the robot navigation model in MATLAB-Simulink. These LASER scan messages are used to detect the distances and locations of the obstacles in the robot navigation environment. A LASER scan message contains a vector of ranges obtained by the LASER sensor of the robot. A range is the distance of an obstacle at a particular angle of LASER scan. Therefore, there is a corresponding angle for each of the range present in the ranges vector. The first range of the ranges vector belongs to the minimum angle of the LASER scan. The other angles corresponding to the ranges can be calculated by using the specified angle increment of the LASER sensor of the robot.



FIS Name: NK_MT_ZV FIS Type: mamdani

And method: min
Or method: max
Implication: min
Aggregation: max
Defuzzificatio: centroid

System "NK_MT_ZV": 2 inputs, 1 output, and 4 rules

Figure 4.7.: Fuzzy logic design.

The ranges and angles vectors provide the information of distance and scan angle of obstacles. Further, the minimum range and its corresponding angle account the closest obstacles to the robot. If more than one minimum values are present in the ranges vector then the first minimum value is considered. The function to compute the minimum range and its corresponding angle is defined in Fig. 4.13. In this manner, the minimum range and its corresponding angle are provided to the fuzzy controller as the two inputs. As the only output, the fuzzy controller is providing the change in the angular velocity of the robot. Fig. 4.7 presents the proposed fuzzy logic design of the fuzzy controller.

4.2.2.2. Membership functions

For the given FIS, the π-shaped membership functions are used because the fuzzy concepts like 'around', 'approximately', 'close to', are present.

The membership function for the *minimumRange* input variable of the fuzzy controller is divided into two categories named as *close* and *normal*. The category 'close' describes that the obstacle is close to the robot. In this situation, the robot needs the required change in its angular velocity so that it can avoid the obstacle in its navigation path from start to goal. The membership function plots for the input variable *minimumRange* is given in Fig. 4.8.

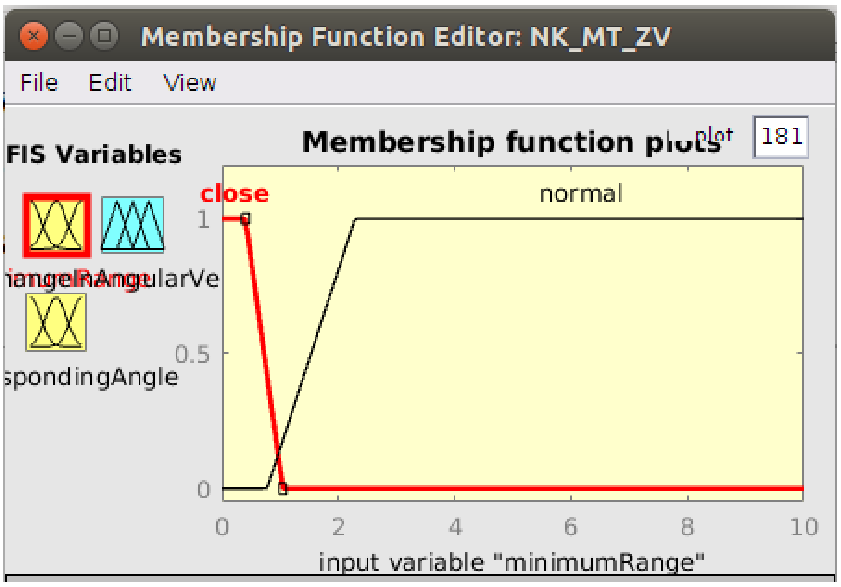

Membership Function Editor: NK_MT_ZV
File
Edit
View
FIS Variables
Membership function plots
plot
181
close
normal
iangeRAngularVe
spondingAngle
1
0.5
0
0
2
4
6
8
10
input variable "minimumRange"

Figure 4.8.: Membership function plots for *minimumRange*.

The membership functions taken in Fig. 4.8 are defined using (4.1).

$$\mu(x) = \begin{cases} 0, & \text{for } x \le p \\ \frac{x-p}{a-p}, & \text{for } p < x \le a \\ 1, & \text{for } a < x \le b \\ \frac{q-x}{q-b}, & \text{for } b < x \le q \\ 0 & \text{for } q < x \end{cases} \tag{4.1}$$

where, the set of parameters $\{p, a, b, q\}$ is same as taken in (2.6).

In Fig. 4.8, the sets of parameter values for the membership functions of *close* and *nor-*

mal are taken as $\{-3.43, -0.216, 0.404, 1.319\}$ and $\{1.124, 2.29, 12.5, 18.3\}$, respectively.

The second input variable of the fuzzy controller is *CorrespondingAngle*. This angle gives the information that whether the obstacle is at the right side, left side or at the central position in respect to the robot. Based on this, there are three categories for *CorrespondingAngle*, namely: *rightSide*, *center* and *leftSide*. Fig. 4.9 represents the membership function plots for the input variable *CorrespondingAngle*.

In Fig. 4.9, the membership function for *center* is defined using (4.2). The membership functions for the remaining two variable in Fig. 4.9 are defined using (4.1).

$$\mu(x) = \begin{cases} 0, & \text{for } x \leq p \\ \frac{x-p}{a-p}, & \text{for } p < x \leq a \\ \frac{q-x}{q-a}, & \text{for } a < x \leq q \\ 0 & \text{for } q < x \end{cases} \tag{4.2}$$

where, the set of parameters $\{p,\ a,\ q\}$ is same as in (2.5).

In Fig. 4.9, the sets of parameter values for the membership functions of *rightSide*, *center*, and *leftSide* are considered as $\{-0.899, -0.565, -0.473, -0.02533\}$, $\{-0.0554, 0.00592, 0.06534\}$, and $\{0.0455, 0.474, 0.565, 0.899\}$, respectively.

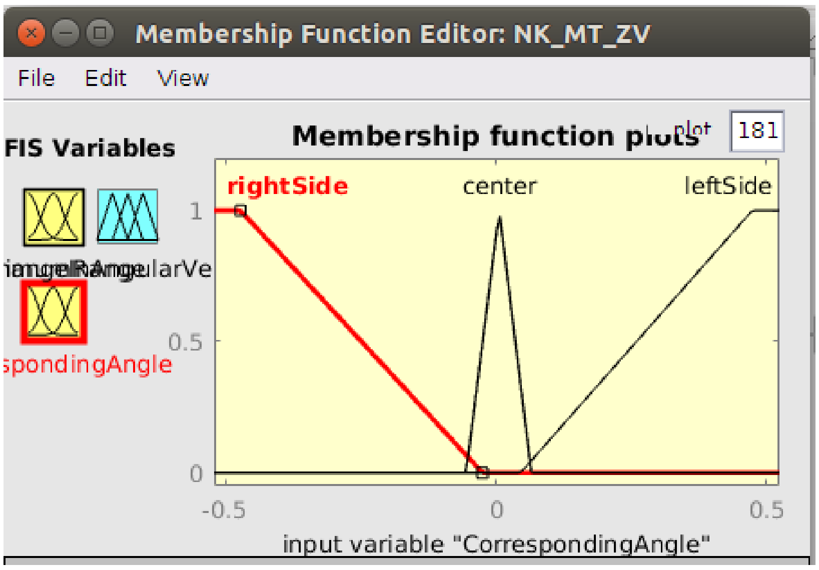


Figure 4.9.: Membership function plots for *CorrespondingAngle*.

The fuzzy controller has a single output variable: *ChangeInAngularVelocity*. The values for this variable are divided into *rightTurn*, *noTurn* and *leftTurn* categories. The membership function plots for the output variable *ChangeInAngularVelocity* are depicted in Fig. 4.10. In Fig. 4.10, the membership function for *noTurn* is given by using (4.2). For *rightTurn* and *leftTurn*, the membership functions are same as presented in (4.1).

In Fig. 4.10, the sets of parameter values for the membership functions of *rightTurn*, *noTurn*, and *leftTurn* are given as $\{-6.02, -3.78, -3.17, -0.1929\}$, $\{-0.35, 0, 0.35\}$, and $\{0.2214, 3.17, 3.78, 6.02\}$, respectively.

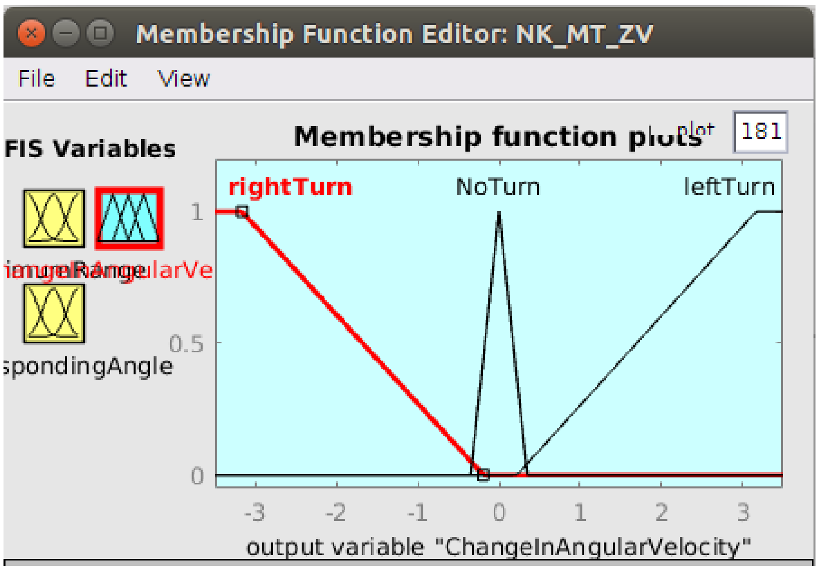
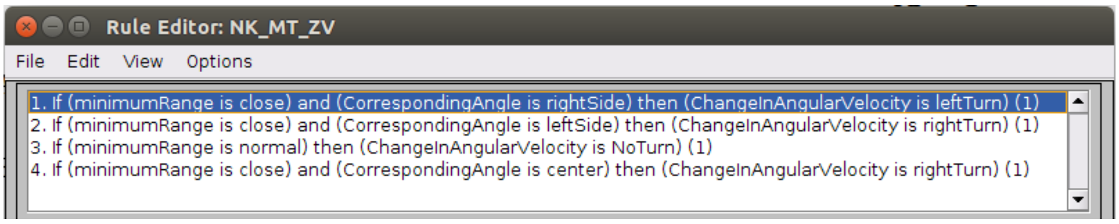

Membership Function Editor: NK_MT_ZV
File
Edit
View
FIS Variables
Membership function plots
plot
181
rightTurn
NoTurn
leftTurn
1
0.5
0
-3
-2
-1
0
1
2
3
output variable "ChangeInAngularVelocity"

Figure 4.10.: Membership function plots for *ChangeInAngularVelocity*.

4.2.2.3. Fuzzy rule base

The fuzzy rules are created on the simple strategy ,like, if the obstacle is close to the left side of the robot's LASER scan area then turn the robot to the opposite side (i.e. right side).



1. If (minimumRange is close) and (CorrespondingAngle is rightSide) then (ChangeInAngularVelocity is leftTurn) (1)
2. If (minimumRange is close) and (CorrespondingAngle is leftSide) then (ChangeInAngularVelocity is rightTurn) (1)
3. If (minimumRange is normal) then (ChangeInAngularVelocity is NoTurn) (1)
4. If (minimumRange is close) and (CorrespondingAngle is center) then (ChangeInAngularVelocity is rightTurn) (1)

Figure 4.11.: Rule base of the fuzzy controller.

On the other hand, if the obstacle is close to the right-side area of LASER scan then the robot turns to the left side. However, if the obstacle is close to the central area of

LASER scan then the fuzzy rule can be left or right biased. For the latter case, the right biased rule has been considered. Fig. 4.11 presents the fuzzy rule base for the fuzzy controller of the system.

4.2.3. Implementation of the model in MATLAB–Simulink

The proposed model for robot navigation is presented by Fig. 4.12 as a model in Simulink. To move the robot from start to goal, two controllers are used in the system. One of the two controllers is pure pursuit controller which is available in the Simulink library. The pure pursuit controller takes two inputs: the current pose of the robot and a $N \times 2$ matrix of way points [37] in the navigation path. The matrix of way points is given in by the constant block of the Simulink library. The pure pursuit block provides three outputs: linear velocity, angular velocity and target direction. Out of the three outputs the later one is optional. Here, the first two outputs are taken from the pure pursuit controller. The pure pursuit controller computes the linear and angular velocity to drive the robot from current position to the goal position without taking the obstacles into consideration. Therefore, an additional controller is required for obstacle avoidance.

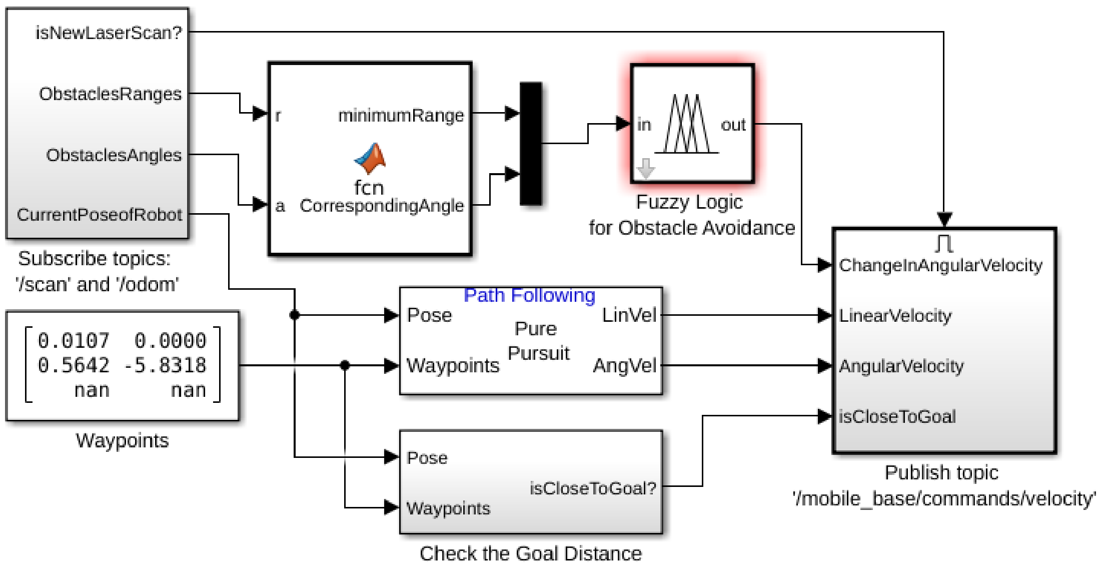
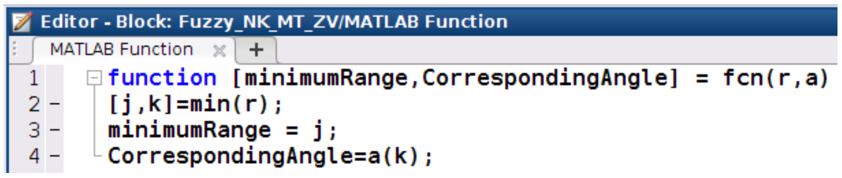


Figure 4.12.: The model of the system in MATLAB-Simulink.

The proposed fuzzy controller is used for the obstacle avoidance purpose. The two inputs, *minimumRange* and *CorrespondingAngle*, for the fuzzy controller are defined in the MATLAB function *fcn* given in the Fig. 4.13. There are two input arguments to the MATLAB function *fcn*. The first parameter (r) is associated with *ObstaclesRanges* vector and the second parameter (a) is linked with *ObstaclesAngles* vector. The fuzzy controller computes the desired change in angular velocity to avoid the obstacle by using the fuzzy membership functions and fuzzy rules defined in its fuzzy rule base. The decision of the nearest obstacle is taken by the function *fcn* (Fig. 4.13). This function uses LASER ranges and their corresponding angles.



```
function [minimumRange,CorrespondingAngle] = fcn(r,a)
[j,k]=min(r);
minimumRange = j;
CorrespondingAngle=a(k);
```

Figure 4.13.: MATLAB function *fcn*.

The program instructions used in Fig. 4.13 are described in Table 4.1.

Table 4.1.: Description of the program instructions used in Fig. 4.13.

Instruction(s)	**Description**
1	(r, a) and (minimumRange, CorrespondingAngle) are input and output parameters, respectively.
2	'min(r)' gives the minimum range (j) and index (k) of the minimum range.
3	The minimum range value (j) is assigned to 'minimumRange'.
4	Angle 'a(k)' for 'j' is assigned to 'CorrespondingAngle'.

Algorithm 4 The robot navigation algorithm

```
1: while (True) do
2:     Subscribe the LASER scan readings of the robot and set start and goal positions
       for it.
3:     Compute the required linear and angular velocities for the robot to navigate it
       from the current position to goal using pure pursuit path following navigation
       algorithm block of Simulink. In addition, compute the change in angular velocity
       using the fuzzy controller to avoid the closest obstacle.
4:     Compute the resultant angular velocity by adding the angular velocity from the
       pure pursuit path following block and change in angular velocity from fuzzy
       controller obtained in step 2.
5:     Publish linear and the resultant angular velocities to navigate the robot.
6: end while
```

At a time, A LASER scan message from the robot gives the distance ranges in the robot view area. The function *fcn* returns the first minimum range and its corresponding angle. Since, the function *fcn* selects the first minimum rage out of the scan message, therefore, the possible ambiguity in selecting the nearest obstacle may be addressed by the method. This first minimum range and its angle are taken as input to the membership function. Using the membership function, the robot need to change its ongoing direction only when any obstacle is close enough in the path.

The process robot navigation, from start to goal, can be summarized as in Algorithm 4.

4.2.4. Experimental set-up and results

To test the method, the default *world* (*playground.world*) of the *turtlebot_gazebo* package of ROS is used with customization of having two spot lights. The *playground.world* contains five different objects which can act as obstacles during robot navigation task. Therefore, these obstacles can play good role for testing the method.

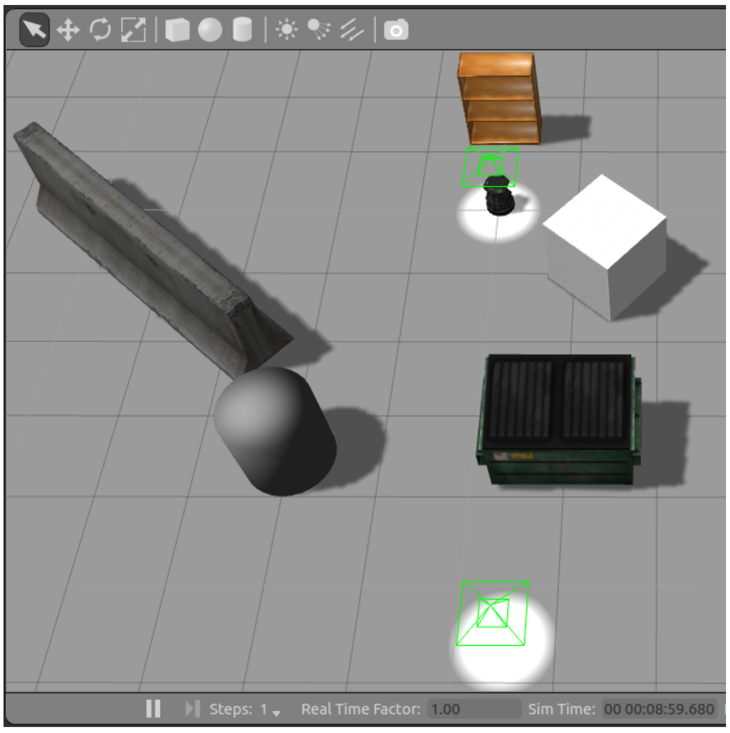


Figure 4.14.: The Gazebo's *playground.world*.

Fig. 4.14 shows the Gazebo-world consisting an instance of each of the five models: a *mobile_base* (turtlebot), a *bookshelf*, a *jersey_barrier*, a *unit_cylinder_1*, a *cube_20k*, a *Dumpster* and a *ground_plane_0*.

The mobile_base is heading towards positive X-axis direction. Two spot lights are taken in the figure to show the start and goal positions of robot navigation. The first spot light (circumscribing mobile base) from the top consists of the start point and the second spot light towards bottom contains the goal position of the navigation path. The coordinates of the start and goal points supplied to the system are $(0.0107, 0.0000)$ and $(0.5642, -5.8318)$, respectively.

The execution of the model, as given in Fig. 4.12, on the Gazebo-world taken in Fig. 4.14, drives the robot from start to goal by avoiding the first obstacle, a *cube_20k*, on the right side and then avoiding the second obstacle, a *Dumpster*, on its left side during the navigation from the start to goal.

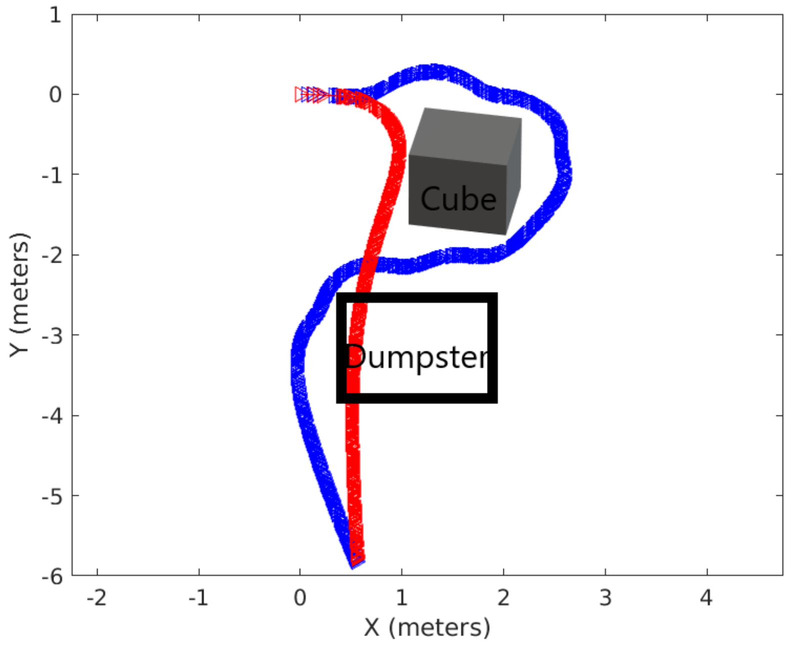


Figure 4.15.: Robot paths with and without obstacles. The blue path is observed in the presence of the obstacles and the red path is received in the absence of the obstacles.

The fuzzy controller provides the left turn in the presence of cube on the left side and the required right turn to avoid the *Dumpster* on left. This navigation path is shown by blue color in the Fig. 4.15. If the two obstacles in the navigation path, a *cube_20k* and a *Dumpster*, in the Gazebo world as given in Fig. 4.14, are deleted then there is no obstacle in the navigation path of the robot from start to goal. In this case, the fuzzy controller gives the change in angular velocity near to zero and therefore the pure pursuit controller drives the robot towards the goal directly as shown by the red path in the

Fig. 4.15. Own publications pertaining to Section 4.2 are given by [NK-121, NK-122].

4.2.5. Summary

The proposed fuzzy controller provides the required change in angular velocity to avoid obstacles on each side of LASER scan area. Further, it can be noted from the Fig. 4.15 that the turns are smooth. Therefore, there are very little chance to stuck at the turns in the robot navigation path on either side. Further, the proposed system does not need any information about the obstacles before starting the navigation from start to goal. Hence, the system is suitable for the navigation in unknown environment where the obstacles are not known in advance or the positions of the obstacles are changing with time.

The given fuzzy controller is right turn biased for the obstacles present on the centre of LASER scan area. Future research work may remove the left turn or right turn biasing from the fuzzy controller.

4.3. Robot navigation with obstacle avoidance in unknown environment using adaptive neuro-fuzzy inference system

In this section, a robot navigation model is constructed in MATLAB-Simulink. This robot navigation model makes the robot capable for the obstacles avoidance in unknown environment. The navigation model presented in this section is based on the navigation model proposed in Section 4.2.3. Using the Mamdani-type FIS of Section 4.2.2, training data, of input and output mapping, is collected. This training data is supplied to the ANFIS to obtain a Sugeno-type FIS. The navigation model, using the Sugeno-type FIS, is implemented on the simulated as well as real robot.

The remaining part of this section is organized as follows: The model for the robot navigation is presented in Section 4.3.1. Section 4.3.2 describes the FIS for the fuzzy controller taken in the model. Results are given in Section 4.3.3. Summary of the section is presented in Section 4.3.4.

4.3.1. Robot navigation model in MATLAB-Simulink

The robot navigation model is presented using MATLAB-Simulink software. In this model, the navigation task is accomplished by the following two controllers:

(i) The pure pursuit controller,

(ii) The fuzzy based controller (as described in Section 4.2).

Fig. 4.16 describes the details of the presented robot navigation model. This model contains four subsystems as follows:

(i) Subscribing topic: '/odom',

(ii) Subscribing topic: '/scan',

(iii) Goal Distance Checking,

(iv) Publishing topic: 'mobile_base/commands/velocity'.

The *Subscribing topic: '/odom'* subsystem, in the given model, subscribes the */odom* topic of the ROS to obtain current pose of the robot. A typical message vector on */scan* topic contains distance ranges of obstacles in the scan area. In two dimensional state, the scan area is of 'V' shape. From right to left, a single LASER scan may contain some hundred ranges. Using minimum angle of scan and angle increment, the corresponding angle to each range in a message can be find out. The obstacle ranges and their corresponding angles of scan, together, give the locations of the obstacles in the robot path. The *Subscribing topic: '/scan'* subsystem is subscribing the ROS topic */scan* to get the vectors of ranges.

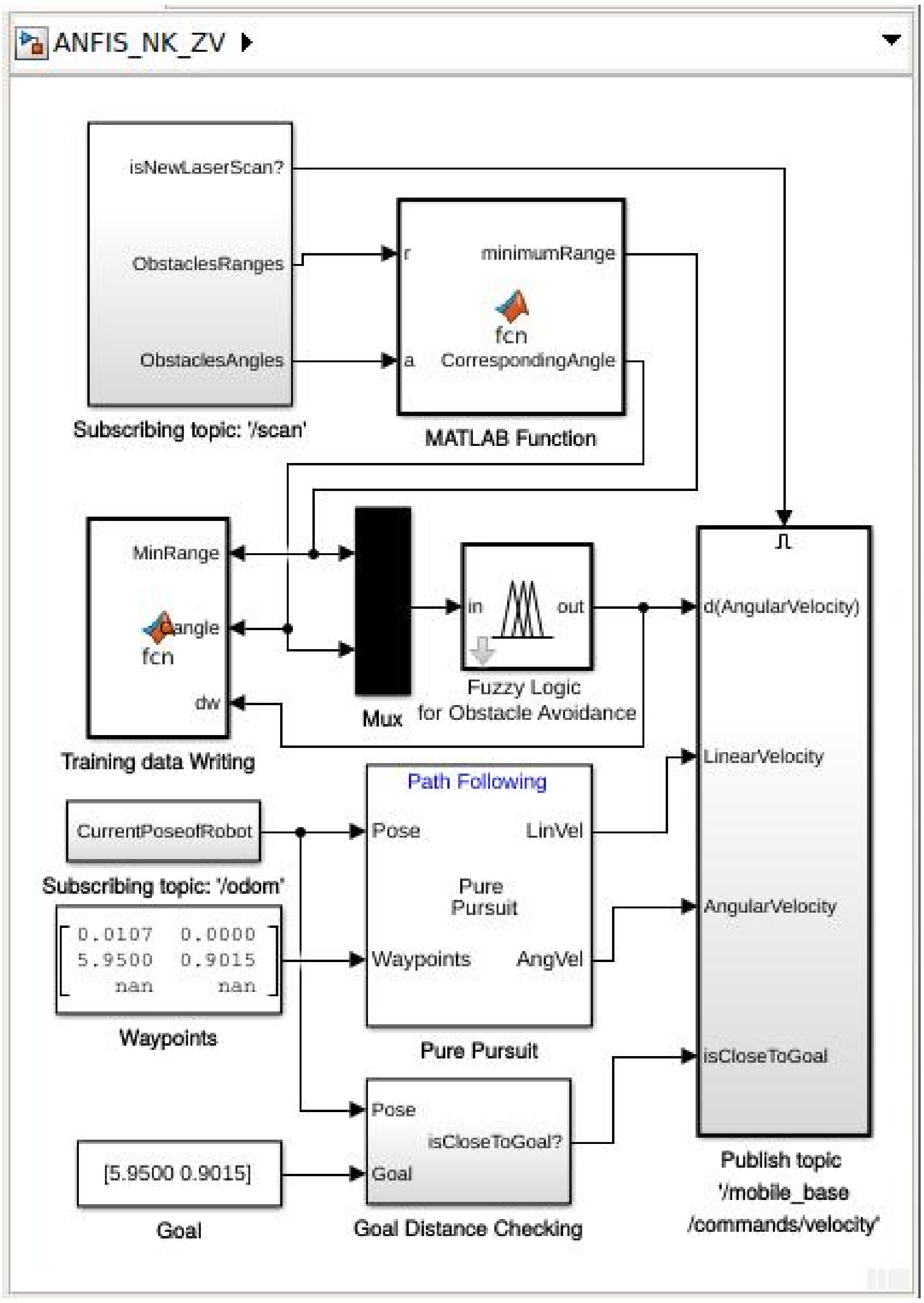


Figure 4.16.: Robot navigation model in Simulink.

The goal distance (p) between the goal position (x, y) and current position (x', y') of the robot can be expressed by (4.3) as follows:

$$p = \sqrt{(x - x')^2 + (y - y')^2}. \tag{4.3}$$

Before sending the new velocity commands to the robot, the *Goal Distance Checking* subsystem checks the distance of goal from the current position of the robot. If the distance between the goal and the robot is less than or equal to the given value (in our case, it is 0.1 meter) then the new velocity commands will not be given to robot. As a result, the robot navigation process gets completed and, therefore, terminated. Robot is subscribing the topic */mobile_base/commands/velocity* of ROS which is being published by the subsystem *Publishing topic: '/mobile_base/commands/velocity'*.

In addition to the four subsystems, the remaining blocks of the model are as follows:

(i) two constant blocks named as *Waypoints* and *Goal*,

(ii) a *Mux* block,

(iii) a *Pure Pursuit* path following algorithm block,

(iv) two MATLAB functions,

(v) a *fuzzy logic for obstacle avoidance* block.

The *Waypoints* constant block is to contain a $N \times 2$ matrix of way points of navigation path. In case of unknown environment, it is considered that only start and goal positions of the robot are known, initially. The *Goal* constant block contains a two dimensional coordinates of goal position. The *Mux* block is combining the two inputs (*minimumRange* and *CorrespondingAngle* received from the *MATLAB Function* block) in a single output link. So that, these two input values can be passed to the fuzzy controller block by a single link. The *Pure Pursuit* block is used to compute the required linear and angular velocities to drive the robot using inputs from way-points of the path and current pose of the robot. Since the navigation environment is considered as unknown, the way-points of the feasible path are unknown except the two points: start and goal. The function *fcn* in *MATLAB Function* block computes the minimum range and its corresponding angle of scan from the vectors of *ObstaclesRanges* and *ObstaclesAngles* received from *Subscribing topic: '/scan'* subsystem block. The MATLAB function in the *Training data writing* block is used to write the data in three separate columns for minimum range, its corresponding angle and the output from the fuzzy controller. This training data can be used to train a FIS using ANFIS model. The block named *Fuzzy Logic for Obstacle Avoidance* is a fuzzy logic controller block.

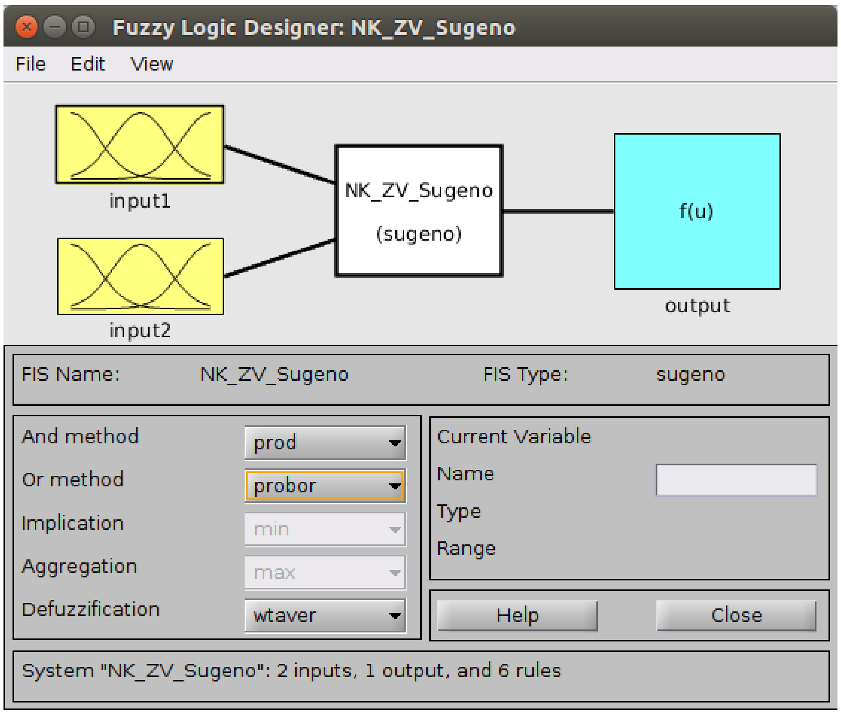
The published linear velocity is same as linear velocity given by the pure pursuit block. Since the pure pursuit block does not consider the obstacles in the path, the angular velocity is needed to be adjusted before its publication on the ROS. This required adjustment in the angular velocity is obtained by the *Fuzzy Logic for Obstacle Avoidance* block. The published angular velocity (ω) can be stated by (4.4) as follows:

$$\omega = \omega_1 + \omega_2. \tag{4.4}$$

where, ω_1 is the angular velocity (i.e. *AngVel*) provided by path pursuit controller. ω_2 is the output (i.e. *d(AngularVelocity)*) of the fuzzy logic controller.

4.3.2. Sugeno-type FIS for the Fuzzy Controller

The training data, received from the execution of the navigation model with Mamdani-type FIS in its fuzzy controller, are given as input to ANFIS to find the Sugeno-type FIS. The received Sugeno-type FIS is shown in Fig. 4.17.



FIS Name: NK_ZV_Sugeno FIS Type: sugeno

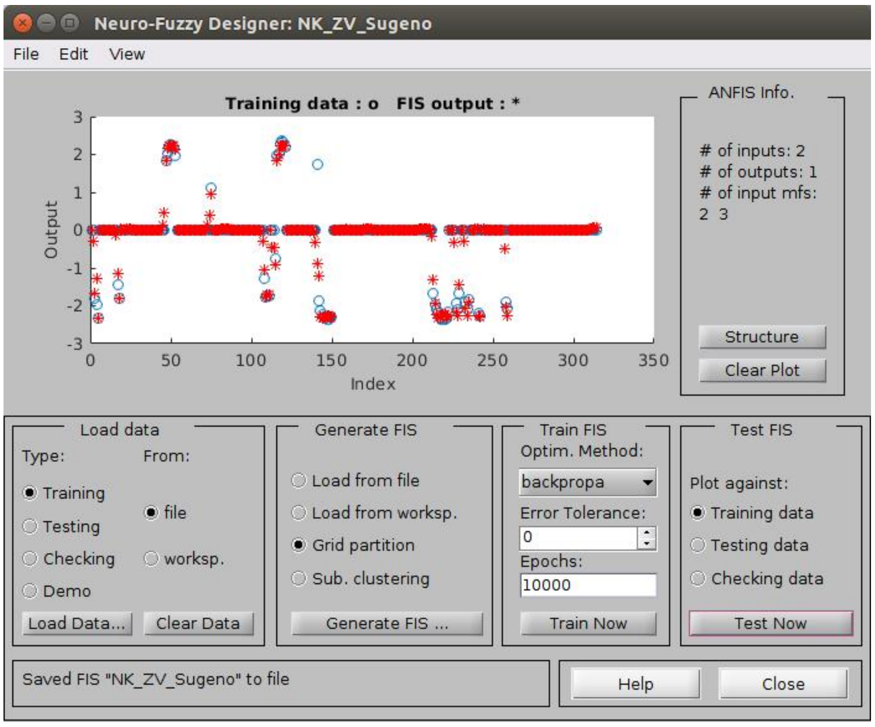
And method: prod
Or method: probor
Implication: min
Aggregation: max
Defuzzification: wtaver

Current Variable
Name
Type
Range

System "NK_ZV_Sugeno": 2 inputs, 1 output, and 6 rules

Figure 4.17.: Sugeno-type FIS obtained using ANFIS.

The '*and*' and '*or*', the rule connecting operators, are taken as product and probabilistic-OR (algebraic sum), respectively. The defuzzification method is *weighted average* method as defined in (2.14). Fig. 4.18 depicts the plot between the training data and FIS output.



# of inputs: 2
# of outputs: 1
# of input mfs:
2 3

Type: [✓ Training] Testing Checking Demo
From: [✓ file] worksp.

Load from file / Load from worksp. / [✓ Grid partition] / Sub. clustering

Optim. Method: backpropa
Error Tolerance: 0
Epochs: 10000

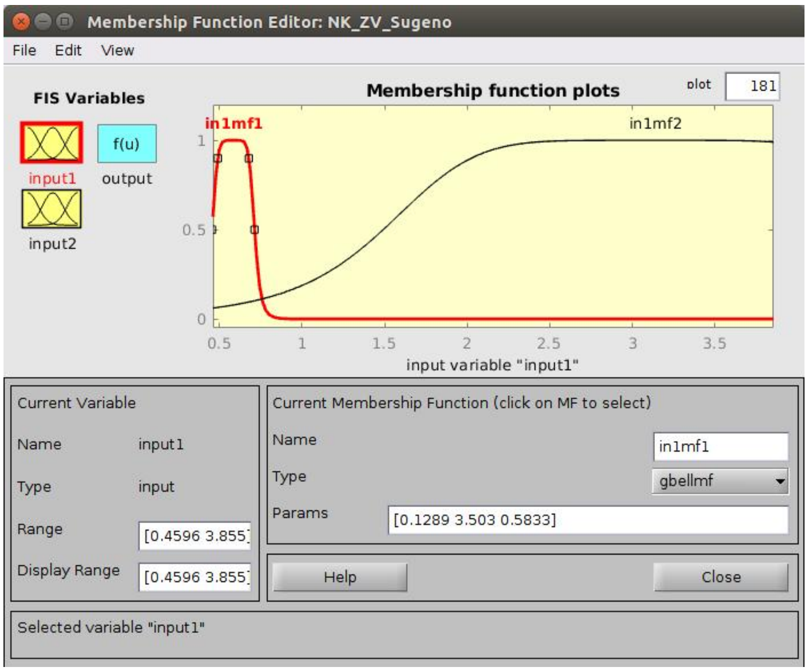
Plot against: [✓ Training data] Testing data Checking data

Saved FIS "NK_ZV_Sugeno" to file

Figure 4.18.: Training data and FIS output in neuro-fuzzy designer.

After training, the number of rules and output membership functions are six each. It is observed that for the given number of input variables, output variable(s) and input membership functions for each input variable, the ANFIS has generated input membership functions, rules and output functions.

Fig. 4.19–4.20 presents the resultant input membership functions. The input variables, *input1* and *input2*, are auto generated names after the training through ANFIS and correspond to the input variables of Mamdani-type FIS presented in Fig. 4.7. Similarly, the output variable, *output*, is corresponding to the output variable of the Mamdani-type FIS (given by Fig. 4.7). It can be observed from the Fig. 4.19 and Fig. 4.20 that the type of the membership functions is generalized bell membership function as defined in (2.7). Here, the sets of parameters $\{w,\ s,\ c\}$ for the membership functions of *in1mf1*, *in1mf2*, *in2mf1*, *in2mf2*, *in2mf3* are received as $\{0.1289, 3.503, 0.5833\}$, $\{1.644, 2.8, 3.134\}$, $\{0.1479, 2.086, -0.3496\}$, $\{0.03622, 2.272, 0.004019\}$, $\{0.09114, 2.076, 0.3323\}$, respectively.



Current Variable

Name input1

Type input

Range [0.4596 3.855]

Display Range [0.4596 3.855]

Current Membership Function (click on MF to select)

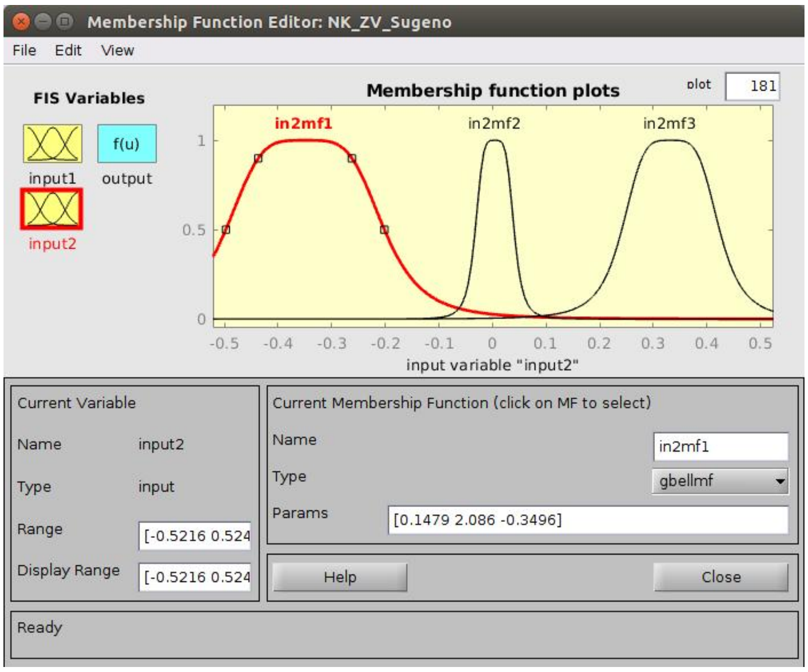
Name in1mf1

Type gbellmf

Params [0.1289 3.503 0.5833]

Selected variable "input1"

Figure 4.19.: Membership function plots for *input1* in Sugeno-type FIS.



Current Variable

Name input2

Type input

Range [-0.5216 0.524

Display Range [-0.5216 0.524

Current Membership Function (click on MF to select)

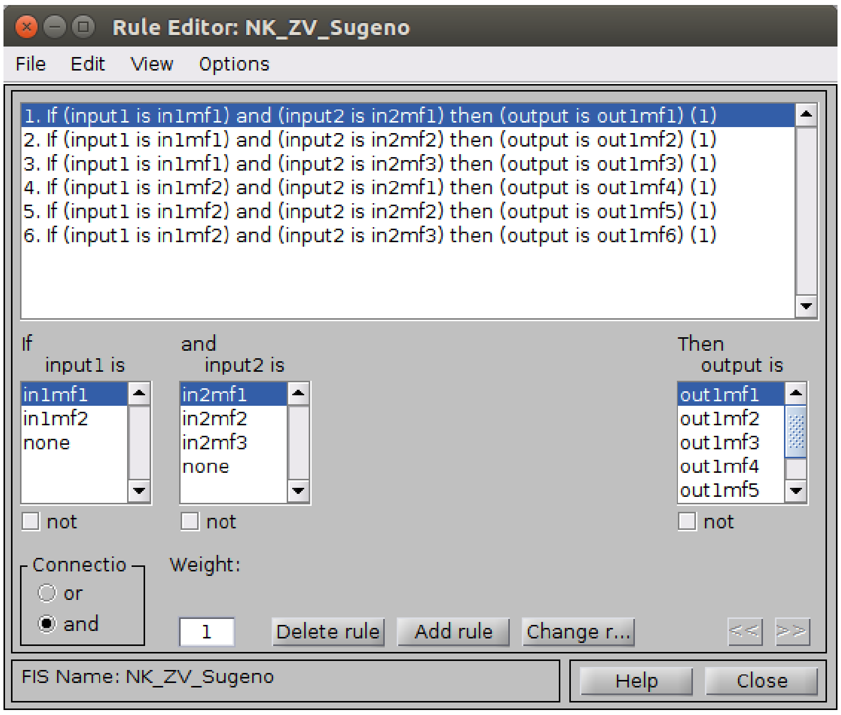
Name in2mf1

Type gbellmf

Params [0.1479 2.086 -0.3496]

Ready

Figure 4.20.: Membership function plots for *input2* in Sugeno-type FIS.



1. If (input1 is in1mf1) and (input2 is in2mf1) then (output is out1mf1) (1)
2. If (input1 is in1mf1) and (input2 is in2mf2) then (output is out1mf2) (1)
3. If (input1 is in1mf1) and (input2 is in2mf3) then (output is out1mf3) (1)
4. If (input1 is in1mf2) and (input2 is in2mf1) then (output is out1mf4) (1)
5. If (input1 is in1mf2) and (input2 is in2mf2) then (output is out1mf5) (1)
6. If (input1 is in1mf2) and (input2 is in2mf3) then (output is out1mf6) (1)

Figure 4.21.: Sugeno-type fuzzy rules received using ANFIS.

Fig. 4.21 presents the rule editor of the Sugeno FIS obtained. The function of the only output variable in the Sugeno-type FIS is defined using (2.16). In our study, zero-order Sugeno-type FIS is obtained. For rules from 1 to 6 (in Fig. 4.21), the values of the output are observed as $2.405, -2.303, -2.528, -0.005126, 0.04744, 0.01124$, respectively.

The rules in the Fig. 4.21 are auto generated rules after the successful training from the ANFIS model. It can be noted that the number of rules of this Sugeno-type FIS is different from the number of rules used in the Mamdani-type FIS given by Fig. 4.7.

4.3.3. Results

The robot navigation model (as given in Section 4.3.1) is tested on the simulated world shown by Fig. 4.22. This simulated world has been constructed by using the Gazebo simulator. In the simulated world of Fig. 4.22, three spot lights are used to show one starting and two goal positions. Using the robot navigation model, presented in Fig. 4.16, the simulated Turtlebot robot navigates from the starting position to each goal positions (i.e. goal 1 and goal 2), successfully. During its navigation from start to goal 1 or goal 2, the robot avoids the obstacles present in each of the path. In our case, the two dimensional coordinates of the start, goal 1 and goal 2 positions are given as (0.0007, 0.0000), (5.9500, 0.9015) and (7.1586, 2.0189), respectively.

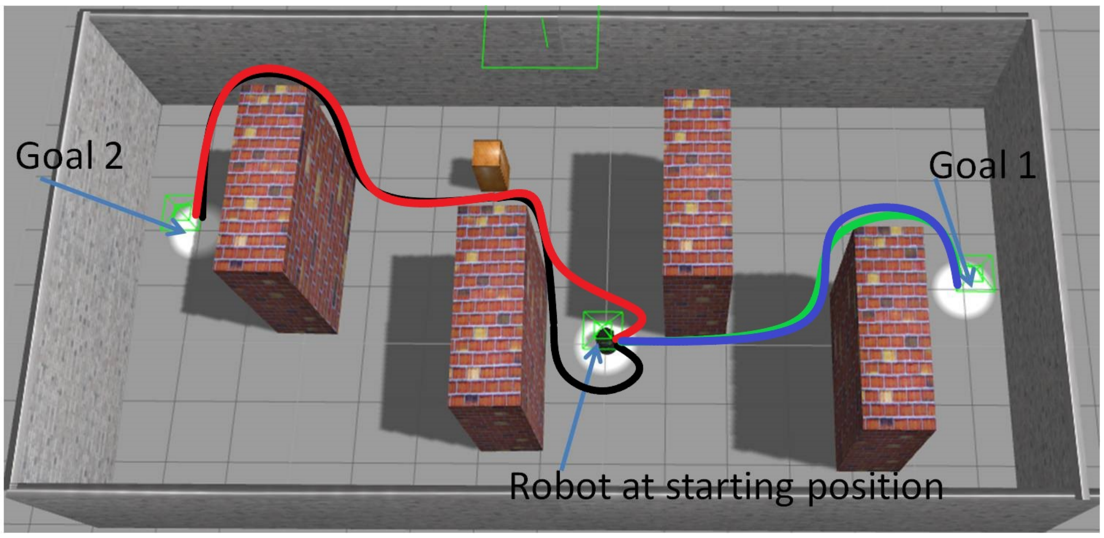


Figure 4.22.: Paths followed by the robot in simulated world.

In addition, at start, the robot is heading towards positive X-axis direction. In the navigation model (presented by Fig. 4.16), the fuzzy logic controller block provides the required change in the angular velocity for obstacle avoidance. Consequently, the robot avoids the obstacles encountered in the path. The paths followed by the robot, using Mamdani and Sugeno-type FISs, are shown using different colours (defined in Table 4.2).

Table 4.2.: Coloured paths and FISs used in the Fig. 4.22.

Path, Goal, and FIS	**Colour**
Path to “Goal 1” using Mamdani-type FIS	Blue
Path to “Goal 2” using Mamdani-type FIS	Red
Path to “Goal 1” using Sugeno-type FIS	Green
Path to “Goal 2” using Sugeno-type FIS	Black

The proposed model for the robot navigation is implemented on real Turtlebot robot equipped with Microsoft *Kinect XBOX 360* sensor. Robot path from starting position to the goal position, in a real world environment, is presented by Fig. 4.23. The linear velocity of the robot is taken as 0.5 *meter per second*. It is evident from Fig. 4.23 that the robot has to avoid obstacles on both of its sides during its drive from start to goal. In addition, some of the portion of the corridor is fenced by the iron railing instead of solid brick wall fence. Therefore, this real environment is an excellent case for testing the proposed navigation model and FISs.

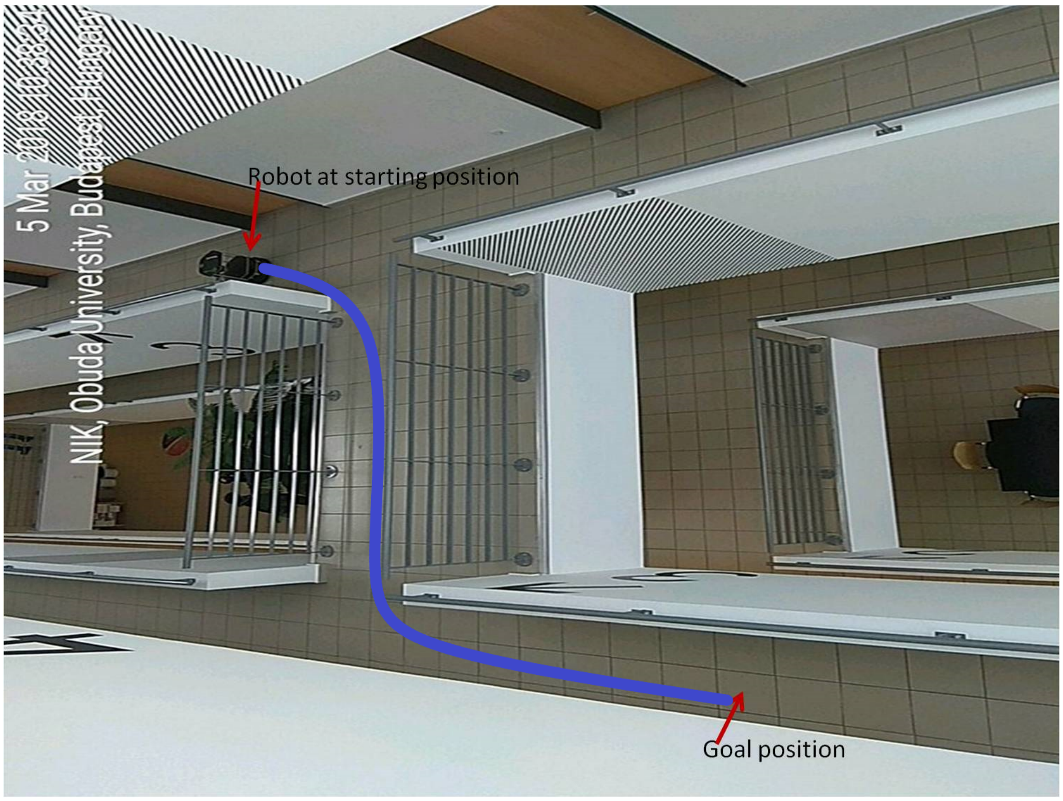

5 Mar 2018
NIK, Obuda University, Budapest, Hungary
Robot at starting position
Goal position

Figure 4.23.: Path followed by real robot in real world environment.

The Sugeno-type FIS is used as the FIS of the fuzzy controller block of the robot navigation model. It is clear from the path that the robot takes turn whenever the fence wall or railing comes in the path, otherwise, the robot directly navigates towards the goal. Own publication pertaining to Section 4.3 is provided in [NK-123].

4.3.4. Summary

The ranges and the types of input membership functions of Sugeno-type fuzzy inference systems can be successfully obtained using ANFIS model. Further, the number of required rules and the number of output membership functions generated from the ANFIS model differ from the Mamdani-type FIS. It can be noted from the results that the robot follows the same paths using Mamdani and Sugeno FIS except few situations. The FIS generated through the simulator are capable of obstacle avoidance in real world environment for the real robot. For better results, it has been observed through rigorous experimental work that the linear velocity of the real robot should be close to the linear velocity taken for the simulated robot.

Therefore, in addition to the static obstacles avoidance, the avoidance for dynamic obstacles is achieved under the reasonable conditions that the speed of motion of this obstacle is much smaller than that of the robot, and its size is limited in comparison with that of the whole workspace.

5. Obstacle recognition and avoidance during robot navigation in unknown static environment

To perform LASER scan matching and obstacle recognition during robot navigation, this chapter considers that the environment, in which the robot is navigating, is static or at least very slowly varying.

5.1. Robot navigation with obstacle recognition using LASER sensor

Feature extraction technique is used by many of the researchers for providing new scientific contributions in image processing and pattern recognition (e.g. [124–126]). The complexity of the object matching task rises for similar objects [127]. In obstacle avoidance, feature extraction method is used to find the path points in [128]. Objects can be extracted using the video sequence of camera mounted on a robot. Using video camera, a keypoint-based method, to produce map of the environment, is given by [129]. However, using video camera for feature extraction of the objects during the robot navigation is computationally expensive [130]. In addition, feature selection in high-dimensional data is a complex task [131]. For the robot navigation including obstacle detection and/or search related operations, feature extraction techniques can be applied. A task-based selection of features is given in [132]. Mobile robots can be controlled using distance and angle features of the image of targets [133]. Frequently appearing objects can be classified as edges, corners or planes. A feature extraction method using neural network is applied to classify the objects in [134]. Schemes for motion related feature extraction using range images are presented in [135]. Using the LASER ranges, methodologies for feature extraction are described in [136–138]. However, the state-of-the-art feature extraction methods yet not considered standard deviation of the similar objects for obstacle recognition purpose. In addition, to keep the computation cost realistic, feature extraction on the LASER range data can be considered.

Robot navigation in unknown and static environments may result in aimless wandering, corner traps and repetitive path loops. To address these issues, this section presents the solution by comparing the standard deviation of the distance ranges of the obstacles appeared in the robot navigation path. For the similar obstacles, the standard deviations of distance range vectors, obtained from the LASER range finder sensor of the robot at similar pose, are very close to each other. Therefore, the measurements of odometer sensor are also combined with the standard deviation to recognize the location of the obstacles. An algorithm, with obstacle detection feature, is presented for robot navigation. The algorithm checks the similarity of the distance range vectors of the obstacles in the path and uses this information in combination with the odometer measurements to identify the obstacles and their locations. The experimental work is carried out using Gazebo simulator.

In this section, the scan range vectors are compared to detect the previously occurred obstacles on the same locations of the navigation path.

The remaining section is organized as follows: The Section 5.1.1 provides the problem definition and solution. Section 5.1.2 is dedicated to the experimental results and discussion. The summary of the section is provided in Section 5.1.3.

5.1.1. Problem definition and solution

During robot navigation in unknown environment, the robot may trap in a loop and may repeat the same path again and again. This may be due to the presence of static obstacles. To deal with this type of problem, there is a need to find a method which can recognize the previously occurred obstacles in successive repetition of the path. So that, the robot can be directed to reverse the angular velocity applied so far. In this way, the robot will come out of the repetitive navigation path loop.

The LASER scanning is a reliable method for obstacles range finding. Generally, the LASER scanner produces a vector of ranges ($\mathbf{R}$) of obstacles. If there are n number of readings in a scan then $\mathbf{R}$ can be expressed by (5.1) as follows:

$$\mathbf{R} = [d_k], \text{ for } k = 1 \text{ to } n. \tag{5.1}$$

where, d_k is the k^{th} distance range in $\mathbf{R}$.

Standard deviations of two range vectors can be compared to find the similarity between them. If the standard deviations of the two scan range vectors are close enough then there is the possibility that these scans belong to the same obstacle.

Algorithm 5 Obstacle recognition using LASER sensor and odometer

```
1: Create vectors: $\mathbf{U}$, $\mathbf{V}$, $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{T}$, $\mathbf{S}$;
2: Initialize: $R_{max}, count, D_t, c, T_s, T_f, \sigma_t, P_t, v_c, \omega_c, v_b$;
   (The description of variables and initialized values can be found in Table 5.1.)
3: while ($Time < T_f$) do
4:     Receive scan reading from robot;
       $\mathbf{R} \leftarrow$ scan ranges received;
5:     for $i \leftarrow 1$ to length of $\mathbf{R}$ do
6:         if $\mathbf{R}(i) == NaN$ then
7:             $\mathbf{R}(i) \leftarrow R_{max}$;
8:         end if
9:     end for
10:    $count \leftarrow count + 1$;
11:    Collect odometer readings at the time of scan;
       $\mathbf{X}(count) \leftarrow X$ coordinate from odometer;
       $\mathbf{Y}(count) \leftarrow Y$ coordinate from odometer;
12:    $D_{min} \leftarrow$ minimum of $\mathbf{R}$;
13:    if $D_{min} < D_t$ then
14:        $c \leftarrow c + 1$;
15:        $\sigma_{\mathbf{R}} \leftarrow$ standard deviation of $\mathbf{R}$;
16:        $\mathbf{S}(c) \leftarrow \sigma_{\mathbf{R}}$;
17:        $\mathbf{U}(c) \leftarrow X$ coordinate from odometer;
18:        $\mathbf{V}(c) \leftarrow Y$ coordinate from odometer;
19:        $\mathbf{T}(c) \leftarrow$ time of scan;
20:        for $j \leftarrow 1$ to (length of $\mathbf{S} - 1$) do
21:            $D_{j,c} \leftarrow \sqrt{(\mathbf{U}(c) - \mathbf{U}(j))^2 + (\mathbf{V}(c) - \mathbf{V}(j))^2}$ ;
22:            $\sigma_r \leftarrow |(\mathbf{S}(c) - \mathbf{S}(j))|$;
23:            if ($\sigma_r < \sigma_t$ and $D_{j,c} < P_t$) then
24:                Match found between scans $j$ and $c$.
                   Record corresponding scan times and positions of robot with
                   respect to $j$ and $c$.;
25:            end if
26:        end for
27:        $v \leftarrow -|v_b|$;
28:        $\omega \leftarrow \omega_c$;
29:    else
30:        $v \leftarrow v_c$;
31:        $\omega \leftarrow 0$;
32:    end if
33:    Send $v, \omega$ to robot;
34: end while
```

Further, the standard deviation, in combination of the robot position coordinates, can be used to confirm that the scan ranges belong to the same obstacle. To achieve this solution, the Algorithm 5 is presented.

Standard deviation ($\sigma_{\mathbf{R}}$) of $\mathbf{R}$ can be expressed by (5.2) as given below:

$$\sigma_{\mathbf{R}} = \sqrt{\frac{\sum_{k=1}^{n}(d_k - \overline{d})^2}{n}}. \tag{5.2}$$

where, $\overline{d}$ is the mean, defined by (5.3), of $\mathbf{R}$.

$$\overline{d} = \frac{1}{n}\sum_{k=1}^{n} d_k. \tag{5.3}$$

5.1.2. Experimental results and discussion

The numerical values of the variables, initialized in Algorithm 5, are presented in Table 5.1. The robot is driven, using Algorithm 5, in the *Gazebo world* as given by Fig. 5.1.

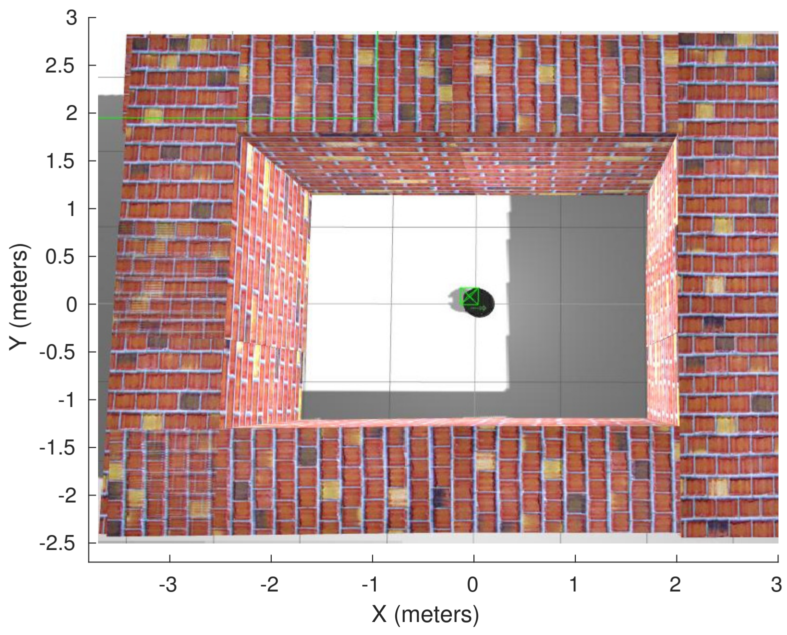


Figure 5.1.: Simulated world in Gazebo.

In the beginning, in Fig. 5.1, the *Turtlebot* is located at the centre of the world and heading towards positive X-axis. In Fig. 5.1, coordinate axes and their measurement marker are added to provide the understanding of the coordinate positions in the *Gazebo world*. The *Gazebo world* is constructed by using *brick boxes* of the dimensions $3 \times 1 \times 3$ from the model database of the *Gazebo* simulator. Purposely, the *Gazebo world* is constructed using only one model so that the same type of obstacle encountered each time to increase the complexity in differentiating the obstacles and their respective locations.

Table 5.1.: Description of variables and initialized values

Variable	**Description**	**Initialized Value**
R_{max}	Maximum range of the LASER scanner	Default
$count$	a counter variable	0
D_t	Threshold distance between robot and obstacles	0.6 (meter)
c	a counter variable	0
T_s	Starting time	0
T_f	Finish time	80 (seconds)
σ_t	Threshold difference between σ of two scans	0.0005
P_t	Threshold distance between two positions	0.2 (meter)
v_c	Constant forward linear velocity	0.5 (meter/sec.)
ω_c	Constant Angular velocity	-0.6 (radian/sec.)
v_b	Constant backward linear velocity	-0.02 (meter/sec.)
v	Published linear velocity	-
ω	Published angular velocity	-
$\mathbf{U}$	Vector of X coordinates of poses when obstacles are near	-
$\mathbf{V}$	Vector of Y coordinates of poses when obstacles are near	-
$\mathbf{X}$	Vector of X coordinates of all poses	-
$\mathbf{Y}$	Vector of Y coordinates of all poses	-
$\mathbf{T}$	Vector of time at which scans performed	-
$\mathbf{S}$	Vector of standard deviations scan range vectors	-

In addition, the maze structure is kept closed so that the robot can be forced to follow a repeated path in a loop.

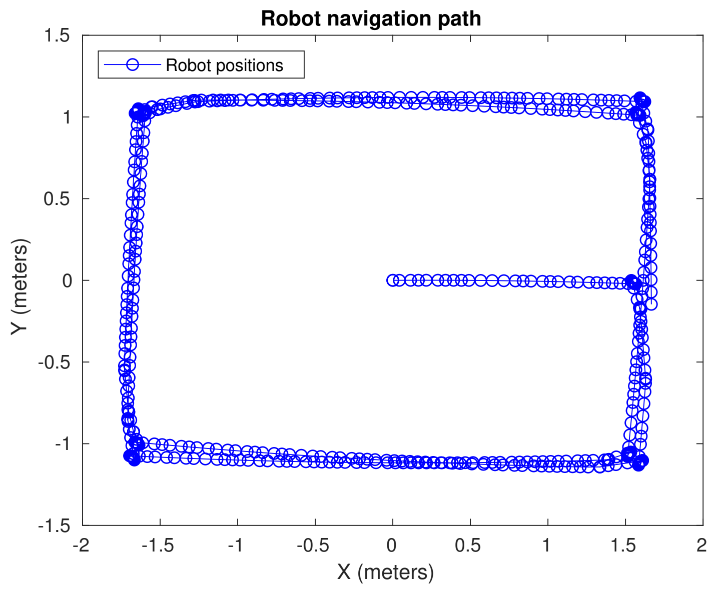


Figure 5.2.: Robot path generated from odometer measurements.

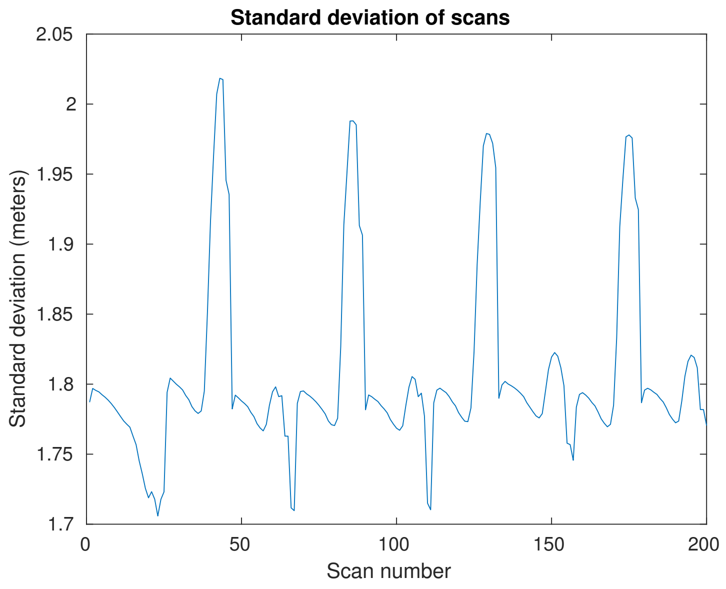


Figure 5.3.: Standard deviation of scan ranges.

Robot is driven at a constant forward linear velocity and zero angular velocity until any closed obstacle is encountered. In this navigation process, if any obstacle is close enough to the robot then a backward linear velocity and a constant non zero angular velocity is provided to the robot. Using Algorithm 5, the robot navigates in a path loop. Fig. 5.2 shows the resultant path of the robot. It is clear from Fig. 5.2 that robot is driven in a navigation loop so that it can observe the same obstacles more than once. During the navigation process, the obstacle distances are measured by LASER scan ranges. For *Turtlebot*, a LASER scan message contains a vector of 640 range readings of the scan area from right to left. If robot pose is same, each time, the LASER scan of an obstacle produces same vector of ranges.

To find the similarity between two vectors of ranges, standard deviations of the two vectors are compared. Fig. 5.3 presents the standard deviations of the range vectors of different scans performed during the navigation of the robot. Fig. 5.3 depicts that there are periodic spikes which are due to the similarity of the obstacles and repetitive path loop. Further, the pattern of the graph in the periods is found very similar to each other because of the similarity of the obstacles. Subsequently, it can be inferred that the standard deviations of the similar obstacles are observed close to each other. However, in addition to the standard deviation, the odometer measurements, for the robot locations during the navigation, are used to determine the location of the obstacles. Further, on the basis of the standard deviations and odometer measurements, it can not be identified that either the obstacle was visited by the robot previously or the obstacle is encountered first time during the navigation. For this reason, there is the need for additional method to recognize that whether any particular obstacle is found first time or it is repeated. The occurrence number (i.e. first time encountered or repeated) of an obstacle can be described on the basis of the time of the scan. Obviously, the standard deviations and robot positions of two nearby scans at very short time of difference will not vary largely. On the other hand, if the standard deviations of the distance range vectors of any two scans and the robot positions received from the odometer at the time of scans are very close to each for both of the scans and the time difference between the two scans is larger than a threshold value then it can be said that the obstacle is same and the occurrence of this obstacle is the repeated one.

Table 5.2 presents the positions of the robot and their corresponding times with respect to scan range matches. It can be observed from the Table 5.2 that using standard deviations and robot positions, the similarity between the two scan range vectors are found either at times with short difference or at times where the time interval is larger. The matches at short and large time intervals indicate about the occurrence time of the obstacles. For example, the first row of Table 5.2 indicates that the match between the range vectors is found at the scan times 10.23 (seconds) and 10.46 (seconds). Thus, the time interval (Time2 – Time1) between these scans is less than a second.

Table 5.2.: LASER scan range vectors matches and robot positions

Time1 (Sec.)	**Time2 (Sec.)**	**Scan Numbers Matched**	**Robot Position at Time1**	**Robot Position at Time2**
10.23	10.46	35, 37	(1.59, -1.11)	(1.59, -1.11)
19.08	21.28	48, 63	(-1.66, -0.99)	(-1.67, -0.93)
21.73	21.84	64, 65	(-1.69, -0.86)	(-1.70, -0.81)
26.32	26.44	69, 70	(-1.64, 1.05)	(-1.64, 1.05)
28.68	28.8	85, 86	(-1.60, 1.05)	(-1.55, 1.06)
37.62	38.64	95, 103	(1.59, 1.01)	(1.57, 1.02)
37.17	39.66	92, 107	(1.59, 1.01)	(1.61, 0.89)
9.68	44.91	31, 113	(1.59, -1.12)	(1.51, -1.07)
9.68	45.13	31, 115	(1.59, -1.12)	(1.51, -1.08)
10.56	45.13	38, 115	(1.60, -1.11)	(1.51, -1.08)
44.91	45.13	113, 115	(1.51, -1.07)	(1.51, -1.08)
8.97	45.24	26, 116	(1.59, -1.08)	(1.51, -1.07)
46.12	46.3	122, 123	(1.52, -1.06)	(1.52, -1.06)
19.24	55.1	49, 133	(-1.66, -0.99)	(-1.63, -1.08)
21.13	56.32	62, 141	(-1.65, -0.97)	(-1.68, -1.08)
19.61	56.7	52, 143	(-1.66, -0.99)	(-1.68, -1.08)
19.77	56.84	53, 144	(-1.65, -0.99)	(-1.68, -1.08)
19.88	56.97	54, 145	(-1.65, -0.99)	(-1.68, -1.08)
20.66	57.41	60, 148	(-1.64, -1.00)	(-1.67, -1.09)
55.74	58.29	137, 154	(-1.69, -1.07)	(-1.67, -1.10)
26.61	64.17	71, 159	(-1.64, 1.05)	(-1.59, 1.03)
64.17	64.4	159, 161	(-1.59, 1.03)	(-1.59, 1.03)
26.84	64.57	73, 162	(-1.64, 1.04)	(-1.59, 1.03)

Keeping in mind the forward linear velocity of the robot (i.e. 0.5 meter/seconds), it is clear that the scans were performed at nearly same time, therefore, the match is found between the first appearance of the obstacle and this is not the case of repetition.

Similarly, the last row of the Table 5.2 presents the time interval between the scans as 37.73 (seconds). Therefore, on the basis of robot's linear and angular velocities it can be deduced that the occurrence of the obstacle at Time2 (at last row) is repeated in navigation path loop.

5.1.3. Summary

Using the standard deviation of laser scan range vectors, the similarity of obstacles is examined in unknown static environment. In addition to the standard deviation, the robot position is also used to identify whether the obstacle position is same as of some earlier position during the navigation path loop. Time interval between the scans plays an important role in the detection of the repetitive occurrences of the obstacles. Using the standard deviation of LASER scan range vectors, robot positions and time of scan, the angular velocity can be reversed to break the path loop.

5.2. Obstacle recognition and avoidance during robot navigation in unknown static environment

In this Section, firstly, a model for robot navigation in unknown environment is presented in MATLAB-Simulink. This model is applicable for obstacle avoidance during the robot navigation. However, the first model is unable to recognize the re-occurrences of the obstacles during the navigation. Secondly, an advanced algorithm, based on the standard deviations of LASER scan range vectors, is proposed and implemented for robot navigation. The standard deviations of the LASER scans, robot positions and the time of scans with similar standard deviations are used to obtain the obstacle recognition feature. In addition to the obstacle avoidance, the second algorithm recognises the re-appearances of the obstacles in the navigation path. Further, the obstacle recognition feature is used to break the repetitive path loop in the robot navigation. The experimental work is carried out on the simulated Turtlebot robot model using the *Gazebo* simulator.

Remaining part of this section is structured as follows: a robot navigation model to navigate a robot in unknown environment is presented in Section 5.2.1. Section 5.2.2 proposes an advanced solution for obstacle avoidance using obstacle recognition. The results are given in Section 5.2.3. Section 5.2.4 is for the summary of the section.

5.2.1. Robot navigation model and problem definition

A robot navigation model with obstacle avoidance is given in Fig. 5.4. In Fig. 5.4, the *Odom Subscriber* block is subscribing the */odom* topic of the ROS. Using the *Odom Subscriber* block the given navigation model is receiving the positions of the robot during the navigation. The *'/scan' Subscriber* block of the navigation model is subscribing the */scan* topic of the ROS. On the other hand, the robot is publishing its pose on the ROS topic */odom* and LASER scan messages on the ROS topic */scan*. Therefore, the robot positions and its LASER sensor readings are available to the robot navigation model presented by Fig. 5.4. The *Publish geometry_msgs/Twist* block is publishing the linear and angular velocities, computed by the given model, on the ROS topic *mobile_base/commands/velocity*. On the other side, the robot subscribes the ROS topic *mobile_base/commands/velocity*, therefore, the robot is capable of receiving the linear and angular velocities published by the given model. The *Path positions data Writing* block contains a MATLAB function to write the robot positions in a text file for the future use.

The Algorithm 6 is used in *MATLAB Function* block of the Fig. 5.4. The input parameters of the function defined in Algorithm 6 are corresponding to the input parameters of

MATLAB Function block of Fig. 5.4 from *Msg* to *DT*, respectively in that order.

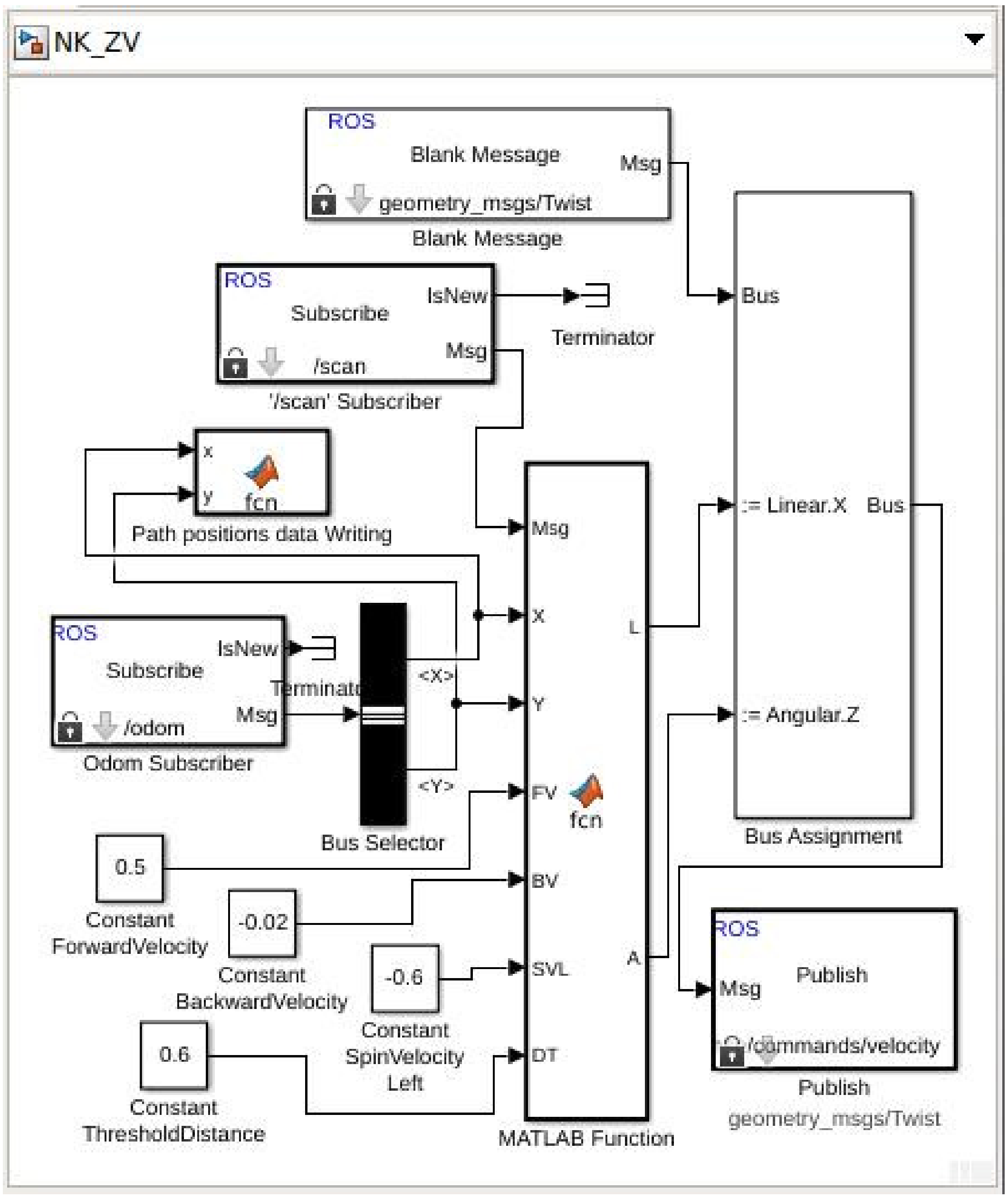


Figure 5.4.: Robot navigation model in Simulink.

The robot navigates on a repetitive path using the negative or positive angular velocity. The rest of the blocks in the model are self explanatory.

Algorithm 6 Obstacles avoidance algorithm

```
1:  function FCN($M, X, Y, v_c, v_b, \omega_c, D_t$)
2:      Receive scan reading from robot;
3:      $\mathbf{R} \leftarrow$ scan ranges received from $M$;
4:      $D_{min} \leftarrow \min(\mathbf{R})$
5:      if ($D_{min} < D_t$) then
6:          $\omega \leftarrow \omega_c$;
7:          $v \leftarrow -|v_b|$;
8:      else
9:          $\omega \leftarrow 0$;
10:         $v \leftarrow v_c$;
11:     end if
12:     Store $X, Y$;
13:     return $\omega, v$;
14: end function
```

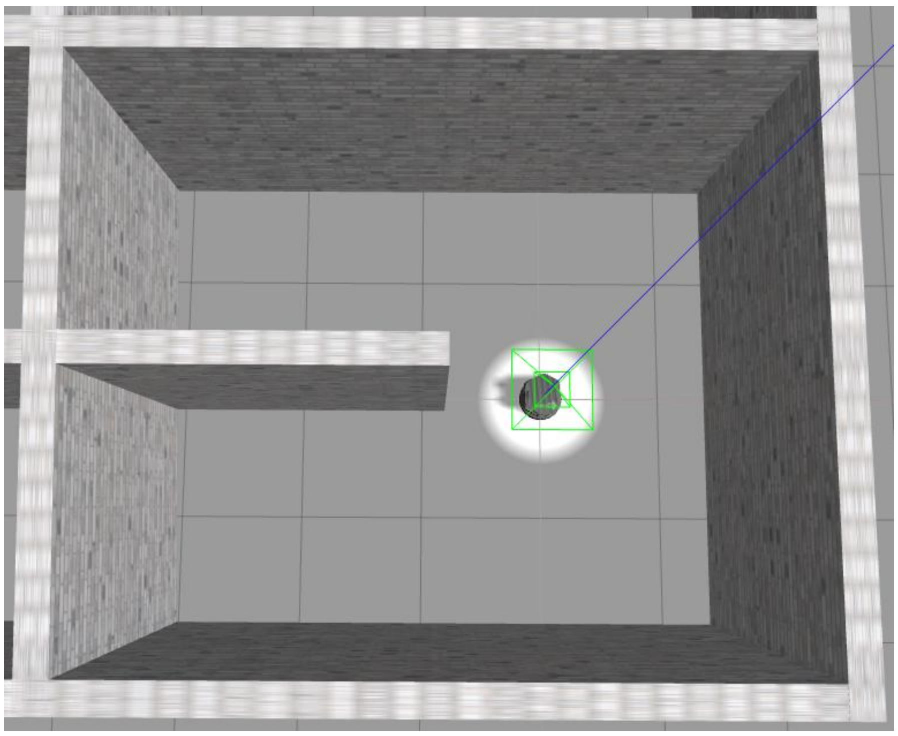


Figure 5.5.: Simulated Turtlebot robot in a Gazebo world.

The robot navigation model (Fig. 5.2.1), using the Algorithm 6 in its *MATLAB Function* block, is executed with the simulated Turtlebot robot in Gazebo world depicted by Fig. 5.5. The Gazebo world is constructed in the Gazebo simulator using the *Gray_wall* available in the model database. To increase the complexity, the Gazebo world is kept closed so that the robot can navigate in a path loop. In addition, the Gazebo world is built by a single type of walls so that the complexity of the obstacle recognition may be raised. Initially, the Turtlebot robot model is present at the $(0, 0)$ coordinate position and heading towards the positive X-axis direction. For better visibility and identification of the robot in the Gazebo world, a spot light is taken into consideration.

Following the Algorithm 6, the robot follows a straight path (i.e. with zero angular velocity) in the forward direction until it finds any obstacle closer than a threshold distance. When the robot finds any nearby obstacle (closer than threshold distance) then it is instructed to spin in left (using positive angular velocity) or right (using negative angular velocity) side of its heading direction. However, in either of the cases (i.e. left or right spins) the robot traps in a repetitive navigation path loop and continue to navigate on almost same path loop. In this situation the other possible paths remain unexplored.

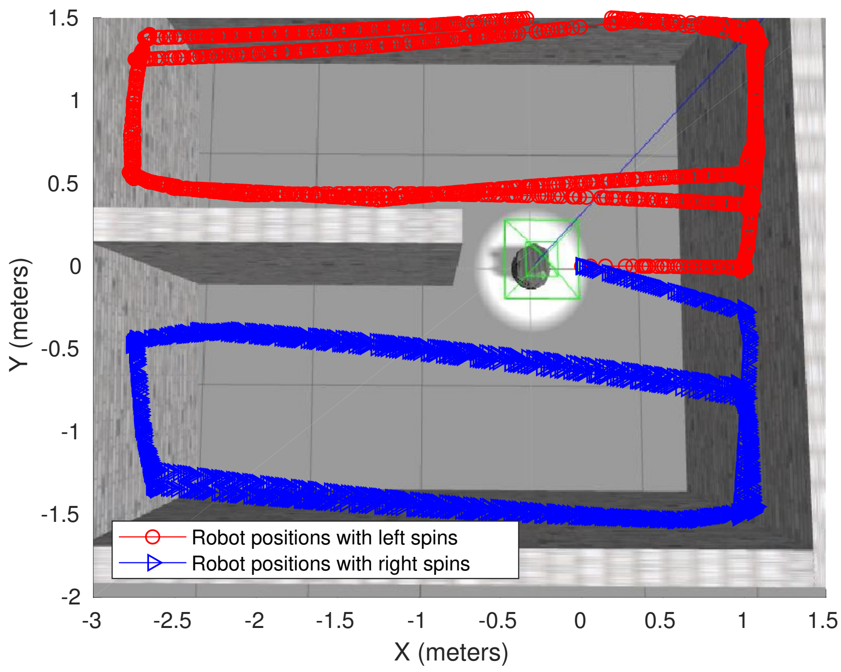


Figure 5.6.: Repetitive paths followed by robot (using the proposed model given in Fig. 5.4) in Gazebo world.

The navigation path followed by the robot is shown in Fig. 5.6. In fact, in Fig. 5.6, the robot navigation path plot is combined with the Gazebo world (given in Fig. 5.5) to make the navigation path more understandable with respect to the coordinate system point of view as well. In Fig. 5.6, the upper path loop (i.e. in positive Y coordinates) is with positive angular velocity. On the other-hand, the lower path loop (i.e. in negative Y coordinates) is received by using negative angular velocity.

In the situation explained by Fig. 5.6, the robot will repeatedly follow the same path loop and will not turn to the other possible paths. However, for the searching robots, it may be required to explore the whole area. To overcome from the problem, an advanced solution is presented by Algorithm 7 in the Section 5.2.2 ahead.

5.2.2. Advanced algorithm for obstacle avoidance with obstacle recognition

For obstacle range detection, the LASER scan is a trustworthy technique. Typically, a LASER scan contains n numbers of distance range readings. The distance range vector ($\mathbf{R}$), of a LASER scan, can be defined by following (5.1).

Table 5.3.: Variables' descriptions for Algorithms 6−8.

Variable	**Description**
T_c	Current time of the system
$\mathbf{V}$	Y coordinates vector of positions when closer obstacles
$\mathbf{U}$	X coordinates vector of positions when closer obstacles
v	Linear velocity of robot
$\mathbf{Y}$	Y coordinates vector of all positions
$\mathbf{X}$	X coordinates vector of all positions
ω	Angular velocity of robot
$\mathbf{S}$	Standard deviations vector of scan ranges
$\mathbf{T}$	Vector of scans times

Standard deviation ($\sigma_{\mathbf{R}}$) of $\mathbf{R}$ can be expressed as in (5.2). The standard deviation of the LASER scans of the similar objects will produce same results. Therefore, the standard deviation values of LASER scans of objects can be used to identify and differentiate the objects scanned. Using standard deviations, the Algorithm 7 gives the solution of the problem of repetitive paths (as discussed in Section 5.2.1) and breaks the repetitive path loop in robot navigation.

Algorithm 7 Obstacles recognition and avoidance algorithm

```
 1: Create vectors: $\mathbf{Y}, \mathbf{S}, \mathbf{U}, \mathbf{T}, \mathbf{V}, \mathbf{X}$;
 2: Initialize: $D_t,\ R_m,\ c,\ e,\ g,\ T_d,\ T_f,\ T_t,\ T_s,\ P_t,\ \omega_c,\ \sigma_t,\ v_b,\ v_c$;
    (Definition and initialization of variables can be found in Table 5.4.)
 3: while $((T_c - T_s) \leq T_f)$ do
 4:     Read LASER scan from robot;
 5:     $\mathbf{R} \leftarrow$ Distance range vector received from robot;
 6:     for $j \leftarrow 1$ to length $(\mathbf{R})$ do
 7:         if $(\mathbf{R}(j)$ is not defined$)$ then
 8:             $\mathbf{R}(j) \leftarrow R_m$;
 9:         end if
10:     end for
11:     $D_{min} \leftarrow \min(\mathbf{R})$;
12:     $\mathbf{Y}(g) \leftarrow Y$ coordinate of the robot position;
13:     $\mathbf{X}(g) \leftarrow X$ coordinate of the robot position;
14:     $g \leftarrow g + 1$; ;
15:     if $(D_{min} < D_t)$ then
16:         $\mathbf{V}(c) \leftarrow Y$ coordinate of the robot position;
17:         $\mathbf{U}(c) \leftarrow X$ coordinate of the robot position;
18:         Compute $\sigma_{\mathbf{R}}$ using (5.2);
19:         $\mathbf{S}(c) \leftarrow \sigma_{\mathbf{R}}$;
20:         $\mathbf{T}(c) \leftarrow$ time at which scan is performed;
21:         $c \leftarrow c + 1$;
22:         for $k \leftarrow 1$ to $(\text{length}(\mathbf{S}) - 1)$ do
23:             $T_d \leftarrow |\mathbf{T}(k) - \mathbf{T}(c)|$;
24:             $\sigma_r \leftarrow |(\mathbf{S}(c) - \mathbf{S}(k))|$;
25:             $D_{c,k} \leftarrow \sqrt{(\mathbf{V}(c) - \mathbf{V}(k))^2 + (\mathbf{U}(c) - \mathbf{U}(k))^2}$ ;
26:             if $\left(D_{c,k} < P_t \quad \& \quad \sigma_r < \sigma_t \quad \& \quad T_d > T_t\right)$ then
27:                 $e \leftarrow e + 1$;
28:                 Call Procedure REVVEL$(e)$;
29:             end if
30:         end for
31:         $\omega \leftarrow \omega_c$;
32:         $v \leftarrow -|v_b|$;
33:     else
34:         $\omega \leftarrow 0$;
35:         $v \leftarrow v_c$;
36:     end if
37:     Drive the robot using $v, \omega$;
38: end while
```

At the statement 28 of the Algorithm 7, there is a call to the **REVVEL**(e) procedure. The **REVVEL**(e) procedure is explained in Algorithm 8.

Table 5.3 presents the variables' descriptions which are taken through Algorithms 6–8.

Algorithm 8 Procedure to reverse the angular velocity of the robot

```
1:  procedure REVVEL($e$)
2:      if ($e == 1$) then
3:          $\omega_c \leftarrow -\omega_c$;
4:          $T_1 \leftarrow$ current time;
5:      end if
6:      $T_2 \leftarrow$ current time
7:      if ($T_2 - T_1 > T_d$) then
8:          $e \leftarrow 0$;
9:      end if
10: end procedure
```

According to the statement 26 of the Algorithm 7, if the standard deviations, robot positions of the two scans are similar and the time difference between the two scans is larger than a threshold value then this is the re-occurrence of an obstacle.

5.2.3. Experimental results

The numerical values of the variables, initialized in Algorithms 6–8, are presented in Table 5.4. Fig. 5.7 shows the resultant path of the robot.

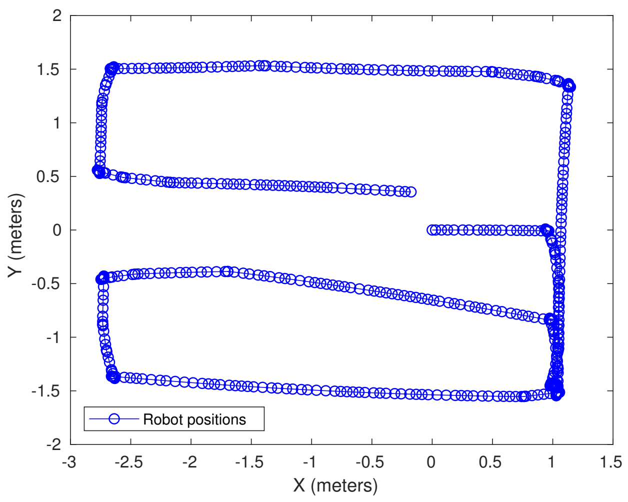


Figure 5.7.: Path followed by robot using advanced Algorithms 7–8.

It is clear from the Fig. 5.7 that robot, using the proposed algorithm, reverses the angular velocity on the re-appearance of the obstacles. In the Fig. 5.7, the initialized angular velocity is negative (See the value of ω_c in Table 5.4). As a result, the robot navigates towards the negative Y coordinates and re-visits back to the position near about $(1, 1.5)$ coordinate position. After reaching at this point, the robots completes its one cycle of the path in the negative Y-axis coordinates. Here, the robot detects the re-visit to the obstacle using the standard deviations, position of itself and the time difference between the scans with similar standard deviations. At this position (i.e. near about $(1, 1.5)$ coordinates position), the angular velocity of the robot reverses and the robot starts the spin in the opposite side (i.e. left side). In this manner, the robot does not enter the previous navigation path loop and heads towards the positive Y-axis coordinates.

Table 5.4.: Initialization of variables for the Algorithms 6–8.

Variable	**Description**	**Initialized Value**
T_f	Finishing time	190 (seconds)
T_d	Time difference of scans	10 (seconds)
T_t	Time threshold	10 (seconds)
T_s	Time of start	0
D_t	Distance threshold of robot and obstacles	0.6 (meter)
P_t	Distance threshold of two positions	0.2 (meter)
σ_t	Threshold fluctuation of σ of scans	0.0005
ω_c	Angular velocity of robot	-0.6 (radian/sec.)
v_b	Backward linear velocity	-0.02 (meter/sec.)
v_c	Forward linear velocity	0.5 (meter/sec.)
e	a counter variable	0
g	a counter variable	0
c	a counter variable	0
R_m	Maximum distance range of LASER sensor	Default

5.2.4. Summary

A model for robot navigation in unknown environment has been proposed and implemented. The Algorithm 6 is capable of obstacle avoidance with repetitive path results. By implementing the Algorithm 7 in combination of the procedure given by Algorithm 8, the robot comes out of the repetitive path and successfully reverses its angular velocity when the same obstacle is found on the next iteration of the path loop. The re-appearances of obstacles are successfully recognized and avoided. Moreover, the obstacle recognition is used to drive the robot to the yet unexplored path. In spite of the similar obstacles in the navigation path, the re-occurrences of the obstacles has been detected using the robot positions and the time of LASER scans performed.

5.3. Obstacle recognition and avoidance using two-sample t-test

In Section 5.2, the Algorithm 7 has presented a solution for obstacle recognition and avoidance using the difference of the standard deviations of LASER-scan distance-range-vectors. In Algorithm 7, the statement 26 is stated as follows:

$$\texttt{if} \quad (D_{c,k} < P_t \quad \& \quad \sigma_r < \sigma_t \quad \& \quad T_d > T_t) \quad \texttt{then}$$

here, the `if` condition composes logical "`AND`" of three conditions. The second condition $(\sigma_r < \sigma_t)$ of `if` compares the difference of standard deviations (σ_r) of two LASER-scan distance-range-vectors with an arbitrary constant value (σ_t). However, setting the value of (σ_t) is problem specific. Therefore, there is research scope to standardise the condition $(\sigma_r < \sigma_t)$.

Various statistical techniques are presented in literature (e.g. [139]). Two independent samples can be compared for similarity on various aspects using t-test [140]. In this section, the comparison of LASER-scan distance-range-vectors is standardised using t-test.

The remaining of the section is structured in following manner: Section 5.3.1 describes Student's t statistics. The t-test for single-mean is explained in Section 5.3.2. Section 5.3.3 presents t-test to compare the means of two independent samples. Application of two-sample t-test in obstacle recognition and avoidance is provided in Section 5.3.4. Section 5.3.5 consists experimental results acquired by applying the algorithm based on the two-sample t-test. Finally, the summary of the section is given in the Section 5.3.6.

5.3.1. The Student's t

For a sample: $\{d_1, d_2, d_3....., d_n\}$, William Sealy Gosset (who used the pseudonym as "Student" [141]) presented the statistics t as follows:

$$t = \frac{\left(\overline{d} - \mu\right)\sqrt{n}}{\sigma_{\mathbf{S}}}. \tag{5.4}$$

where, n is the size of the sample. The mean $\left(\overline{d}\right)$ and standard deviation $(\sigma_{\mathbf{S}})$ of the sample are defined in (5.3) and (5.5), respectively. μ is the mean of the population.

$$\sigma_{\mathbf{S}} = \sqrt{\frac{\sum_{k=1}^{n}(d_k - \overline{d})^2}{n-1}}. \tag{5.5}$$

The statistics t, defined in (5.4), is known as "Student's t" having $(n-1)$ degree of freedom $\left(d_f\right)$.

5.3.2. t-Test for single mean

The steps for the t-test are as follows:

(i) Set-up the null-hypothesis (H_0) as follows:
H_0 : "*There is no significant difference between the sample-mean and mean of the population.*"

(ii) Compute the value of t using (5.4).

(iii) Compare the computed value of t with the critical value of t (say t_c) at the desired level of significance.
`If` $(|t| \leqslant t_c)$ `then` H_0 may be accepted and `if` $(|t| > t_c)$ `then` H_0 is rejected.

For different values of d_f, the critical values of t (i.e. t_c), at 1% and 5% levels of significance, are tabulated in Appendix A.2.

5.3.3. t-Test for difference of two means

Consider two independent samples (x and y), given in (5.6), as follows:

$$\begin{aligned} x &= \{d_{x_1}, d_{x_2}, d_{x_3}, \ldots\ldots, d_{x_{n_1}}\}. \\ y &= \{d_{y_1}, d_{y_2}, d_{y_3}, \ldots\ldots, d_{y_{n_2}}\}. \end{aligned} \tag{5.6}$$

Following [142], the statistics t can be defined by (5.7).

$$t = \frac{\overline{d}_x - \overline{d}_y}{\sigma_{\mathbf{S}} \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}. \tag{5.7}$$

where, n_1 and n_2 are the sizes of the x and y samples considered in (5.6), respectively. The means $\left(\overline{d}_x \textit{ and } \overline{d}_y\right)$ of the two samples (x and y, respectively) can be computed using (5.3). The value of $\sigma_{\mathbf{S}}$ is given by (5.8).

$$\sigma_{\mathbf{S}} = \sqrt{\frac{1}{n_1 + n_2 - 2} \left[\sum_{i=1}^{n_1} (d_{x_i} - \overline{d}_x)^2 + \sum_{j=1}^{n_2} (d_{y_j} - \overline{d}_y)^2 \right]}. \tag{5.8}$$

here,

$$d_f = n_1 + n_2 - 2. \tag{5.9}$$

In this case, the significance of t is tested by following steps:

(i) Set-up the null-hypothesis (H_0) as follows:
H_0 : "The means $\left(\overline{d}_x \textit{ and } \overline{d}_y\right)$ of the two samples do not differ significantly."

(ii) Compute the value of t using (5.8).

(iii) Same as step (iii) of Section 5.3.2.

5.3.4. Obstacle recognition and avoidance using t-test

To apply t-test in obstacle recognition and avoidance, the statements 19, 24, and 26 of the Algorithm 7 can be replaced with the statements given in (5.10), (5.11), and (5.12), respectively.

$$\mathbf{S}(c) \leftarrow \mathbf{R} \tag{5.10}$$

$$h \leftarrow \texttt{tTestNk}\left(\mathbf{S}(c), \mathbf{S}(k)\right) \tag{5.11}$$

$$\texttt{if}\ \ (\ D_{c,k} < P_t \quad \& \ \ h = 0 \ \ \& \quad T_d > T_t)\ \texttt{then} \tag{5.12}$$

where, "`tTestNk`" is name of the procedure (defined in Algorithm 9). The distance range-vectors, $\mathbf{S}(c)$ and $\mathbf{S}(k)$, are the two input arguments for a call to the procedure. The procedure returns "1" `if` the t-test rejects the null-hypothesis (H_0). On contrary, the procedure returns "0" `if` the t-test accepts the null-hypothesis.

Algorithm 9 Procedure to perform t-test on two independent range-vectors x and y.

```
1:  procedure tTestNk($x, y$)
2:      $n_1 \leftarrow$ size of $x$
3:      $n_2 \leftarrow$ size of $y$
4:      $d_f \leftarrow n_1 + n_2 - 2$
5:      $t_c \leftarrow$ tabulated value (Appendix A.9) corresponding to $d_f$ and level of significance.
6:      $\overline{d_x} \leftarrow \frac{1}{n_1}\sum_{i=1}^{n_1} d_{x_i}$
7:      $\overline{d_y} \leftarrow \frac{1}{n_2}\sum_{j=1}^{n_2} d_{x_j}$
8:      Compute $\sigma_{\mathbf{S}}$ using equation (5.8).
9:      Compute $t$ using equation (5.7).
10:     if $(|t| > t_c)$ then
11:         Return 1
12:     else
13:         Return 0
14:     end if
15: end procedure
```

The procedure in Algorithm 9 has two input parameters, x and y, pertaining to the input arguments, $\mathbf{S}(c)$ and $\mathbf{S}(k)$, in it's call at the modified Algorithm 7 using (5.10) – (5.12).

5.3.5. Experimental results and discussion

To obtain the results of application of t-test in Algorithm 7, the Gazebo-world as taken in Fig. 5.5 has been considered. Moreover, the starting position of the robot is nearly $(0, 0)$ (same as in Fig. 5.5). The t-test is used to find out the similarity of two LASER-scan distance-range-vectors at 5% level of significance.

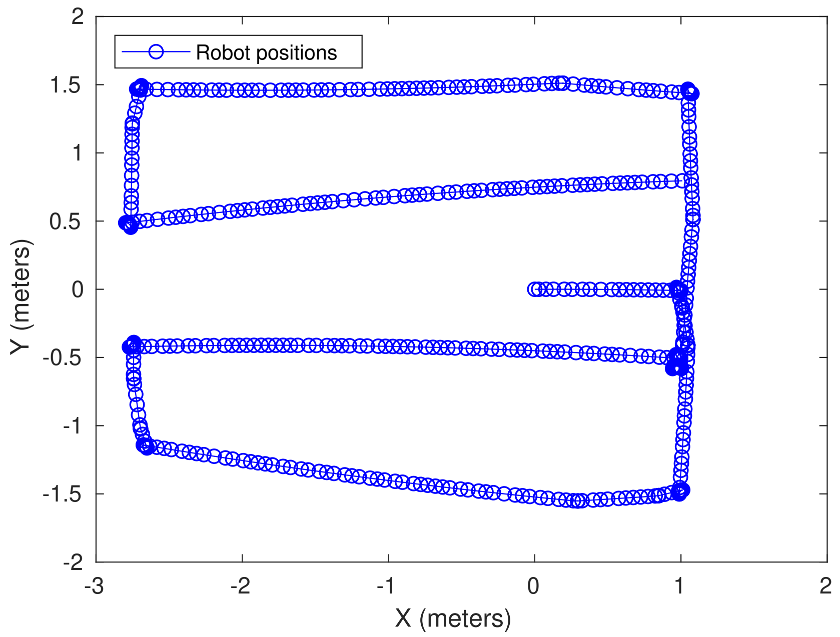


Figure 5.8.: Robot path based on the applied t-test.

Fig. 5.8 presents the robot navigation path by applying the t-test on LASER-scan distance-range-vectors. The following two main observations can be noted down from the Fig. 5.7–5.8:

(i) Similar to the Algorithm 7 (without modifications for t-test) in Section 5.2, the Algorithm 7 (applying t-test) in Section 5.3 successfully detects the similarity of the obstacle wall. As a result, the robot avoids to enter the repetitive path loop.

(ii) Using the t-test, the Algorithm 7 reverses the angular velocity, first time, around the position $(1, -0.5)$ (see Fig. 5.8). On the other hand, without modifications for t-test the Algorithm 7 reverses the angular velocity of the robot, first time, nearby the position $(1, -1.5)$ (see Fig. 5.7).

Using t-test the Algorithm 7 can early detect the first repetition of similar obstacle wall because the t-test is based on standard statistical technique to compare two samples. In this way, there is no need to compare the difference of two LASER-scan distance-range-vectors with some arbitrary constant value like σ_t in statement 26.

5.3.6. Summary

Similarity of LASER-scan distance-range-vectors has been found out using two sample t-test. Two distance-range-vectors are considered as similar *if computed value of the t-statistics is not greater than the tabulated critical value of t*. Importantly, the comparison between the computed t-statistics and tabulated critical value of t is based on the degree of freedom and level of significance.

The "Algorithm for obstacle detection and obstacle avoidance" (proposed in Section 5.2) has been modified using the t-test. As a result, the requirement of comparing "the difference of two LASER-scan distance-range-vectors" with "an arbitrary small value" has been removed.

5.4. Theses group: *Robot navigation with obstacle recognition in unknown static environment having rectangular obstacles*

5.4.1. Thesis point-*Robot navigation in unknown static environment with obstacle recognition using LASER sensor*

The standard deviations of LASER scan distance-range-vectors are suitable for detecting similar obstacle configurations.

Own publication concerned to the **thesis point** 5.4.1: [NK-143, NK-144, NK-145].

5.4.2. Thesis point-*Obstacle recognition and avoidance during robot navigation in unknown static environment*

In some situations, if similar standard deviations of two LASER scans appears at similar locations and the time difference between the two scans is larger than a threshold value then the path loops can be broken by reversing the angular velocity of the robot.

Own publication concerned to the **thesis point** 5.4.2: [NK-146].

5.4.3. Thesis point-*Application of t-test for obstacle recognition and avoidance in robot navigation*

Using t-test, similarity of two LASER-scan distance-range-vectors can be checked without comparing the difference of the distance-range-vectors with an arbitrary value. Consequently, the outcome of the t-test can be used for obstacle detection and avoidance.

Own publication concerned to the **thesis point** 5.4.3: [NK-147].

6. Conclusion, applicability, and future research scope

6.1. Conclusion

This dissertation has presented solutions for robot navigation in known and unknown environments. Firstly, the robot navigation in known and static environment has been considered. Secondly, the robot navigation in unknown and dynamic environment has been presented. Thirdly, obstacle recognition for obstacle avoidance during robot navigation in unknown static environment has been explored. A theses group, consisting three thesis points (5.4.1–5.4.3), has been proposed.

In known and static environment, A* and probabilistic raodmaps approaches for path planning in known and static environment are explored. To find a path from start to goal, the performance of A* algorithm is analysed using Manhattan, octile, and Euclidean distance heuristics. Using probabilistic roadmaps approach, robot simulations are performed to investigate the number of nodes in the PRM and distance between the nodes.

In unknown and dynamic environment, three different methodologies have been used to investigate the robot navigation . Firstly, an algorithm based on robot's bumper events has been proposed for the robot navigation. The algorithm can be useful when the other sensors, of the robot, with higher efficiencies are not working. Secondly, a fuzzy controller with Mamdani-type FIS has been presented. Using this Mamdani-type FIS, a robot navigation model having obstacle avoidance functionality has been developed. Thirdly, an ANFIS based Sugeno-type FIS has been proposed to obtain a robot navigation model.

In unknown static environment, an algorithm for the obstacle recognition using the standard deviations of distance range vectors, received from the laser scanner of the robot during the robot navigation, is provided. The algorithm for the obstacle recognition set the foundation for the thesis point in 5.4.1. Further, the obstacle recognition is used for the obstacle avoidance to achieve the thesis point 5.4.2. Finally, the thesis point 5.4.3, based on the application of t-test in obstacle recognition and avoidance, has been presented.

6.2. Applicability of the results and future research scope

Main results of the study are provided from Chapter 3 to Chapter 5 of the dissertation. Applicability of the results, presented in the dissertation, can be summarized in the points ((i.) to (iii.)) as follows:

(i.) In Chapter 3, the path planning methodologies are taken for static environment only. Therefore, the results presented in this Chapters are applicable in those industrial or household robot navigation operations where the surrounding environment of the robot remains unchanged during the navigation task.

(ii.) In Chapter 4, the robot navigation models have been presented for unknown dynamic environment. Hence, the results of the Chapter can be suitable for the mobile robot navigation operations in the areas where the surroundings may get change during the navigation task.

(iii.) The results presented in Chapter 5 are applicable to the search robots in indoor static environments.

The future research possibilities can be summarized in the following points ((i) – (iii)):

(i) In Chapter 3, heuristic functions have been applied to execute the A* algorithm. However, `if` heuristic overestimates the cost of reaching the goal `then` the heuristic will not be able to find a path to the goal. Future research work can lead to the solution to the overestimation case of the heuristic functions.

(ii) In Chapter 4, `if` obstacle(s) is/are close enough during robot navigation `then` the robot has been directed using the following instructions (a.–c.):

   (a.) `if` obstacle(s) is/are encountered at the left side of the robot vision `then` turn towards right.

   (b.) `if` obstacle(s) is/are encountered at the right side of the robot vision `then` turn towards left.

   (c.) `if` obstacle(s) is/are encountered at the center of the robot vision `then` turn right.

   In case of the instruction 'c.', the present work can be right or left turn biased. Therefore, further research can remove the left or right turn bias in this case.

(iii) In Chapter 5, novel methodologies for obstacle recognition and avoidance have been presented for static environment only. Therefore, the proposed methodologies can further be developed for the dynamic environments too.

Bibliography

References

[1] A. S. Matveev, A. V. Savkin, M. Hoy, and C. Wang. “8 - Biologically-inspired algorithm for safe navigation of a wheeled robot among moving obstacles”. In: *Safe Robot Navigation Among Moving and Steady Obstacles*. Butterworth-Heinemann, 2016, pp. 161–184.

[2] M. Wang, J. Luo, and U. Walter. “A non-linear model predictive controller with obstacle avoidance for a space robot”. In: *Advances in Space Research* 57.8 (2016). Advances in Asteroid and Space Debris Science and Technology - Part 2, pp. 1737–1746.

[3] Y. Chen and J. Sun. “Distributed optimal control for multi-agent systems with obstacle avoidance”. In: *Neurocomputing* 173 (2016), pp. 2014–2021.

[4] Jun-Hao Liang and Ching-Hung Lee. “Efficient collision-free path-planning of multiple mobile robots system using efficient artificial bee colony algorithm”. In: *Advances in Engineering Software* 79 (2015), pp. 47–56.

[5] J. Lee. “Heterogeneous-ants-based path planner for global path planning of mobile robot applications”. In: *International Journal of Control, Automation and Systems* 15.4 (Aug. 2017), pp. 1754–1769.

[6] R. Tang and H. Yuan. “Cyclic error correction based q-learning for mobile robots navigation”. In: *International Journal of Control, Automation and Systems* 15.4 (Aug. 2017), pp. 1790–1798.

[7] A. Narayan, E. Tuci, F. Labrosse, and M. H. M. Alkilabi. “A dynamic colour perception system for autonomous robot navigation on unmarked roads”. In: *Neurocomputing* 275 (2018), pp. 2251–2263.

[8] K. Charalampous, I. Kostavelis, and A. Gasteratos. “Thorough robot navigation based on SVM local planning”. In: *Robotics and Autonomous Systems* 70 (2015), pp. 166–180.

[9] A. Hidalgo-Paniagua, M. A. Vega-Rodríguez, and J. Ferruz. “Applying the MOVNS (multi-objective variable neighborhood search) algorithm to solve the path planning problem in mobile robotics”. In: *Expert Systems with Applications* 58 (2016), pp. 20–35.

[10] A. S. Matveev, A. V. Savkin, M. Hoy, and C. Wang. “3 - Survey of algorithms for safe navigation of mobile robots in complex environments”. In: *Safe Robot Navigation Among Moving and Steady Obstacles*. Butterworth-Heinemann, 2016, pp. 21–49.

[11] A. S. Matveev, A. V. Savkin, M. Hoy, and C. Wang. “4 - Shortest path algorithm for navigation of wheeled mobile robots among steady obstacles”. In: *Safe Robot Navigation Among Moving and Steady Obstacles*. Butterworth-Heinemann, 2016, pp. 51–61.

[12] M. Liu, J. Lai, Z. Li, and J. Liu. “An adaptive cubature Kalman filter algorithm for inertial and land-based navigation system”. In: *Aerospace Science and Technology* 51 (2016), pp. 52–60.

[13] A.L. Amith, A. Singh, H.N. Harsha, N.R. Prasad, and L. Shrinivasan. “Normal Probability and Heuristics based Path Planning and Navigation System for Mapped Roads”. In: *Procedia Computer Science* 89 (2016), pp. 369–377.

[14] M. Đakulović, M. Čikeš, and I. Petrović. “Efficient Interpolated Path Planning of Mobile Robots based on Occupancy Grid Maps”. In: *IFAC Proceedings Volumes* 45.22 (2012). 10th IFAC Symposium on Robot Control, pp. 349–354.

[15] J.-Y. Jun, J.-P. Saut, and F. Benamar. “Pose estimation-based path planning for a tracked mobile robot traversing uneven terrains”. In: *Robotics and Autonomous Systems* 75 (2016), pp. 325–339.

[16] T. T. Mac, C. Copot, D. T. Tran, and R. D. Keyser. “Heuristic approaches in robot path planning: A survey”. In: *Robotics and Autonomous Systems* 86 (2016), pp. 13–28.

[18] H. Williams, W. N. Browne, and D. A. Carnegie. “Learned Action SLAM: Sharing SLAM through learned path planning information between heterogeneous robotic platforms”. In: *Applied Soft Computing* 50 (2017), pp. 313–326.

[19] G. Younes, D. Asmar, E. Shammas, and J. Zelek. “Keyframe-based monocular SLAM: design, survey, and future directions”. In: *Robotics and Autonomous Systems* 98 (2017), pp. 67–88.

[20] K. Lenac, A. Kitanov, R. Cupec, and I. Petrović. “Fast planar surface 3D SLAM using LIDAR”. In: *Robotics and Autonomous Systems* 92 (2017), pp. 197–220.

[21] S. Wen, X. Chen, C. Ma, H.K. Lam, and S. Hua. “The Q -learning Obstacle Avoidance Algorithm Based on EKF-SLAM for NAO Autonomous Walking Under Unknown Environments”. In: *Robot. Auton. Syst.* 72.C (2015), pp. 29–36.

[22] L. Pfotzer, S. Klemm, A. Roennau, J. M. Zöllner, and R. Dillmann. “Autonomous navigation for reconfigurable snake-like robots in challenging, unknown environments”. In: *Robotics and Autonomous Systems* 89 (2017), pp. 123–135.

[23] A. Elfes. “Using occupancy grids for mobile robot perception and navigation”. In: *Computer* 22.6 (June 1989), pp. 46–57.

[24] S. Cossell and J. Guivant. “Concurrent dynamic programming for grid-based problems and its application for real-time path planning”. In: *Robotics and Autonomous Systems* 62.6 (2014), pp. 737–751.

[25] A. Gil, M. Juliá, and Ò. Reinoso. “Occupancy Grid Based graph-SLAM Using the Distance Transform, SURF Features and SGD”. In: *Eng. Appl. Artif. Intell.* 40.C (2015), pp. 1–10.

[26] Z. Liu and G. v. Wichert. “Extracting semantic indoor maps from occupancy grids”. In: *Robotics and Autonomous Systems* 62.5 (2014). Special Issue Semantic Perception, Mapping and Exploration, pp. 663–674.

[27] J. Nordh and K. Berntorp. “Extending the Occupancy Grid Concept for Low-Cost Sensor-Based SLAM”. In: *IFAC Proceedings Volumes* 45.22 (2012). 10th IFAC Symposium on Robot Control, pp. 151–156.

[28] A. M. Santana, K. R.T. Aires, R. M.S. Veras, and A. A.D. Medeiros. “An Approach for 2D Visual Occupancy Grid Map Using Monocular Vision”. In: *Electronic Notes in Theoretical Computer Science* 281 (2011). Proceedings of the 2011 Latin American Conference in Informatics (CLEI), pp. 175–191.

[29] L. Dantanarayana, G. Dissanayake, and R. Ranasinge. “C-LOG: A Chamfer distance based algorithm for localisation in occupancy grid-maps”. In: *CAAI Transactions on Intelligence Technology* 1.3 (2016), pp. 272–284.

[30] P. Schmuck, S. A. Scherer, and A. Zell. “Hybrid Metric-Topological 3D Occupancy Grid Maps for Large-scale Mapping”. In: *IFAC-PapersOnLine* 49.15 (2016). 9th IFAC Symposium on Intelligent Autonomous Vehicles IAV 2016, pp. 230–235.

[31] N. Morales, J. Toledo, and L. Acosta. “Path planning using a Multiclass Support Vector Machine”. In: *Applied Soft Computing* 43 (2016), pp. 498–509.

[32] O. Montiel, U. Orozco-Rosas, and R. Sepúlveda. “Path planning for mobile robots using Bacterial Potential Field for avoiding static and dynamic obstacles”. In: *Expert Systems with Applications* 42.12 (2015), pp. 5177–5191.

[33] Y. M. Marghi, F. Towhidkhah, and S. Gharibzadeh. “A two level real-time path planning method inspired by cognitive map and predictive optimization in human brain”. In: *Applied Soft Computing* 21 (2014), pp. 352–364.

[34] P. E. Hart, N. J. Nilsson, and B. Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (July 1968), pp. 100–107.

[35] M. M. Deza and E. Deza. Springer-Verlag Berlin Heidelberg, 2009.

[36] L. Zuo, Q. Guo, X. Xu, and H. Fu. “A hierarchical path planning approach based on A* and least-squares policy iteration for mobile robots”. In: *Neurocomputing* 170 (2015). Advances on Biological Rhythmic Pattern Generation: Experiments, Algorithms and Applications Selected Papers from the 2013 International Conference on Intelligence Science and Big Data Engineering (IScIDE 2013) Computational Energy Management in Smart Grids, pp. 257–266.

[37] D. Stojcsics. “Autonomous Waypoint-based Guidance Methods for Small Size Unmanned Aerial Vehicles”. In: *Acta Polytechnica Hungarica* 11.10 (2014), pp. 215–233.

[38] E. Hernandez, M. Carreras, and P. Ridao. “A comparison of homotopic path planning algorithms for robotic applications”. In: *Robotics and Autonomous Systems* 64 (2015), pp. 44–58.

[39] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars. “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”. In: *IEEE Transactions on Robotics and Automation* 12.4 (Aug. 1996), pp. 566–580.

[40] R. Geraerts and M. H. Overmars. "Sampling and node adding in probabilistic roadmap planners". In: *Robotics and Autonomous Systems* 54.2 (2006). Intelligent Autonomous Systems, pp. 165–173.

[41] Z. Zheng, Y. Liu, and X. Zhang. "The more obstacle information sharing, the more effective real-time path planning?" In: *Knowledge-Based Systems* 114 (2016), pp. 36–46.

[42] P. Muñoz, M. D. R-Moreno, and D. F. Barrero. "Unified framework for path-planning and task-planning for autonomous robots". In: *Robotics and Autonomous Systems* 82 (2016), pp. 1–14.

[43] R. C. Coulter. *Implementation of the Pure Pursuit Path Tracking Algorithm*. Tech. rep. CMU-RI-TR-92-01. Pittsburgh, PA: Carnegie Mellon University, Jan. 1992.

[44] Q. Zhu, J. Hu, W. Cai, and L. Henschen. "A new robot navigation algorithm for dynamic unknown environments based on dynamic path re-computation and an improved scout ant algorithm". In: *Applied Soft Computing* 11.8 (2011), pp. 4667–4676.

[45] A. Yorozu and M. Takahashi. "Obstacle avoidance with translational and efficient rotational motion control considering movable gaps and footprint for autonomous mobile robot". In: *International Journal of Control, Automation and Systems* 14.5 (Oct. 2016), pp. 1352–1364.

[46] M. Hank and M. Haddad. "A hybrid approach for autonomous navigation of mobile robots in partially-known environments". In: *Robotics and Autonomous Systems* 86 (2016), pp. 113–127.

[47] I. Arvanitakis, A. Tzes, and K. Giannousakis. "Mobile Robot Navigation Under Pose Uncertainty in Unknown Environments". In: *IFAC-PapersOnLine* 50.1 (2017). 20th IFAC World Congress, pp. 12710–12714.

[48] A. S. Matveev, A. V. Savkin, M. Hoy, and C. Wang. "10 - Seeking a path through the crowd: Robot navigation among unknowingly moving obstacles based on an integrated representation of the environment". In: *Safe Robot Navigation Among Moving and Steady Obstacles*. Butterworth-Heinemann, 2016, pp. 229–250.

[49] K. Alisher, K. Alexander, and B. Alexandr. “Control of the Mobile Robots with ROS in Robotics Courses”. In: *Procedia Engineering* 100 (2015). 25th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2014, pp. 1475–1484.

[50] I. Rodríguez-Fdez, M. Mucientes, and A. Bugarín. “Learning fuzzy controllers in mobile robotics with embedded preprocessing”. In: *Applied Soft Computing* 26 (2015), pp. 123–142.

[51] Ngangbam Herojit Singh and Khelchandra Thongam. “Mobile Robot Navigation Using Fuzzy Logic in Static Environments”. In: *Procedia Computer Science* 125 (2018). The 6th International Conference on Smart Computing and Communications, pp. 11–17.

[52] T.D. Frank, T.D. Gifford, and S. Chiangga. “Minimalistic model for navigation of mobile robots around obstacles based on complex-number calculus and inspired by human navigation behavior”. In: *Mathematics and Computers in Simulation* 97 (2014), pp. 108–122.

[53] A. S. Matveev, M. C. Hoy, and A. V. Savkin. “A globally converging algorithm for reactive robot navigation among moving and deforming obstacles”. In: *Automatica* 54 (2015), pp. 292–304.

[54] M. Wang and J. N. K. Liu. “Fuzzy logic-based real-time robot navigation in unknown environment with dead ends”. In: *Robotics and Autonomous Systems* 56.7 (2008), pp. 625–643.

[55] S. A. Moezi, M. Rafeeyan, E. Zakeri, and A. Zare. “Simulation and experimental control of a 3-RPR parallel robot using optimal fuzzy controller and fast on/off solenoid valves based on the PWM wave”. In: *ISA Transactions* 61 (2016), pp. 265–286.

[56] E. Tóth-Laufer, I. J. Rudas, and M. Takács. “Operator dependent variations of the Mamdani-type inference system model to reduce the computational needs in real-time evaluation”. In: *International Journal of Fuzzy Systems* 16.1 (2014), pp. 57–72.

[57] D. N. M. Abadi and M. H. Khooban. “Design of optimal Mamdani-type fuzzy controller for nonholonomic wheeled mobile robots”. In: *Journal of King Saud University - Engineering Sciences* 27.1 (2015), pp. 92–100.

[58] A. Medina-Santiago, J.L. Camas-Anzueto, J.A. Vazquez-Feijoo, H.R. Hernández de León, and R. Mota-Grajales. "Neural Control System in Obstacle Avoidance in Mobile Robots Using Ultrasonic Sensors". In: *Journal of Applied Research and Technology* 12.1 (2014), pp. 104–110.

[59] M. Algabri, H. Mathkour, H. Ramdane, and M. Alsulaiman. "Comparative study of soft computing techniques for mobile robot navigation in an unknown environment". In: *Computers in Human Behavior* 50 (2015), pp. 42–56.

[60] M. S. Masmoudi, N. Krichen, M. Masmoudi, and N. Derbel. "Fuzzy logic controllers design for omnidirectional mobile robot navigation". In: *Applied Soft Computing* 49 (2016), pp. 901–919.

[61] Z. Sun, J. van d. Ven, F. Ramos, X. Mao, and H. Durrant-Whyte. "Inferring laser-scan matching uncertainty with conditional random fields". In: *Robotics and Autonomous Systems* 60.1 (2012), pp. 83–94.

[62] J. Biswas and M. Veloso. "Depth camera based indoor mobile robot localization and navigation". In: *2012 IEEE International Conference on Robotics and Automation*. May 2012, pp. 1697–1702.

[63] A. Ruiz-Mayor, J.-C. Crespo, and G. Trivino. "Perceptual ambiguity maps for robot localizability with range perception". In: *Expert Systems with Applications* 85 (2017), pp. 33–45.

[64] P. Biber and W. Strasser. "The normal distributions transform: a new approach to laser scan matching". In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*. Vol. 3. Oct. 2003, pp. 2743–2748.

[65] W. Kowalczyk and K. Kozłowski. "Control of the differentially-driven mobile robot in the environment with a non-convex star-shape obstacle: Simulation and experiments". In: *Acta Polytechnica Hungarica* 13.1 (2016), pp. 123–135.

[66] B. Lantos and G. Max. "Hierarchical control of unmanned ground vehicle formations using multi-body approach". In: *Acta Polytechnica Hungarica* 13.1 (2016), pp. 137–156.

[67] Q.-L. Li, Y. Song, and Z.-G. Hou. "Neural network based FastSLAM for autonomous robots in unknown environments". In: *Neurocomputing* 165 (2015), pp. 99–110.

[68] F. Kamil, T. S. Hong, W. Khaksar, M. Y. Moghrabiah, N. Zulkifli, and S. A. Ahmad. “New robot navigation algorithm for arbitrary unknown dynamic environments based on future prediction and priority behavior”. In: *Expert Systems with Applications* 86 (2017), pp. 274–291.

[69] H. Mousazadeh, H. Jafarbiglu, H. Abdolmaleki, E. Omrani, F. Monhaseri, M.-reza Abdollahzadeh, A. Mohammadi-Aghdam, A. Kiapei, Y. Salmani-Zakaria, and A. Makhsoos. “Developing a navigation, guidance and obstacle avoidance algorithm for an Unmanned Surface Vehicle (USV) by algorithms fusion”. In: *Ocean Engineering* 159 (2018), pp. 56–65.

[70] Y. Zhao, X. Chai, F. Gao, and C. Qi. “Obstacle avoidance and motion planning scheme for a hexapod robot Octopus-III”. In: *Robotics and Autonomous Systems* 103 (2018), pp. 199–212.

[71] L. Nardi and C. Stachniss. “User preferred behaviors for robot navigation exploiting previous experiences”. In: *Robotics and Autonomous Systems* 97 (2017), pp. 204–216.

[72] L. Zong, J. Luo, M. Wang, and J. Yuan. “Obstacle avoidance handling and mixed integer predictive control for space robots”. In: *Advances in Space Research* 61.8 (2018), pp. 1997–2009.

[73] A. I. Ross, T. Schenk, J. Billino, M. J. Macleod, and C. Hesse. “Avoiding unseen obstacles: Subcortical vision is not sufficient to maintain normal obstacle avoidance behaviour during reaching”. In: *Cortex* 98 (2018), pp. 177–193.

[74] M. Mujahed, D. Fischer, and B. Mertsching. “Tangential Gap Flow (TGF) navigation: A new reactive obstacle avoidance approach for highly cluttered environments”. In: *Robotics and Autonomous Systems* 84 (2016), pp. 15–30.

[75] A. M. Zaki, O. Arafa, and S. I. Amer. “Microcontroller-based mobile robot positioning and obstacle avoidance”. In: *Journal of Electrical Systems and Information Technology* 1.1 (2014), pp. 58–71.

[76] M. Ragaglia, A. M. Zanchettin, L. Bascetta, and P. Rocco. “Accurate sensorless lead-through programming for lightweight robots in structured environments”. In: *Robotics and Computer-Integrated Manufacturing* 39 (2016), pp. 9–21.

[77] Á. Takács, L. Kovács, I. J. Rudas, Radu-Emil Precup, and T. Haidegger. “Models for Force Control in Telesurgical Robot Systems”. In: *Acta Polytechnica Hungarica* 12.8 (Jan. 2015), pp. 95–114.

[78] B. Kosko. “Fuzzy systems as universal approximators”. In: *[1992 Proceedings] IEEE International Conference on Fuzzy Systems*. Mar. 1992, pp. 1153–1162.

[79] J. L. Castro. “Fuzzy logic controllers are universal approximators”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 25.4 (Apr. 1995), pp. 629–635.

[80] L. T. Koczy E. P. Klement and B. Moser. “Are fuzzy systems universal approximators?” In: *International Journal of General Systems* 28.2-3 (1999), pp. 259–282.

[81] W. S. McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (Dec. 1943), pp. 115–133.

[82] Geoffrey E. Hinton David E. Rumelhart and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (Sept. 1986), pp. 533–536.

[83] J J Hopfield. “Neural networks and physical systems with emergent collective computational abilities”. In: *Proceedings of the National Academy of Sciences* 79.8 (1982), pp. 2554–2558.

[84] Jeffrey L. Elman. “Finding structure in time”. In: *Cognitive Science* 14.2 (1990), pp. 179–211.

[85] Tharindu P Miyanawala and Rajeev K Jaiman. “An efficient deep learning technique for the Navier-Stokes equations: Application to unsteady wake flow dynamics”. In: *arXiv preprint arXiv:1710.09099v3* (2018).

[86] V. Volterra. In: *Theory of Functionals and of Integrals and Integro-Differential Equations (reprinted translation of the original work in Spanish issued in Madrid in 1927)*. New York: Dover Publications, 1959.

[87] Teuvo Kohonen. “Self-organized formation of topologically correct feature maps”. In: *Biological Cybernetics* 43.1 (Jan. 1982), pp. 59–69.

[88] L.A. Zadeh. “Fuzzy sets”. In: *Information and Control* 8.3 (1965), pp. 338–353.

[89] J.G. Brown. “A note on fuzzy sets”. In: *Information and Control* 18 (1971), pp. 32–39.

[90] J. Dombi. “A general class of fuzzy operators, the de Morgan class of fuzzy operators and fuzziness measures induced by fuzzy operators”. In: *Fuzzy Sets and Systems* 8 (1982), pp. 197–216.

[91] J. Dombi. “A general class of fuzzy operators, the demorgan class of fuzzy operators and fuzziness measures induced by fuzzy operators”. In: *Fuzzy Sets and Systems* 8.2 (1982), pp. 149–163.

[92] J. -. R. Jang. “ANFIS: adaptive-network-based fuzzy inference system”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 23.3 (May 1993), pp. 665–685.

[93] E.H. Mamdani and S. Assilian. “An experiment in linguistic synthesis with a fuzzy logic controller”. In: *International Journal of Man-Machine Studies* 7.1 (1975), pp. 1–13.

[94] T. Takagi and M. Sugeno. “Derivation of fuzzy control rules from human operator’s control actions”. In: *IFAC Symp. on Fuzzy Information, Knowledge Representation and Decision Analysis.* July 1983, pp. 55–60.

[96] M. R. B. Bahar, A. R. Ghiasi, and H. B. Bahar. “Grid roadmap based ANN corridor search for collision free, path planning”. In: *Scientia Iranica* 19.6 (2012), pp. 1850–1855.

[97] S. Chaudhuri and V. Koltun. “Smoothed analysis of probabilistic roadmaps”. In: *Computational Geometry* 42.8 (2009). Special Issue on the 23rd European Workshop on Computational Geometry, pp. 731–747.

[98] G. Oriolo, S. Panzieri, and A. Turli. “Increasing the connectivity of probabilistic roadmaps via genetic post-processing”. In: *IFAC Proceedings Volumes* 39.15 (2006). 8th IFAC Symposium on Robot Control, pp. 212–217.

[99] T. Bera, M. S. Bhat, and D. Ghose. “Analysis of Obstacle based Probabilistic RoadMap Method using Geometric Probability”. In: *IFAC Proceedings Volumes* 47.1 (2014). 3rd International Conference on Advances in Control and Optimization of Dynamical Systems (2014), pp. 462–469.

[102] Jan Łukasiewicz. “O Logice Trójwartociowej”. In: *Studia Filozoficzne* 270.5 (1988).

[103] Jan Łukasiewicz. Amsterdam: North-Holland Pub. Co., 1970.

[104] József Dániel Dombi. *A special class of fuzzy operators and its application in modelling effects and decision problems (Thesis of PhD).* University of Szeged, Faculty of Science, Informatics, Department of Computer Algorithms, and Artificial Intelligence, PhD School in Computer Science, 2013.

[105] Péter Földesi, János Botzheim, and L. Kóczy. “Eugenic bacterial memetic algorithm for fuzzy road transport traveling salesman problem”. In: *International Journal of Innovative Computing, Information and Control* 7.5 B (May 2011), pp. 2775–2798.

[106] János Botzheim, Yuichiro Toda, and Naoyuki Kubota. “Bacterial memetic algorithm for offline path planning of mobile robots”. In: *Memetic Computing* 4.1 (Mar. 2012), pp. 73–86.

[107] János Botzheim, Yuichiro Toda, and Naoyuki Kubota. “Bacterial memetic algorithm for simultaneous optimization of path planning and flow shop scheduling problems”. In: *Artificial Life and Robotics* 17 (Oct. 2012).

[108] Reinhard Siegmund-Schultze. “Der Beweis des Weierstraßschen approximationssatzes 1885 vor dem hintergrund der entwicklung der fourieranalysis”. In: *Historia Mathematica* 15.4 (1988), pp. 299–310.

[109] M. H. Stone. “The Generalized Weierstrass Approximation Theorem”. In: *Mathematics Magazine* 21.4 (1948), pp. 167–184.

[110] L. -. Wang and J. M. Mendel. “Fuzzy basis functions, universal approximation, and orthogonal least-squares learning”. In: *IEEE Transactions on Neural Networks* 3.5 (Sept. 1992), pp. 807–814.

[111] David Hilbert. “Mathematical problems”. In: *Bulletin of the American Mathematical Society* 8.10 (1902), pp. 437–479.

[112] V.I. Arnold. “On functions of three variables (in russian)”. In: *Dokl. Akad. Nauk. SSSR* 114 (1957), pp. 679–681.

[113] A.N. Kolmogorov. “On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition (in russian)”. In: *Dokl. Akad. Nauk. SSSR* 114 (1957), pp. 953–956.

[114] David A. Sprecher. “On the Structure of Continuous Functions of Several Variables”. In: *Transactions of the American Mathematical Society* 115 (1965), pp. 340–355.

[115] G.G. Lorentz. *Approximation of Functions.* Holt, Reinhard and Winston, New York, 1965.

[116] G.G. Lorentz. *Mathematical Developments Arising from Hilbert’s Problems (F. Browder, ed.)* Vol. 2. American Mathematical Society,Providence, RI, 1976, pp. 419–430.

[117] Edward K. Blum and Leong Kwan Li. “Approximation theory and feedforward networks”. In: *Neural Networks* 4.4 (1991), pp. 511–515.

[118] Věra Kůrková. “Kolmogorov’s theorem and multilayer neural networks”. In: *Neural Networks* 5.3 (1992), pp. 501–506.

[119] Bernhard Moser. “Sugeno controllers with a bounded number of rules are nowhere dense”. In: *Fuzzy Sets and Systems* 104.2 (1999), pp. 269–277.

[120] Domonkos Tikk. “On nowhere denseness of certain fuzzy controllers containing prerestricted number of rules”. In: *Tatra Mountains Math. Publ* 16.1 (1999), pp. 369–377.

[124] Syed Afaq Ali Shah, Mohammed Bennamoun, and Farid Boussaid. “A novel feature representation for automatic 3D object recognition in cluttered scenes”. In: *Neurocomputing* 205 (2016), pp. 1–15.

[125] Hsi-Jian Lee and Hsi-Chou Deng. “Three-frame corner matching and moving object extraction in a sequence of images”. In: *Computer Vision, Graphics, and Image Processing* 52.2 (1990), pp. 210–238.

[126] A. Branca, E. Stella, and A. Distante. “Feature matching constrained by cross ratio invariance”. In: *Pattern Recognition* 33.3 (2000), pp. 465–481.

[127] G. Kertész, S. Szénási, and Z. Vámossy. “Multi-Directional Image Projections with Fixed Resolution for Object Matching”. In: *Acta Polytechnica Hungarica* 15.2 (2018), pp. 211–229.

[128] Huan Xun Li, Jun Jie Shen, and Shuai Guo. “Research on Navigation and Obstacle Avoidance Algorithm for Autonomous Mobile Robot in Narrow Area”. In: *Functional Manufacturing Technologies and Ceeusro II*. Vol. 464. Key Engineering Materials. Trans Tech Publications Ltd, Apr. 2011, pp. 204–207.

[129] Nicola Krombach, David Droeschel, Sebastian Houben, and Sven Behnke. “Feature-based visual odometry prior for real-time semi-dense stereo SLAM”. In: *Robotics and Autonomous Systems* 109 (2018), pp. 38–58.

[130] Marc Ebner. “Extraction of Moving Objects with a Moving Mobile Robot”. In: *IFAC Proceedings Volumes* 31.3 (1998). 3rd IFAC Symposium on Intelligent Autonomous Vehicles 1998 (IAV’98), Madrid, Spain, 25-27 March, pp. 483–488.

[131] Baofeng Guo, R.I. Damper, Steve R. Gunn, and J.D.B. Nelson. “A fast separability-based feature-selection method for high-dimensional remotely sensed image classification”. In: *Pattern Recognition* 41.5 (2008), pp. 1653–1662.

[132] Farinaz Alamiyan Harandi, Vali Derhami, and Fatemeh Jamshidi. “A new feature selection method based on task environments for controlling robots”. In: *Applied Soft Computing* 85 (2019), p. 105812.

[133] Satoshi Komada, Kousuke Kinoshita, Tatsuhiko Hirukawa, and Junji Hirai. “Image Feature based Navigation of Nonholonomic Mobile Robots with Active Camera”. In: *IFAC Proceedings Volumes* 41.2 (2008). 17th IFAC World Congress, pp. 14714–14719.

[134] C. -H. Kim, J. -Y. Lee, and J. -J. Lee. “Feature extraction method for a robot map using neural networks”. In: *Artificial Life and Robotics* 7.3 (Sept. 2003), pp. 86–90.

[135] G. Karl and G. Schmidt. “Sensor Model Based Preprocessing of 3-D Laser Range Image Data and Motion Oriented Feature Extraction for Mobile Robot Applications”. In: *IFAC Proceedings Volumes* 21.16 (1988). 2nd IFAC Symposium on Robot Control 1988 (SYROCO 88), Karlsruhe, FRG, 5-7 October, pp. 285–291.

[136] Daniel Castro, Urbano Nunes, and António Ruano. “Feature extraction for moving objects tracking system in indoor environments”. In: *IFAC Proceedings Volumes* 37.8 (2004). IFAC/EURON Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal, 5-7 July 2004, pp. 96–101.

[137] N.E. Pears. “Feature extraction and tracking for scanning range sensors”. In: *Robotics and Autonomous Systems* 33.1 (2000), pp. 43–58.

[138] H. Park, S. Lee, and W. Chung. “Obstacle Detection and Feature Extraction using 2.5D Range Sensor System”. In: *2006 SICE-ICASE International Joint Conference*. Oct. 2006, pp. 2000–2004.

[139] G. W. Snedecor and W. G. Cochran. In: *Statistical Methods*. Iowa State University Press, Ames, 1989.

[140] Mike Fritz and Paul D. Berger. “Chapter 2 - Comparing two designs (or anything else!) using independent sample T-tests”. In: *Improving the User Experience Through Practical Data Analytics*. Boston: Morgan Kaufmann, 2015, pp. 47–69.

[141] Student. “Probable error of a correlation coefficient”. In: *Biometrika* 6.2-3 (Sept. 1908), pp. 302–310.

[142] Lloyd Fisher and John Mcdonald. “3 - Two-sample t-test”. In: *Fixed Effects Analysis of Variance*. Probability and Mathematical Statistics: A Series of Monographs and Textbooks. Academic Press, 1978, pp. 21–35.

Own Publications Related to the Thesis

[NK-17] F. Kosser and N. Kumar. “Robot navigation and path planning techniques challenges: a review”. In: *International Journal of Electronics Engineering* 11.2 (2019), pp. 115–125.

[NK-95] N. Kumar, Z. Vámossy, and Z. M. Szabó-Resch. “Heuristic approaches in robot navigation”. In: *2016 IEEE 20th Jubilee International Conference on Intelligent Engineering Systems (INES)*. June 2016, pp. 219–222.

[NK-100] N. Kumar, Z. Vámossy, and Z. M. Szabó-Resch. “Robot path pursuit using probabilistic roadmap”. In: *2016 IEEE 17th International Symposium on Computational Intelligence and Informatics (CINTI)*. Nov. 2016, pp. 000139–000144.

[NK-101] N. Kumar, Z. Vámossy, and Z. M. Szabó-Resch. “Robot obstacle avoidance using bumper event”. In: *2016 IEEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI)*. May 2016, pp. 485–490.

[NK-121] N. Kumar, M. Takács, and Z. Vámossy. “Robot navigation in unknown environment using fuzzy logic”. In: *2017 IEEE 15th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*. Jan. 2017, pp. 279–284.

[NK-122] S. M. Nasti, Z. Vámossy, and N. Kumar. “Obstacle avoidance during robot navigation in dynamic environment using fuzzy controller”. In: *International Journal of Recent Technology and Engineering* 8.2 (2019), pp. 817–822.

[NK-123] N. Kumar and Z. Vámossy. “Robot navigation with obstacle avoidance in unknown environment”. In: *International Journal of Engineering & Technology* 7.4 (2018), pp. 2410–2417.

[NK-143] N. Kumar and Z. Vámossy. “Laser Scan Matching in Robot Navigation”. In: *2018 IEEE 12th International Symposium on Applied Computational Intelligence and Informatics (SACI)*. May 2018, pp. 241–245.

[NK-144] N. Kumar and Z. Vámossy. “Laser scan matching based simultaneous localization and mapping in robot navigation using fuzzy logic”. In: *13th Miklós Iványi International PhD & DLA Symposium*. Abstract book. Nov. 2017, p. 135.

[NK-145] N. Kumar and Z. Vámossy. “Robot navigation in unknown environment with obstacle recognition using laser sensor”. In: *International Journal of Electrical and Computer Engineering* 9.3 (2019), pp. 1773–1779.

[NK-146] N. Kumar and Z. Vámossy. “Obstacle recognition and avoidance during robot navigation in unknown environment”. In: *International Journal of Engineering & Technology* 7.3 (2018), pp. 1400–1404.

[NK-147] N. Kumar and Z. Vámossy. “Obstacle avoidance in robot navigation using two-sample t-test based obstacle-recognition”. In: *Proceedings of 3rd International Conference on Recent Innovations in Computing (ICRIC-2020)*. Accepted. Springer International Publishing, 2020.

A. Appendices

A.1. Simulated Turtlebot robot in Gazebo simulator

The Gazebo simulator helps to create virtual environment for the simulations. Gazebo can be used for robot simulation. It can efficiently simulate and visualize the robotic acts in a three-dimensional environment. Robot simulation using different types of robot models in indoor and outdoor environment is possible in Gazebo. Simulated Turtlebot robot model can be used in Gazebo simulator with the help of *turtlebot_gazebo* package of ROS.

A.1.1. Creating user-defined Gazebo-world

To create user defined Gazebo-world files, user can insert various models of different shapes available in the Gazebo simulator by drag and drop. Various options like: insert, delete, selection, rotation, translation etc., are available to customize the user defined Gazebo-world. User can create the Gazebo-world from scratch. However, creating user defined world from the scratch may require to define many of the Physics properties like friction. Therefore, in quick manner, the inbuilt Gazebo-world files of the *turtlebot_gazebo* package may be edited to get customized Gazebo-world so that most of the Physics properties need not to be set again.

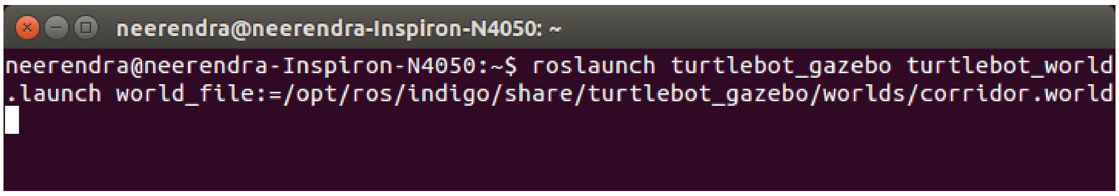
A.1.2. Pre-defined Gazebo-worlds in turtlebot_gazebo package

The *turtlebot_gazebo* package of ROS provides a convenient way to bring a simulated Turtlebot robot within the Gazebo simulator. There are three inbuilt Gazebo-world files in the *turtlebot_gazebo* package namely: *corridor.world*, *empty.world* and *playground.world.*

ROS command to launch Gazebo simulator containing simulated Turtlebot is as follows:
$ roslaunch turtlebot_gazebo turtlebot_world.launch world_file:=path

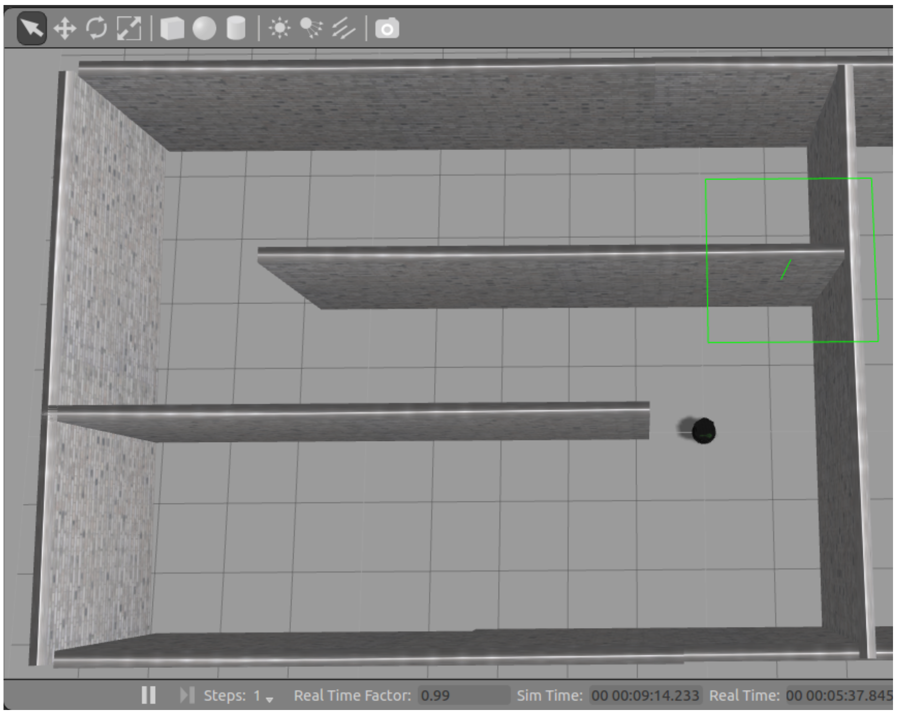
where, *Path* is the full path of the source Gazebo-world file. The last field of the command, *world_file:=path*, of the command is optional. If the parameter *world_file:=path* is not provided in the above command then the default world file, *playground.world*, will be opened. By editing the predefined Gazebo-worlds, users can create and save their own Gazebo-world files [NK-101].

Fig. A.1 shows the command to bring-up the *corridor.world* (Fig. A.2) in ROS.



```
neerendra@neerendra-Inspiron-N4050: ~
neerendra@neerendra-Inspiron-N4050:~$ roslaunch turtlebot_gazebo turtlebot_world
.launch world_file:=/opt/ros/indigo/share/turtlebot_gazebo/worlds/corridor.world
```

Figure A.1.: Command to bring-up the *corridor.world* in ROS.

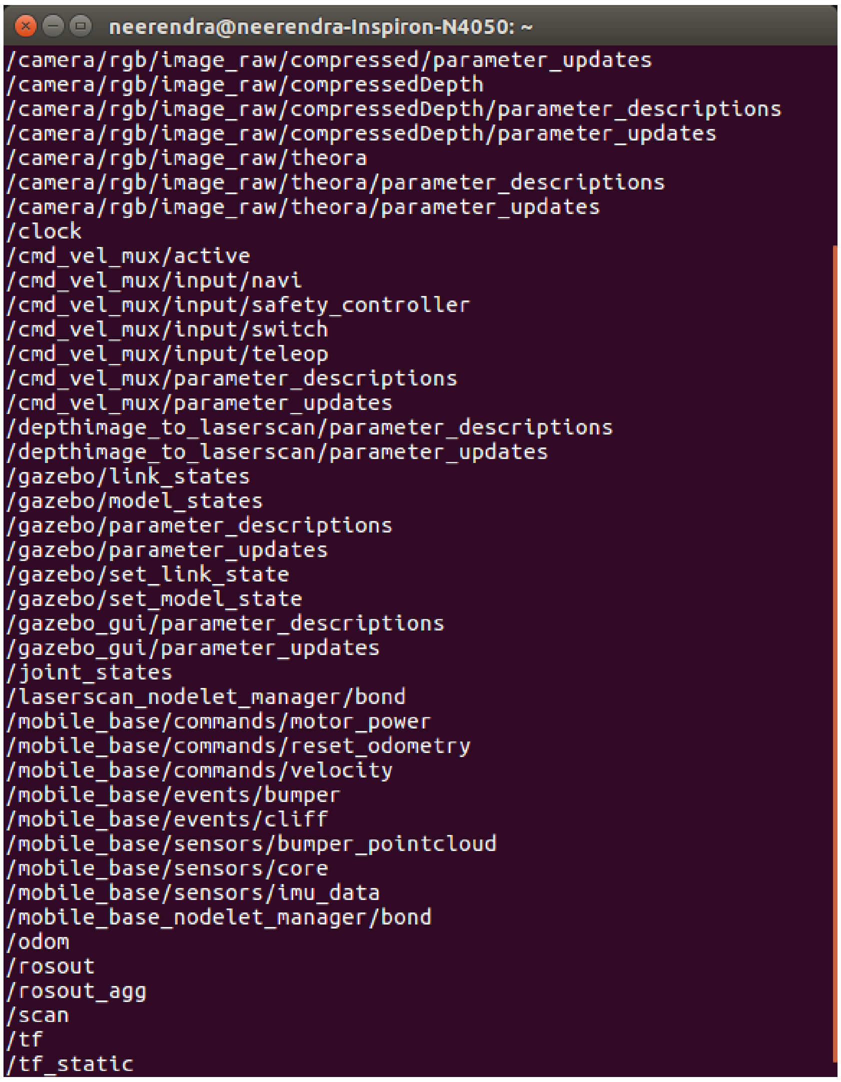


```
Steps: 1   Real Time Factor: 0.99   Sim Time: 00 00:09:14.233   Real Time: 00 00:05:37.845
```

Figure A.2.: The *corridor.world* in Gazebo.

A.2. Communication between software packages using ROS

To execute robotic operations, generally, several robot controlling software packages (i.e. nodes) need to communicate between each other. ROS provides a mechanism for the communication among these nodes. This communication mechanism is known as publisher-subscriber mechanism. In the publisher-subscriber mechanism, a node can send a message to any other node in the system on a topic of the ROS. The sender and receiver, of a message, nodes are called publisher and subscriber, respectively. A list of ROS topics is provided in Fig. A.3.


neerendra@neerendra-Inspiron-N4050: ~

```
/camera/rgb/image_raw/compressed/parameter_updates
/camera/rgb/image_raw/compressedDepth
/camera/rgb/image_raw/compressedDepth/parameter_descriptions
/camera/rgb/image_raw/compressedDepth/parameter_updates
/camera/rgb/image_raw/theora
/camera/rgb/image_raw/theora/parameter_descriptions
/camera/rgb/image_raw/theora/parameter_updates
/clock
/cmd_vel_mux/active
/cmd_vel_mux/input/navi
/cmd_vel_mux/input/safety_controller
/cmd_vel_mux/input/switch
/cmd_vel_mux/input/teleop
/cmd_vel_mux/parameter_descriptions
/cmd_vel_mux/parameter_updates
/depthimage_to_laserscan/parameter_descriptions
/depthimage_to_laserscan/parameter_updates
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/gazebo_gui/parameter_descriptions
/gazebo_gui/parameter_updates
/joint_states
/laserscan_nodelet_manager/bond
/mobile_base/commands/motor_power
/mobile_base/commands/reset_odometry
/mobile_base/commands/velocity
/mobile_base/events/bumper
/mobile_base/events/cliff
/mobile_base/sensors/bumper_pointcloud
/mobile_base/sensors/core
/mobile_base/sensors/imu_data
/mobile_base_nodelet_manager/bond
/odom
/rosout
/rosout_agg
/scan
/tf
/tf_static
```

Figure A.3.: An instance of ROS topics list.

For navigation related tasks, the */mobile_base/commands/velocity*, */scan*, and */odom* ROS topics are useful in following manner:

(i) The linear and angular velocities commands can be given to the robot by publishing *geometry_msgs/Twist* message on the topic */mobile_base/commands/velocity* of ROS.

(ii) The robot publishes its LASER sensor scan ranges-vector on */scan* topic in ROS.

(iii) The robot publishes its position and orientation, in the environment, on the */odom* topic of the ROS. Therefore, the position coordinates and orientation of the robot, in the navigation environment, can be received by subscribing the ROS topic */odom*.

A.3. Pre-defined and user-defined ROS nodes

Pre-defined ROS packages and their node can be used using *roslaunch* command. For example, the *keyboard_teleop* node of *turtlebot_teleop* can be used to navigate the Turtlebot by the keyboard of the computer. The *turtlebot_teleop* node can be executed by using the command as given in Fig. A.4. In Fig. A.4, the description of the command is as follows:

roslaunch = A ROS package containing roslaunch tools.
turtlebot_teleop = A ROS package containing *keyboard_teleop.launch* file.
keyboard_teleop.launch = the launch file to bring up the keyboard tele-operator node.

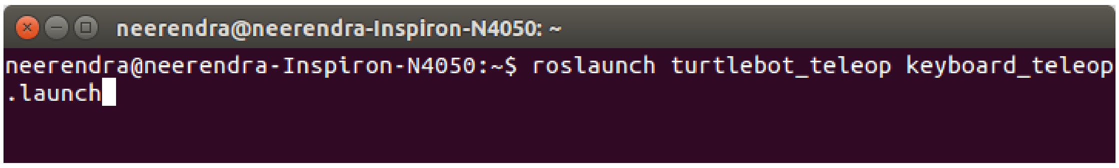
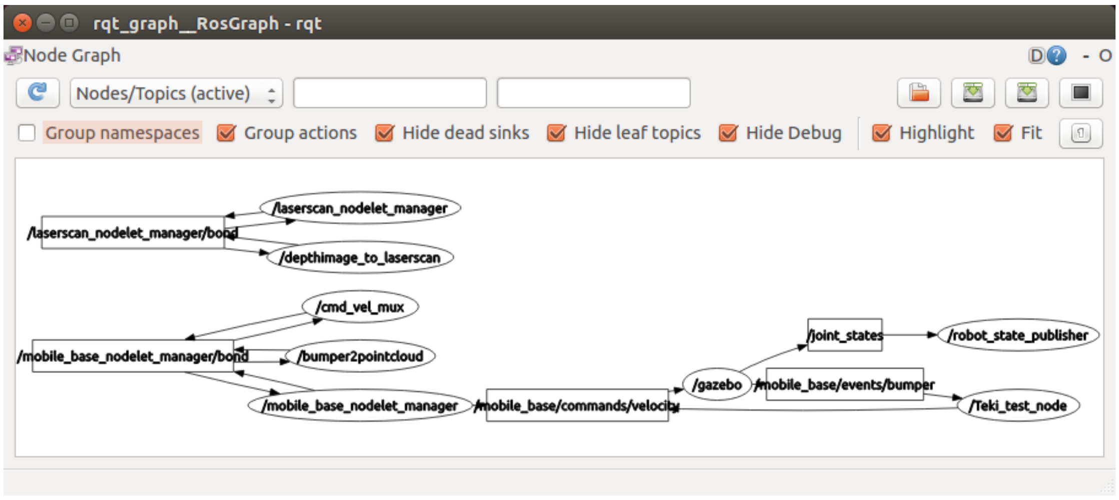

```
neerendra@neerendra-Inspiron-N4050: ~
neerendra@neerendra-Inspiron-N4050:~$ roslaunch turtlebot_teleop keyboard_teleop
.launch
```

Figure A.4.: Turtlebot tele-operator command.

The user-defined nodes can also be linked with the ROS to perform the desired operations on or by the robot by publishing or subscribing the available topics. In this way, The robot and the robot navigation model can communicate with each other through ROS using publisher-subscriber strategy. The user defined nodes can be created in the ROS using a computer programming language. In case of C/C++ programming languages, the user defined nodes can be developed with the help of *catkin* workspace in ROS. The MATLAB software also provides a convenient way to create, compile, and execute a user node in ROS.

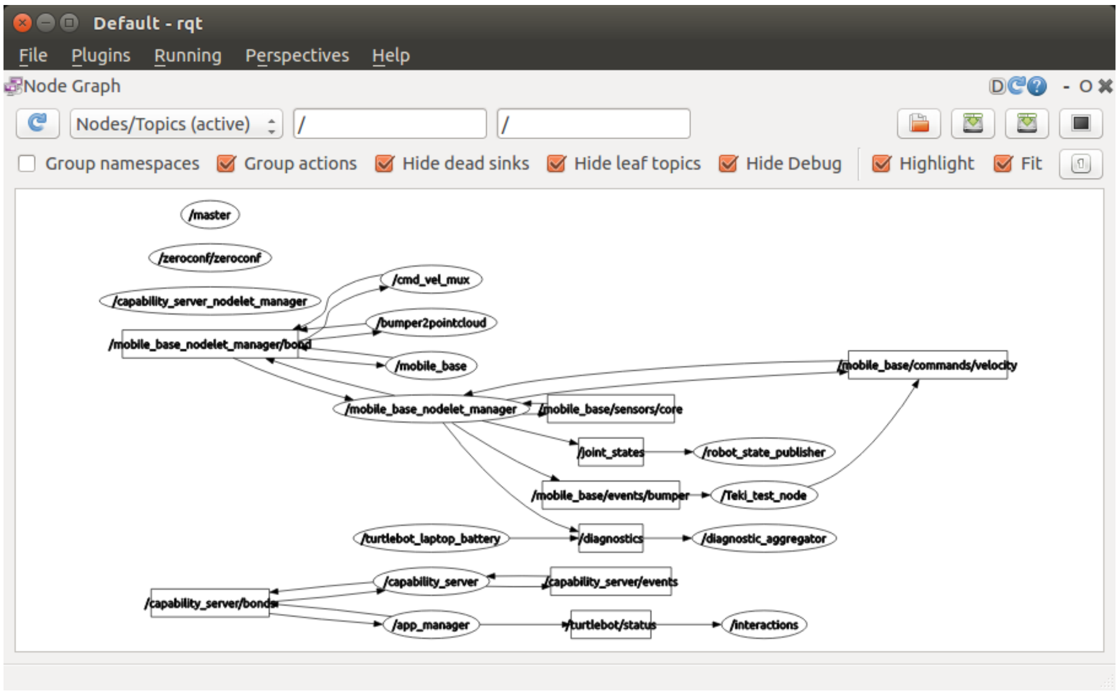
A.4. Graphical representation and execution of ROS nodes

The ROS nodes and the associated topics can be represented graphically with the help of the *rqt* package. The *rqt* is a cross-platform application framework of ROS. It implements the graphical user interface (GUI) tools. Some basic plugins of *rqt* package are: *rqt_bag, rqt_console, rqt_graph, rqt_logger_level, rqt_plot.* The *rqt_bag* provides a GUI plugin for recording and managing bag files. The *rqt_console* provides a GUI plugin to display the messages being published to *rosout* topic. The logger level of ROS nodes can be configured using *rqt_logger_level*. The *rqt_graph* is used to find the ROS graph of the system. The *rqt_plot* plugin constructs the two-dimensional plots for the given ROS topics.


rqt_graph__RosGraph - rqt
Node Graph
Nodes/Topics (active)
/laserscan_nodelet_manager
/laserscan_nodelet_manager/bond
/depthimage_to_laserscan
/cmd_vel_mux
/mobile_base_nodelet_manager/bond
/bumper2pointcloud
/mobile_base_nodelet_manager
/mobile_base/commands/velocity
/gazebo
/joint_states
/robot_state_publisher
/mobile_base/events/bumper
/Teki_test_node

☐ Group namespaces ☑ Group actions ☑ Hide dead sinks ☑ Hide leaf topics ☑ Hide Debug ☑ Highlight ☑ Fit

Figure A.5.: The *Teki_test_node* in case of the Gazebo Simulator.



☐ Group namespaces ☑ Group actions ☑ Hide dead sinks ☑ Hide leaf topics ☑ Hide Debug ☑ Highlight ☑ Fit

Figure A.6.: The *Teki_test_node* in case of real robot.

A user defined node (named as *Teki_test_node*) has been developed to implement the algorithm proposed in Section 4.1.1.

In case of Gazebo simulator, the node graph of the system is presented in Fig. A.5. Similarly, Fig. A.6 shows the node graph of the system in the case of real Turtlebot. In the node graph, the nodes are represented by ellipses and the topics are enclosed by rectangles. An arrow, directed from a topic to a node, represents that the topic is subscribed by the node. On the other hand, an arrow pointing towards a topic from a node shows that the topic is published by the node.

The velocity commands to move the Turtlebot can be published in the form of geometry message: *Twist* on the ROS topic */mobile_base/commands/velocity*. Our node, *Teki_test_node*, is publisher on the topic */mobile_base/commands/velocity*. In "C++", The publisher instruction is as follows:

ros::Publisher pub=nh.advertise<geometry_msgs::Twist>...

...("/mobile_base/commands/velocity",1000);

where, *pub* is user defined name of *Publisher* object, the *nh* is user defined node handle and 1000 is the user defined buffer size. The bumper event can be subscribed in the form of Kobuki message *BumperEvent* using the ROS topic */mobile_base/events/bumper*. The topic */mobile_base/events/bumper* can be subscribed using the ROS command:

ros::Subscriber bumper = nh.Subscribe("/mobile_base/events/bumper", 1, callback);

where, *callback* is the user defined name of the callback function, bumper is user defined name of the object, 1 is user defined buffer size and *nh* is the user given name of the node handle. The callback function of the subscriber can be called by using the ROS function *ros::spinOnce()* or *ros::spin();*. The *ros::spin()* function blocks the execution of the program only for callback function call and in this situation the publisher related task cannot be executed outside the definition of the callback function. To publish the messages outside of the callback function definition, we can use *ros::spinOnce()* for callback function call. In this study, the *ros::spinOnce()* function is used for the callback function call, so that, the twist messages can be published outside the callback function definition as well.

A.4.1. Executing user-defined node with simulated robot

The following two main steps are needed to execute our node (*Teki_test_node*) in the Gazebo:

(i) Bring up Turtlebot in our own Gazebo-world using the ROS command in a terminal.

(ii) In a new terminal execute the following series of commands:

$ cd catkin_ws (to enter into the *catkin_ws* directory)
$ catkin_make (to compile our code)
$ source devel/setup.bash (to set environment variables)
$ rosrun nkt Teki_test_node (to execute our node, here *nkt* is name of user created package)

A.4.2. Executing user-defined node with real robot

The same C++ code is applicable to the real Turtlebot without any change. The following two main steps have to be performed to executed our node (*Teki_test_node*) in the system with real robot:

(i) Bring up the real Turtlebot in the system using the ROS command in a terminal.

(ii) In a new terminal execute the same series of commands as given above in step (ii) of the Section A.4.1 for Gazebo simulator.

However, if we are using the same terminal which we have used in step (ii) with Gazebo then we will already be in the catkin working directory. Further, there is no change in the C++ code, so, we do not need to compile the code again. Furthermore, the environment variables are already set. Therefore, if we are using the same terminal in which we have executed our node previously then the following one command is sufficient:
$ rosrun nkt Teki_test_node
where, *nkt* is the name of our package and *Teki_test_node* is the name of our node.

A.5. Map of the navigation environment

A.5.1. Building map of the environment using ROS package

LASER-based solution is provided by the *gmapping* package of ROS. The gmapping package can be executed as a ROS node: *slam_gmapping*. The environment map can be built and saved by taking the following steps:

(i) In a Ubuntu terminal, execute the *slam_gmapping* ROS node using the following command for real Turtlebot or simulated Turtlebot in Gazebo simulator.
For real Turtlebot:
$ roslaunch turtlebot_navigation gmapping_demo.launch
For simulated Turtlebot in Gazebo simulator:
$ roslaunch turtlebot_gazebo gmapping_demo.launch

(ii) In a new Ubuntu terminal, execute *rviz*, a three-dimensional visualization tool of ROS, to visualize the map building process. The following command brings up the *rviz*:
$ roslaunch turtlebot_rviz_launchers view_navigation.launch

(iii) Save the map for future use. For this, the following command is required to be executed in a new terminal of Ubuntu:
$ rosrun map_server map_saver -f Path

Where, *Path* is the full path of the destination file. Now, close all the Ubuntu terminals opened above. The map file is saved with *.yaml* file name extension. The *slam_gmapping* generates two-dimensional occupancy grid map using laser and pose data of mobile robot.

A.5.2. Turtlebot's autonomous navigation with given map

The prior saved map can be used to help the autonomous navigation of mobile robot. The following are the steps for the real Turtlebot and in Gazebo simulator:

(i) In a Ubuntu terminal, start the Turtlebot using the following command.
For real Turtlebot:
$ roslaunch turtlebot_bringup minimal.launch
The *turtlebot_bringup* gives roslaunch scripts for starting the base functionality of the TurtleBot.
For simulated Turtlebot in Gazebo simulator:
$ roslaunch turtlebot_gazebo turtlebot_world.launch Path

Fig. A.2 shows the visualization of an inbuilt Gazebo-world, *corridor.world*, of the *turtlebot_gazebo* package. To exemplify the practical effects of using different distance metrics in the A* algorithm, the relatively simple environment visualized in Fig. A.2 was chosen because the outcomes of using different distance metrics can be received quickly. In addition, the *corridor.world* has at least two compulsory turns for any path from bottom left corner to top right corner. In this way, the performance of the algorithms can also be observed at the turns.

(ii) Open a new terminal in Ubuntu. The following command passes the generated and saved map file to the Turtlebot.
For real Turtlebot:
$ roslaunch turtlebot_navigation amcl_demo.launch Path
For simulated Turtlebot in Gazebo simulator:
$ roslaunch turtlebot_gazebo amcl_demo.launch Path

where, *Path* is the map file's path. The default map of *playground.world* will be passed if we do not provide the *Path* in the command in this step. There is an alternative method to provide the value to the *Path* argument of this command. In this alternative method, we can configure *TURTLEBOT_MAP_FILE* environment variable using the following command:
$ export TURTLEBOT_MAP_FILE = Path

(iii) Open *rviz* in the new terminal by using the following command:
$ roslaunch turtlebot_rviz_launchers view_navigation.launch
The map of the environment, passed by Step (ii) above, will be shown in *rviz*.

A.5.3. Creating user-defined global planner

The following steps are required to create an user-defined global planner to navigate the simulated Turtlebot in Gazebo simulator:

(i) Creating a new package in catkin (the build system of ROS) workspace. In general terms, the catkin workspace is a folder inside computer where we can build, install, and modify catkin packages.

(ii) Writing the header file and path planner class in the *src* folder of this package adhering to the *BaseGlobalPlanner* interface of the *nav_core* package of the ROS.

(iii) Registering our planner as *BaseGlobalPlanner* plugin by writing the following statement in the *.cpp* file of path planner class:
PLUGINLIB_EXPORT_CLASS...
...(our planner classname, nav_core::BaseGlobalPlanner)
For the above statement, we also need to include *pluginlib/class_list_macros.h* header file.

(iv) Adding the global planner library to the *CmakeLists.txt* file of our package.

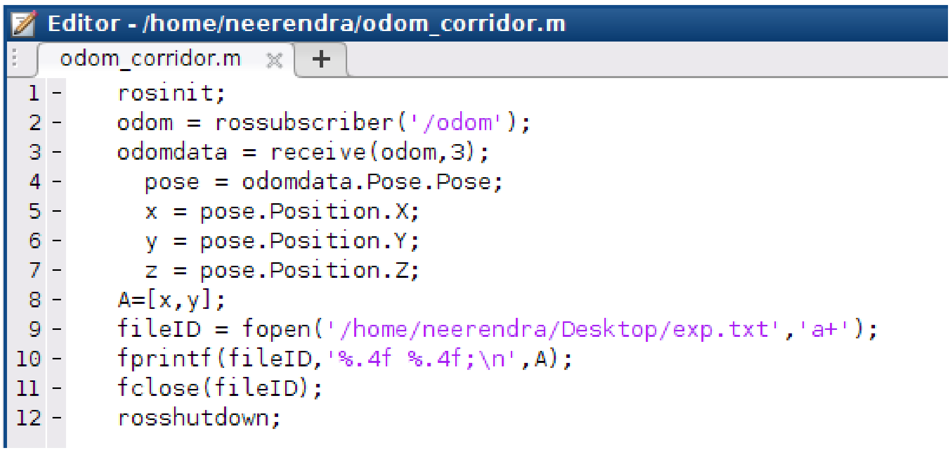
(v) Creating a plugin description file with *.xml* filename extension.

(vi) Registering plugin with ROS package system. We can verify that whether our planner is available now in nav_core package of ROS by executing the following command in the Ubuntu terminal:
$ rospack plugins –attrib=plugin nav_core
If everything is fine our global planner will be displayed in the list of all available planners in the nav_core package.

(vii) Open the *move_base.launch.xml* file of the Turtlebot and add the following statement:
<param name="base_global_planner" value = "our global planner class name"/>

A.6. Writing robot positions in a text file

Fig. A.7 provides a MATLAB code for writing the positions of robot in a text file. The descriptions for the programming instructions, of Fig. A.7, are given in Table A.1.



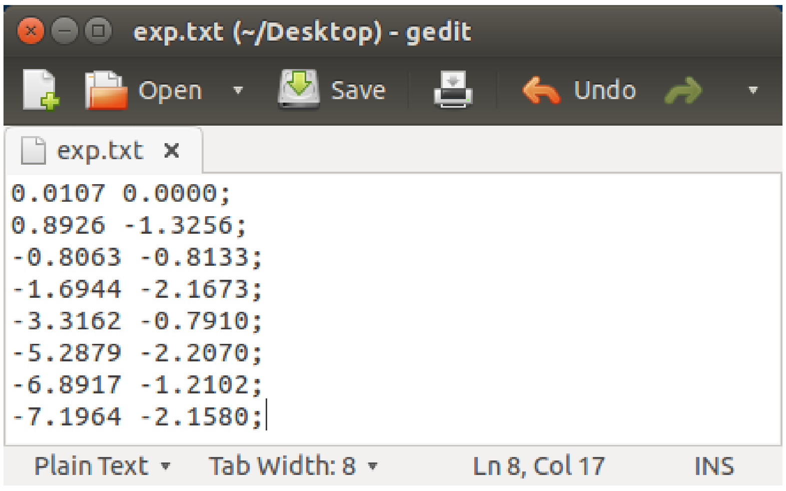
```
 1 -   rosinit;
 2 -   odom = rossubscriber('/odom');
 3 -   odomdata = receive(odom,3);
 4 -     pose = odomdata.Pose.Pose;
 5 -     x = pose.Position.X;
 6 -     y = pose.Position.Y;
 7 -     z = pose.Position.Z;
 8 -   A=[x,y];
 9 -   fileID = fopen('/home/neerendra/Desktop/exp.txt','a+');
10 -   fprintf(fileID,'%.4f %.4f;\n',A);
11 -   fclose(fileID);
12 -   rosshutdown;
```

Figure A.7.: Program to receive odometer information from robot.

In our work, the text file containing some of the initial way-points are depicted in Fig. A.8. Each row of the data in Fig. A.8 contains x and y coordinates (of a robot position) separated by a space, respectively.

Table A.1.: Description of the program instructions used in Fig. A.7.

Instruction(s)	**Description**
1	starts the global ROS node.
2	subscribes the '*/odom*' topic of ROS.
3	receives robot position using the message on '*/odom*'.
4 – 7	find x, y, z coordinates of the current position of the robot.
8	stores x, y coordinates of the current position of the robot.
9 – 11	file operations to write the robot position in a text file.
12	shuts down the global ROS node.



```
0.0107 0.0000;
0.8926 -1.3256;
-0.8063 -0.8133;
-1.6944 -2.1673;
-3.3162 -0.7910;
-5.2879 -2.2070;
-6.8917 -1.2102;
-7.1964 -2.1580;
```

Figure A.8.: Data file of way-points received from program given in Fig. A.7.

A.7. Creating occupancy grid map using MATLAB

The following instructions create an occupancy grid:

map = robotics.OccupancyGrid(W, H, R)

where, *W*, *H* and *R* represent width (in meters), height (in meters) and grid resolution per meter respectively. This instruction creates a probability occupancy grid. The following MATLAB instruction can be used to insert laser scan reading of robot in occupancy grid:

insertRay(map, pose, ranges, angles, maxrange)

where,

map = occupancy grid.

pose = [robot's current position, robot's current orientation].

ranges = length of the laser beam received in scan.

angles = angle of the laser beam received in scan.

maxrange = maximum range of the laser scan.

In the map building process, the number of way-points to be travelled and the *maxrange* parameter of *insertRay* have to be adjusted for the considered environment. It is observed that large number of way-points needs smaller *maxrange* and smaller number of way-points suits with larger *maxrange*. For the way-points taken as in Fig. 3.3, one of the appropriate *maxrange* is 2.5 meters.

The following lines of MATLAB code inflate the map:

robot_radius = 0.20;

$map\_Inflated = copy(map);$
$inflate(map\_Inflated, robot\_radius);$
The inflated map is now saved in the object variable *map_Inflated*. For the future use, this occupancy grid map can be saved as a *.mat* (a MATLAB file format) file.

A.8. Generating probabilistic roadmaps in MATLAB

To find the probabilistic roadmaps, the occupancy grid map of the environment can be imported to the MATLAB workspace by specifying the full path of the *.mat* file as follow:
$filePath = `/home/neerendra/nkcorridor.mat';$
and then followed by the load instruction as below:
$load(filePath);$
where, */home/neerendra/nkcorridor.mat* is the full path of *nkcorridor.mat* in computer system.
Probabilistic roadmaps, within the inflated map, can be generated using the following MATLAB instruction:
$prm = robotics.PRM(map\_Inflated);$
The following line of instruction can find the desired path:
$path = findpath(prm, startpoint, goalpoint);.$
where, *startpoint* and *goalpoint* are the coordinates of the start location and goal location of the path.

A.9. Critical values of t for t-test

For different values of degree of freedom $\left(d_f\right)$, the critical values of t (t_c), used for the t-test (Section 5.3), are given in Table A.2. In the Table A.2, t_1 and t_5 represent t_c at 1% and 5% levels of significance, respectively [139, 141, 142].

Table A.2.: Critical values of t at 1% and 5% levels of significance.

d_f	t_1	t_5	d_f	t_1	t_5
1	63.66	12.71	19	2.86	2.09
2	9.92	4.30	20	2.84	2.09
3	5.84	3.18	21	2.83	2.08
4	4.60	2.78	22	2.82	2.07
5	4.03	2.57	23	2.81	2.07
6	3.71	2.45	24	2.80	2.06
7	3.50	2.36	25	2.79	2.06
8	3.36	2.31	26	2.78	2.06
9	3.25	2.26	27	2.77	2.05
10	3.17	2.23	28	2.76	2.05
11	3.11	2.20	29	2.76	2.05
12	3.06	2.18	30	2.75	2.04
13	3.01	2.16	40	2.70	2.02
14	2.98	2.14	60	2.66	2.00
15	2.95	2.13	80	2.64	1.99
16	2.92	2.12	100	2.63	1.98
17	2.89	2.11	120	2.62	1.98
18	2.88	2.10	> 120	2.58	1.96