SENSOR FUSION FRAMEWORK AND SIMULATION ON A TURTLEBOT ROBOTIC VEHICLE

by

Shruti Gangadhar

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Electrical Engineering

Charlotte

2017

Approved by:

____
Dr. James Conrad

____
Dr. Abasifreke Ebong

____
Dr. Ronald Sass

ABSTRACT

SHRUTI GANGADHAR. Sensor fusion framework and simulation on a TurtleBot robotic vehicle. (Under the direction of DR. JAMES M. CONRAD)

Autonomous robots are being designed and used in a wide variety of environments and unforeseen ways. Intelligent interpretation of sensor data is the key factor in a robot's decision making ability while operating in such environments. Current day robots are also expected to adapt, have human interaction and accomplish many more tasks, than ever before. This work focuses on developing a sensor fusion framework and achieving human robot interaction for a robot navigating an unknown terrain.

The selective sensor fusion framework proposed in this work, combines heterogeneous data from sensors to analyze the parameters of the environment required to accomplish a task. It is scalable, modular and accounts for human interaction. The framework uses fusion algorithms and develops confidence at various stages before arriving at a decision. The framework is simulated and tested on a robotic vehicle called the TurtleBot along with additional LIDAR (Light detection and ranging) sensors on a simulated, unknown terrain. The vehicle navigates successfully through the terrain, avoiding obstacles and terrain irregularities while querying the user for assistance when it enters an uncertain mode. The framework is designed to accommodate higher levels of decision making before deciding on an action. Furthermore, the framework can be customized based on the application.

ACKNOWLEDGEMENTS

I thank Dr. James Conrad for the guidance offered throughout the academic program and for the support and advice offered in the completion of this graduate thesis. I thank Dr. Abasifreke Ebong for the positive encouragement offered throughout the program and for accepting the position as my committee member. I thank Dr. Ronald Sass for the interesting and engaging course lessons and for willing to be a member on my thesis committee.

I thank my colleague, Balasubramaniyan Chandrasekaran for his insights on the idea of this thesis and for the support offered throughout the completion of this thesis. I thank my colleagues, Samuel Shue and Benjamin Rhoades for all their help and support.

I thank my husband, Swaroop, for believing in me and for supporting me in the pursuit of education. I dedicate this work to the memory of my grandfather, S. Subbarao, a teacher, a source of inspiration.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
D-S	Dempster Shafer
HRI	Human Robot Interaction
LIDAR	Light Detection And Ranging
ROS	Robot Operating System
SDF	Simulation Description Format
TTF	Track-to-track fusion

CHAPTER 1: INTRODUCTION

1.1 Motivation

Autonomous robotics is an important branch of robotics and is widely researched and reported [1]. In this branch of robotics, continuous information is obtained by the sensors regarding the state of the environment and processed in real-time. Living beings process multiple sensory data to reason and act. The same principle is applied in multi-sensor data fusion for robotics. It plays a crucial role in enabling the robot to make decisions and act. Significant applications of multi-sensor fusion can be found in mobile robots [2] [3] [4] [5]. A variety of other applications take the benefit of sensor fusion such as defense systems (such as target tracking [2] [6] [7] [8]), medicine [9] [10], transportation systems [11] [12] and industry [13] [14] [15].

A detailed account of data sensor fusion for autonomous robotics is found in [1]. Navigation is one of the challenging tasks in an autonomous robot development. Obstacle detection and estimation is the key factor for the success of navigation algorithms. This work attempts to address that with the help of fusion algorithms. Most of the fusion algorithms are utilized to address a specific issue in the decision-making process such as noise filtering. There is a need to investigate the end-to-end approaches further. Also, the fusion architectures described in Section 2.5 are derived from applications other than robotics. Although these techniques form excellent guidelines for sensor fusion in robotics, there is a need to explore frameworks that employ fusion algorithms so that the decision-making process is scalable.

This work highlights the role of multi-sensor fusion in robot cognition while accommodating better interaction level for human robot collaboration. The decision framework takes into account data streamed by a group of sensors and applies fusion algorithm to estimate the state of the surrounding with a higher confidence level. The framework then offers decisions to the robot that are needed to execute the task. In an event of uncertainty, the robot can query the user for assistance before executing any further action demonstrating key interaction with the human.

This work is intended to be integrated with the “ZapataBot” project of the Embedded Systems and Autonomous Robotics Lab at UNC-Charlotte [16].

From self-driving cars to intelligent robots in smart factories, sensors are heavily relied upon. Sensors have narrow window of measurement, they can be faulty, can be misguided and one sensor cannot reveal all the information about the surroundings. Sensor fusion is the way forward and such frameworks that accomplish the fusion, will be sought after.

1.2 Problem Statement

The problem statement of this work is formulated into the following parts:

1. Develop the selective sensor fusion framework.
2. Create the robot model with sensors for simulation.
3. Create the environment and terrain for the robot.

4. Integrate the fusion framework with lower level robot movement and communication libraries.
5. Test the framework by verifying robot’s ability to navigate the unknown terrain.

1.3 Contributions

A list of main contributions of this thesis work are as follows:

1. Implemented Kalman filter and Track-to-track fusion algorithm to achieve signal level fusion to aid in robotic navigation.
2. Designed and implemented the terrain classifier.
3. Developed the robot model along with its sensors using SDF for simulation on Gazebo.
4. Integrated the framework package with ROS.
5. Designed the test environment for the robot to navigate.

1.4 Organization of Thesis

Chapter 1 introduces sensor fusion and provides an overview of the fusion concept in autonomous and robotic systems. Chapter 2 details the need for sensor fusion and some of the fusion methodologies and architectures found in literature that are considered for this work. Chapter 3 introduces the selective sensor fusion framework and the design of the framework. Chapter 4 focuses on the implementation of the framework. The tools and libraries used to develop the framework are also discussed. Chapter 5 explains simulation and testing. Chapter 6 discusses the advantages of the proposed selective fusion framework and the scope for future work.

CHAPTER 2: SENSOR FUSION OVERVIEW

Sensor fusion is a versatile topic with diverse areas of application. Fusion offers a more unified view of the environment. Multi-sensor data fusion is the process of combining information from individual sensors to obtain more inferences than that can be derived from a single sensor. Multi-sensor fusion plays a crucial role in developing robotic systems because the interaction with the environment is instrumental in successful execution of the task.

2.1 Sensor Data Acquisition

Data acquired by sensors are in a particular format depending on the type of sensor. Various sensors can be set to acquire data at a specified rate. Some sensors are simple, such as a temperature sensor that outputs a temperature reading (a single number). There are other sensors such as the LIDAR and camera that output a complex data structure. The data format, the rate at which sensors publish data and other auxiliary information such as the spatial sector of the environment being measured by the sensor are all key elements to know, for applying the fusion process. Figure 1 describes a LIDAR sweep and Appendix D Section D1 has the data structure for the same.

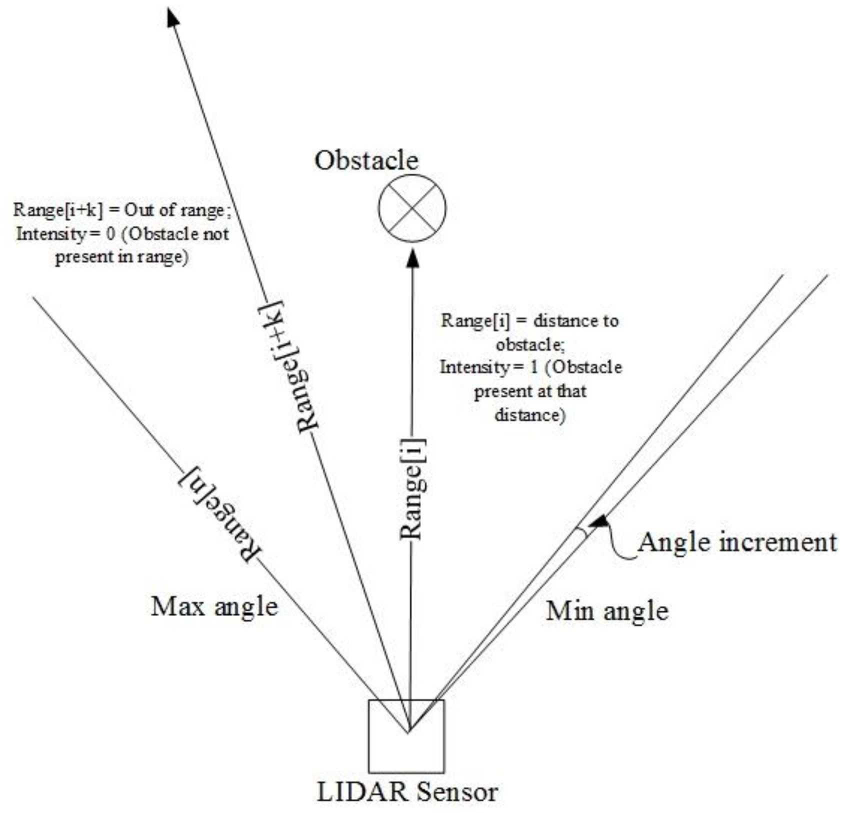

Obstacle
Range[i+k] = Out of range; Intensity = 0 (Obstacle not present in range)
Range[i] = distance to obstacle; Intensity = 1 (Obstacle present at that distance)
Range[n]
Range[i+k]
Range[i]
Angle increment
Max angle
Min angle
LIDAR Sensor

Figure 1: LIDAR data

2.2 Need for Sensor Fusion

In several current day applications ranging from surveillance, object detection, target tracking to navigation, sensing systems are generating higher volumes of data at a rapid rate. It is imperative to combine the data obtained from individual sensors to extract the most useful information to effectively optimize storage and utilization of data [17]. Multi-sensor data fusion also aims at combining complementary and redundant information. Reliable decision making requires the data obtained by the sensors to be accurate. However, sensors are prone to errors and sensor data fusion can improve confidence and reliability in the decision.

Several sensors measuring one or more attributes of their surroundings can increase temporal and spatial coverage. In distributed sensor nodes, it is advantageous to reduce the frequency of communication and utilization of bandwidth. This can result in increased life of equipment, reduction in collision of data and increase in energy savings. By fusing the sensor data, only the result can be transmitted and the above-mentioned advantages can be realized [18].

The main goal of multi-sensor fusion is to achieve better operation of the system using the collective information from all sensors. This is also referred to as the synergistic effect [19] [20] [21]. Combining the data from a single sensor at different time intervals can also produce this effect [21] [22]. To have better spatial and temporal coverage, multiple sensors can be used. Also, with multiple sensors there is increased estimation accuracy and fault-tolerance [21].

2.3 Sensor Fusion Challenges

Several factors make the sensor fusion process a challenging one. The majority of problems arise from imperfection in sensor readings, variety of sensor technologies, limitations imposed by the type of application and environment. These problems are explained in detail in [18] and is briefly summarized as follows:

1. Data imperfection: Data from the sensors contain some amount of noise and imprecision. Data fusion algorithms should be able to take advantage of the redundant data to minimize the effects of such imperfections.

2. Outliers and spurious data: Ambiguity and inconsistencies in the environment that the sensors may not be able to distinguish, causing the measured data to be unreliable. Such data appear as outliers in the data set.
3. Conflicting data: If two sensors are offering conflicting data about an aspect under observation, the fusion algorithm should be able to handle such conflicts to avoid counter-intuitive results.
4. Data modality: The fusion process must take into consideration both qualitatively similar (homogeneous) and different (heterogeneous) sensor data.
5. Data correlation: When sensors are spatially distributed in a system, some sensor nodes are prone to external disturbances. This can bias the sensor readings and the fusion result may suffer from over/under confidence.
6. Data alignment: Data from various sensors must be brought to a common frame of reference before the fusion process. It deals with the calibration error induced by individual sensors.
7. Operational timing: The data used for fusion may be coming from sensors that span a vast area or from sensors that are generating data at different rates. Out-of-sequence arrival of data for fusion process can result in performance degradation especially in real time applications.

2.4 Categories of Sensor Fusion

Depending upon the sensor configuration, there are three main categories of sensor fusion: Complementary, Competitive and Co-operative [23]. These are described below as follows:

1. Complementary: In this method, each sensor provides data about different aspects or attributes of the environment. By combining the data from each of the sensors we can arrive at a more global view of the environment or situation. Since there is no dependency between the sensors combining the data is relatively easy [23] [24].
2. Competitive: In this method, as the name suggests, several sensors measure the same or similar attributes. The data from several sensors is used to determine the overall value for the attribute under measurement. The measurements are taken independently and can also include measurements at different time instants for a single sensor. This method is useful in fault tolerant architectures to provide increased reliability of the measurement [23] [24].
3. Co-operative: When the data from two or more independent sensors in the system is required to derive information, then co-operative sensor networks are used since a sensor individually cannot give the required information regarding the environment. A common example is stereoscopic vision [23] [24].

Several other types of sensor networks exist such as corroborative, concordant, redundant etc [21]. Most of them are derived from the aforementioned sensor fusion categories.

2.5 Fusion Methodologies

Generally, sensor fusion framework is designed or chosen based on the application. However, there are various methodologies to capture the general idea of fusion and set up guidelines. Some of the methodologies are discussed below [18].

2.5.1 JDL Model

JDL stands for the US Joint Directors of Laboratories that was established under the guidance of Department of Defense and was proposed in 1985. The JDL model is functionality dependent and can be customized depending on the application. Varieties of applications from sensor networks to human robot interface can be implemented using this model [24]. JDL model is derived from the military domain. It is based on the input data and processed output.

The model uses five levels for data processing and a database. These components can communicate through a bus interface [24] [25] [26]. The JDL model is as shown in Figure 2 [25] [26]. These levels could be executed sequentially or concurrently during the application.

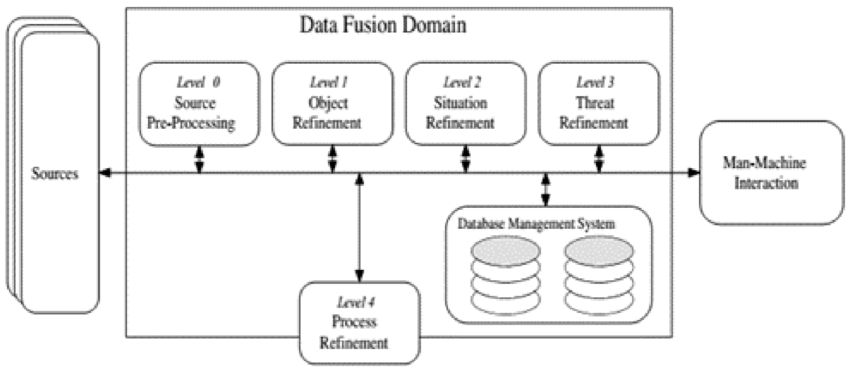

Data Fusion Domain
Level 0 Source Pre-Processing
Level 1 Object Refinement
Level 2 Situation Refinement
Level 3 Threat Refinement
Sources
Man-Machine Interaction
Database Management System
Level 4 Process Refinement

Figure 2: JDL fusion model [25] [26]

Sources, in the JDL model can consist of sensor data or data given by the user such as user input, reference data or geographical data. The Man-Machine Interaction block, as the name suggests, enables the user to interact with the system through user command, reports etc. Furthermore, this block helps in providing alert messages and could use multimedia tools such as displays, sounds etc. to achieve communication with the user.

The Source Pre-Processing also referred to as Level 0, performs pre-screening of data and then allocates it to the appropriate process [25] [26]. In the Object Refinement or Level 1, the following operations are performed namely, alignment of data using frame transformation, data association, tracking and estimation of the current and future position of the object. Also, Level 1 can be composed of kinematic and identity fusion [24]. In kinematic fusion, the velocity, acceleration of the object is determined. In identity fusion, the type of the object such as aircraft or missile is determined using parametric estimation [24] [25]. After processing the data from Level 1, based on the situation the contextual relationship is determined between the event and the object under observation. This process of refinement is called as Situation Refinement or Level 2. Depending on the a priori data and the future situation prediction inferences are drawn in Level 3 or Threat Refinement. The inferences are used to identify the vulnerabilities and the opportunities for the operation. This level uses game theoretic techniques [25]. Process Refinement or Level 4 deals with monitoring the system performance (handles real time constraints) and sensor allocation to satisfy mission objectives and goals. This level does not perform data processing operations and uses sensor management techniques [24] [25] [26]. The Database Management System helps monitor, update, add and provide information to the fusion process [24] [25] [26].

Although the JDL model helps in basic understanding of the sensor fusion process it is data centric and hence hard to extend or reuse the applications based on this model. The JDL model is too restrictive and tuned to the military. It is abstract and interpretation could be difficult [25] [26]. It focuses more on input and output rather than processes. Table 1 highlights the summary of various components used in JDL model.

Table 1: Summary of JDL process components [25]

SOURCES	Can include data from sensors to a priori information from databases to human input.
PROCESS ASSIGNMENT	Enables the data fusion process to concentrate on the data most pertinent to the current situation as well as reducing the data fusion processing load. Involves data pre-screening and allocating data to appropriate processes.
OBJECT REFINEMENT I (Level 1)	Transforms data to a consistent reference frame and units and estimate or predict object position, kinematics, or attributes. Also, assigns data to objects to allow statistical estimation and refine estimates of the objects identity or classification.
SITUATION REFINEMENT (Level 2)	Describes of the relationship between objects and observed events. This processing determines the meaning of a collection of entities and accounts for environmental information, a priori knowledge, and observations.
THREAT REFINEMENT (Level 3)	Projects the current situation into the future and indicates possible threats, vulnerabilities, and opportunities for operations.
PROCESS REFINEMENT (Level 4)	Monitors real-time performance of data-fusion, identifies information required for data fusion improvement. Also, allocates and directs sensor and sources to achieve mission goals.
DATABASE MANAGEMENT SYSTEM	Most extensive ancillary function required to support data fusion. Also features data retrieval, storage, archiving, compression, relational queries, and data protection.
HUMAN-COMPUTER INTERACTION	Enables human input and communication of data fusion results to operators and users, and includes methods of alerting human as well as augmenting cognition.

2.5.2 Waterfall Fusion Process Model

The Waterfall fusion process model (WFFM) deals with the low-level processing of data and is shown in Figure 3 [25] [27]. The Waterfall model has a lot of common features as the JDL model. The processing stages of the Waterfall models relate to the levels of the JDL model [25] [26] [27] and the comparison is shown in Table 2.

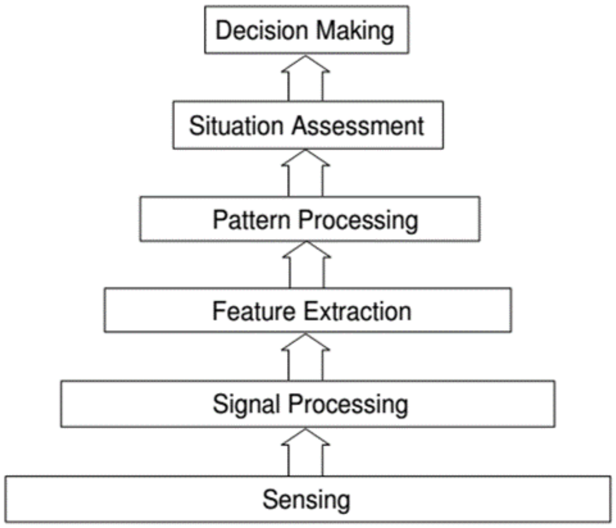


Figure 3: Waterfall fusion process model [27]

Table 2: JDL and waterfall fusion models [25] [26] [27]

JDL levels	Waterfall stages
Level 0	Sensing and Signal Processing
Level 1	Feature Extraction and Pattern Processing
Level 2	Situation Assessment
Level 3	Decision Making

However, similar to the JDL model, the Waterfall fusion model is abstract and doesn’t have feedback between the stages. It is an acyclic model. The modified WFFM is described in [24] that provides for some feedback between the stages. This modified model is action oriented and has the provision for control loop action or feedback loop as shown in Figure 4.

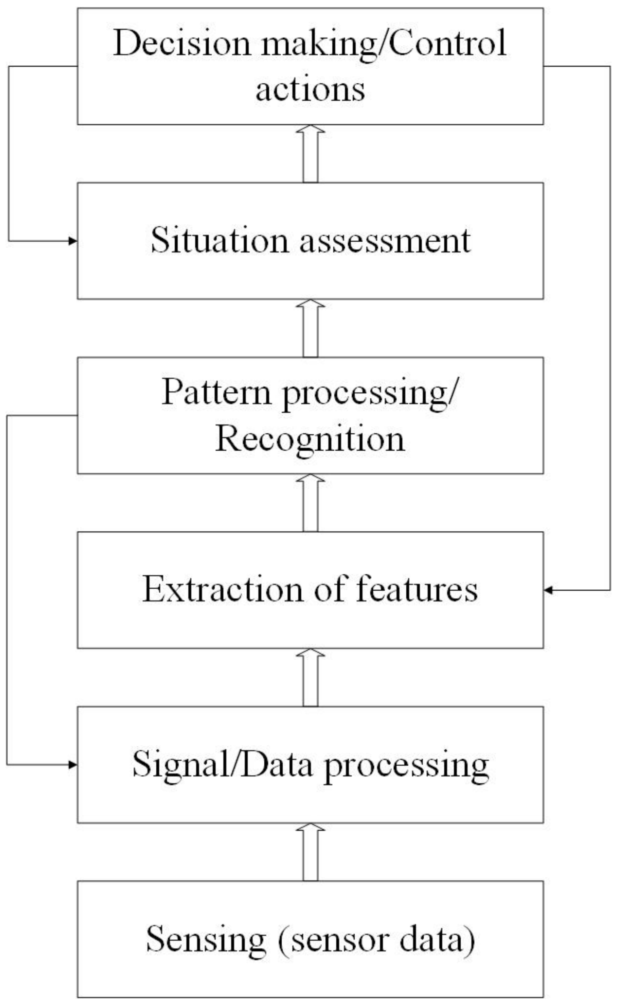


Figure 4: Modified waterfall fusion model [24]

2.5.3 Dasarathy's Classification

This method is derived from software engineering perspective where fusion is a data flow model characterized by input and output as well as functional entities (processes). Figure 5 [28] shows various sensor fusion types as classified by Dasarathy. Only a few combinations are allowed in this scheme for the inputs and outputs.

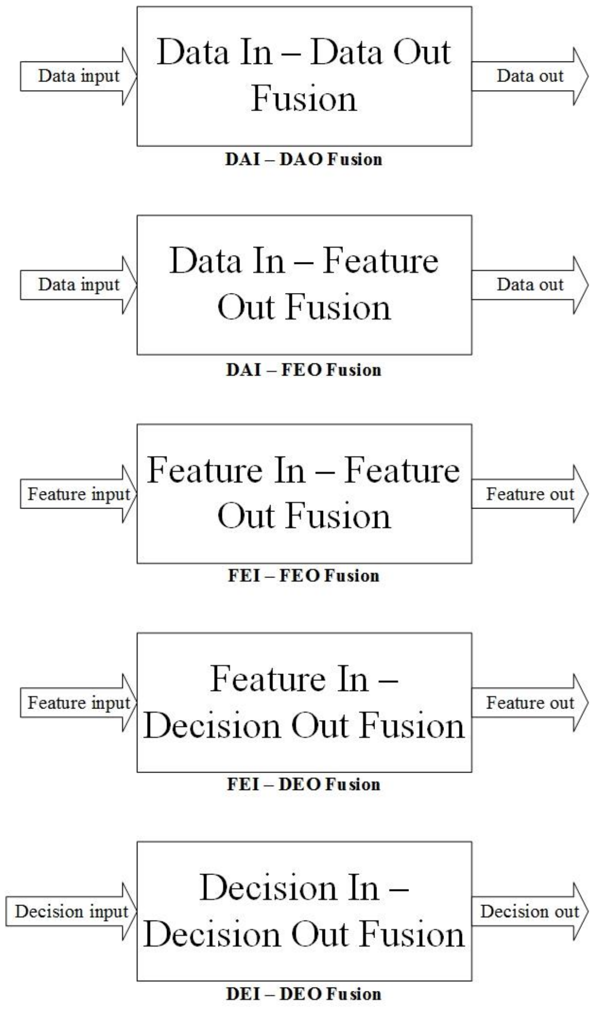


Figure 5: Dasarathy's classification of multi-sensor fusion [28]

2.5.4 Random Sets Based Model by Goodman et al.

This method combines uncertainties with decisions. It is a generic scheme of uncertainty representation.

2.5.5 Category Theory Based Model by Kokar et al.

A general theory to capture all kinds of fusions- data, feature, decision and relational information. It accounts for both data and processing. It allows consistent combination of processing elements (algorithms), while offering measurable and provable performance.

Several other fusion models exist such as the Omnibus model [29], Boyd or OODA model [30], LAAS Architecture [31].

2.6 Sensor Fusion Topologies

There are different topologies namely, Centralized, Decentralized and Hybrid [21] [24] [32] [25]. Each of these is described as follows:

2.6.1 Centralized

In this architecture, a single node handles the fusion process. The sensors undergo preprocessing before they are sent to the central node for the fusion process to take place. Figure 6 shows a typical centralized architecture [21] [24].

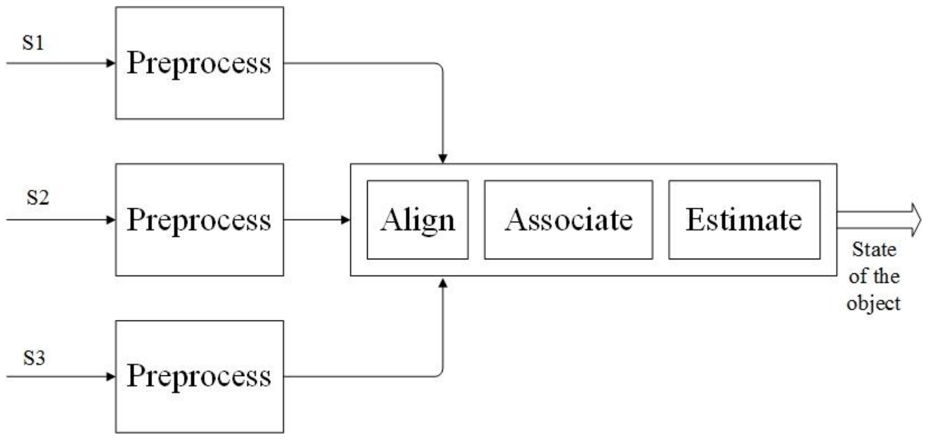


Figure 6: Centralized fusion topology [32]

2.6.2 Decentralized

In this architecture, each of the sensor processes data at its node and there is no need for a global or central node. Since the information is processed individually at the node, it is used in applications that are large and widespread such as huge automated plants, spacecraft health monitoring etc [24]. Figure 7 shows a typical decentralized architecture [21] [24].

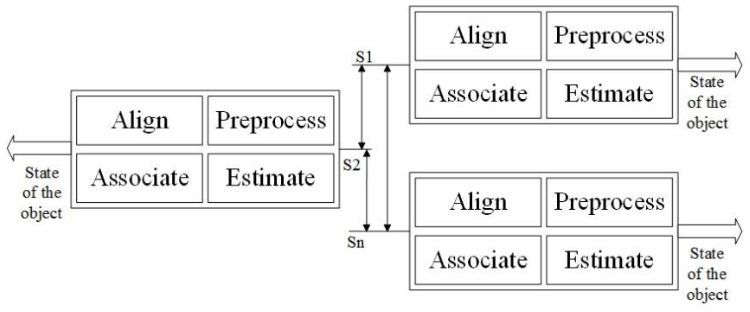


Figure 7: Decentralized fusion topology [32]

2.6.3 Hybrid

This architecture is a combination of both centralized and distributed type. When there are constraints on the system such as a requirement of less computational workload or limitations on the communication bandwidth, distributed scheme can be enabled. Centralized fusion can be used when higher accuracy is necessary [24] [32].

A simple comparison between the centralized and decentralized topologies is shown below in Table 3.

Table 3: Comparison between centralized and decentralized topologies [21] [24]

Parameter	Centralized	Distributed
Communication	Central node acts bottleneck	Data processing load distributed
Computation	Depends on the performance of central processor	Can be easily scaled
Modularity	Re-programming for new sensors	Modular in design
Fault-tolerance	Depends on the central computer	Distributed data processing

2.7 Categories of Fusion Algorithms

Sensor fusion can be performed at various levels based on the condition and type of data. In this context, there are following fusion stages:

1. Signal level fusion
2. Feature level fusion
3. Decision level fusion

2.7.1 Signal Level Fusion

In signal level fusion, data from multiple sources (sensors) are combined to obtain better quality data and higher understanding of the environment being observed. Signal level fusion often has either or both of the following goals:

1. Obtain a higher quality version of the input signals i.e. higher signal to noise ratio [33]. Sensor measurements from several sensors which have same physical properties are combined to determine the parameter being measured, more

accurately [21]. This minimizes and sometimes eliminates any uncertainty or inaccurate predictions caused by measurements from faulty sensors, measurement noise and state noise. For instance, readings from multiple temperature sensors in close proximity in a given space can be used for this kind of fusion.

2. Obtain a feature or mid-level information about the system that a single measuring node cannot reveal. A feature is the first stage in understanding the state of the environment that helps the system in formulating a decision. Heterogeneous sensors are often employed for this process. For instance, signals from radar and images from camera are used in target recognition [25].

For sensor data to undergo signal level fusion, it is essential to condition the signals in the signal preprocessing phase. The signals must be in a common representation format [21]. The stages involved in this process, as shown in Figure 8, include but not limited to: Signal alignment, normalization and scaling [21].

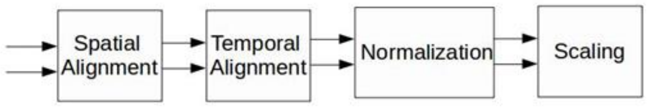


Figure 8: Signal preprocessing functions [21]

There are several methods by which signal level fusion can be achieved. The choice of method depends on various factors like the scenario and type of application, type of data or signal, relationship between the data or the state representation of the system. The following are some of the commonly used signal fusion methodologies:

2.7.1.1 Weighted Averaging

Signal fusion can be achieved by taking an average of the various sensor signals measuring a particular parameter of the environment. If signals from some sensors can be trusted more than the other, a higher weight is assigned to that sensor to increase its contribution towards the fused signal. The confidence level is a function of variance of the sensor signal [33].

$$x_{fused} = \sum_{i=0}^{n} w_i x_i \tag{1}$$

where, w_i = f(variance)

2.7.1.2 Kalman Filter

The Kalman filter method is a common adaptive method of sensor fusion to remove redundancy in the system and to predict the state of the system. This is a linear model and the current state of the system is dependent on the previous state. The system is represented by the following state-space model:

$$x(k) = A\,x(k-1) + B\,u + G\,w \tag{2}$$

$$z(k) = C\,x(k) + v \tag{3}$$

where, x: state vector, A: state transition matrix, B: Input transition matrix, u: Input vector, G: Process noise transition matrix, w: process noise vector, C: Measurement matrix, v: measurement noise vector. The covariance matrices of w and v are Q(k) and R(k) respectively. There are two phases of state estimation with Kalman filter:

Predict phase:

$$\hat{x}_k = A\,x_{k-1} + B\,u_k \tag{4}$$

$$P_k = A\,P_{k-1}\,A^T \tag{5}$$

Update phase:

$$K_k = P_k C^T (C P_k C^T + R)^{-1} \tag{6}$$

$$\hat{x}_k = \hat{x}_k + K_k(z_k - C\,\hat{x}_k) \tag{7}$$

$$P_k = (1 - K_k C) P_k \tag{8}$$

where, P: estimation covariance, K: Kalman gain.

In the update or correction phase, the estimate from the predict phase is updated with the observation. If there are two sensors and both sending data simultaneously, then Z = [z1, z2]. If the sensors are sending data one after the other, then the reading from first sensor can be used as a priori information before observation from second sensor is used to update the prediction [33].

2.7.1.3 Track-to-Track Fusion

Track to track fusion methodology has local tracks generated by distinct local sensors. Then at a central node the tracks are fused as shown in Figure 9 [34].

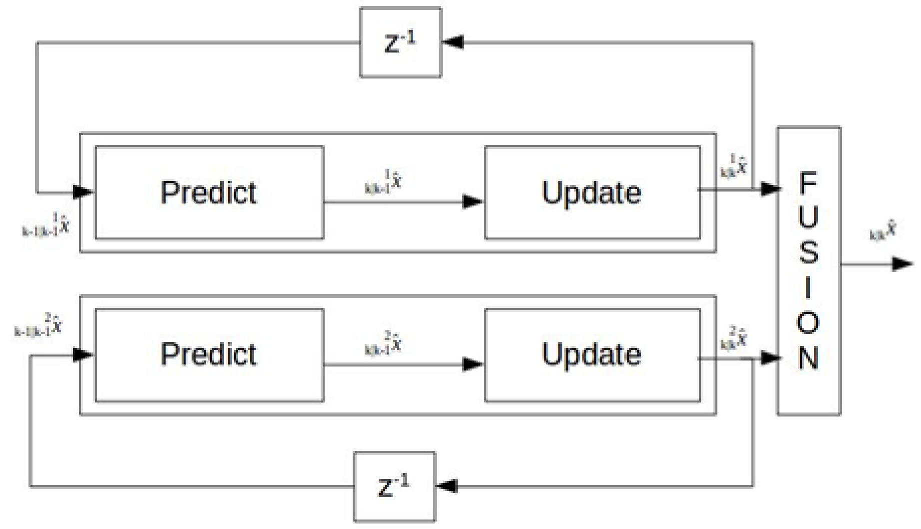


Figure 9: Track-to-track fusion architecture [34]

The local track can be individual Kalman filter nodes that provide state estimation at the local track level. These states are then fused into a state vector that has combined information from all the local sensor nodes. Sometimes, this new estimate is sent as feedback to the local sensor nodes. The new state estimate is obtained by the following formula [34]:

$$\hat{x}_{k|k} = \hat{x}^1_{k|k} + \left[P^1_{k|k} - P^{12}_{k|k}\right]\left[P^1_{k|k} + P^2_{k|k} + P^{12}_{k|k} + P^{21}_{k|k}\right]^{-1}\left(\hat{x}^2_{k|k} - \hat{x}^1_{k|k}\right) \tag{9}$$

where, $P^m_{k|k}$ is the error covariance matrix of the corresponding state estimation $\hat{x}^m_{k|k}$. $P^{12}_{k|k}$ is the cross-covariance matrix of the two state vectors where $P^{21}_{k|k} = (P^{12}_{k|k})^T$.

$P^{12}_{k|k}$ is defined by the following equation:

$$P^{12}{}_{k|k} = \left(1 - K^1_k C^1_k\right)A_{k-1}P^{12}_{k-1|k-1}A^T_{k-1}\left(1 - K^2_k C^2_k\right)^T + \left(1 - K^1_k C^1_k\right)G_{k-1}Q_{k-1}G^T_{k-1}\left(1 - K^2_k C^2_k\right)^T \tag{10}$$

This configuration can be extended for multiple sensors. A modified track-to-track fusion and three sensor fusion algorithm are explained in detail in [34].

Several authors have used track fusion algorithm for signal level fusion to minimize the effects of sensor noise. However, this work explores the use of track fusion algorithm for extracting feature while using a basic Kalman filter for signal level fusion. There are other ways to define the track fusion algorithm such as taking confidence weighted averaging of the tracks based on variance [34].

2.7.1.4 Neural Networks

An artificial neural network consists of interconnection of processing nodes called neurons. There is a pattern of interconnection between the neuronal layers that are weighted and the learning process that updates these weights. Data fusion models can be established using neural networks such that neurons and interconnecting weights are assigned based on the relationship between the multi-sensor data input and the signal output. The neural networks can be multilayer feed-forward or recurrent type [35].

Unlike Kalman filters, neural networks offer non-linear transfer functions and parallel processing capabilities. This can help in performing image fusion. Figure 10 shows a basic structure of three-layer neural network with nonlinear mapping.

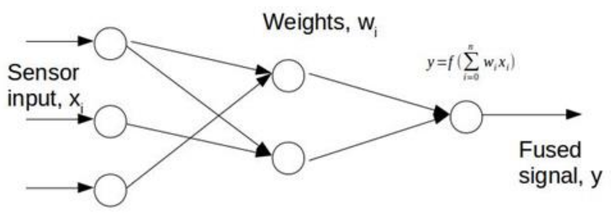


Figure 10: Neural network structure for sensor fusion [35]

The fused output is a combination of input signal and corresponding weights calculated by the equation [35]:

$$y = \sum_{i=0}^{n} w_i x_i \tag{11}$$

where, w_i is the weight; x_i is the sensor data.

Several fusion methodologies are used and depending on the input and outputs required, the stages in the model can perform either signal, feature or decision level fusion. These methods are either used as standalone or can be combined with other signal fusion methods.

The probabilistic approach for sensor fusion includes the use of joint probability distributions and Gaussian distributions [36]. Other fusion methods include Bayesian, least-squares for feature extraction [37] and some statistical approaches [21] [33] [38]. In [39] [40] [41], the authors explain various approaches for modeling sensor fusion architecture using neural networks.

2.7.2 Decision level fusion

Also known as Symbol level fusion, the decision level fusion combines several sub-decisions or features to yield a final or higher decision that can be used to take an action. Symbol could be an input decision. In this case, fusion of symbolic information insists the use of reasoning and inference while handling uncertainty. Symbol level fusion increases the confidence or truth value and is considered as decision fusion [42] [43]. Identity and Knowledge based methods form the two categories of decision fusion [24]

[43]. Table 4 [24] [43] lists few of the decision fusion methods or AI techniques for each category.

Table 4: Decision fusion models [24] [43]

Identity based	Knowledge based
Maximum a priori (MAP)	Syntax rule
Maximum Likelihood (ML)	Neural Network (NW)
Dempster-Shafer etc	Fuzzy Logic etc

One of the most widely used decision or inference method is Dempster-Shafer theory (D-S theory). This method is very useful for human-robot interaction based applications [42] [43] [44] [45]. The D-S theory is described in detail followed by a comparison with Bayesian inference which is another widely used decision fusion technique.

2.7.2.1 Dempster-Shafer Theory of Evidence:

D-S theory is a generalization of the probability theory [42] [46] [47] [44]. In this method, a frame of discernment Ω is defined which is set of elementary hypotheses:

$$\Omega = \{a_i\}, \text{where i} = 1, \ldots, \text{n} \tag{12}$$

The sum of the mass function of all hypotheses is one. Belief function is used to express inaccurate beliefs. Mass values are assigned to the elements of the power set 2^{Ω} of the frame of discernment which hold the following properties:

belief (null) = 0

belief (hypothesis) = Sum of all mass functions for all evidence to support the proposition.

The confidence interval is upper-bounded by the plausibility value to include all observations that don't rule out the proposition supported by the corresponding belief function. In order to combine two mass functions m1 and m2 the Dempster-Shafer theory defines the following rule [46] [47]:

$$m1 \oplus m2(\emptyset) \ = \ 0 \tag{13}$$

$$m_1 \oplus \ m_2(H) = \frac{\sum_{X \cap Y = H} m_1(X) m_2(Y)}{1 - \ \sum_{X \cap Y = \ \emptyset} m_1(X) m_2(Y)} \tag{14}$$

2.7.2.2 Comparison of D-S and Bayesian Fusion:

Although both these methods are widely used in inference engines there are few differences between them [43] [45]. The main difference being the concept of support and plausibility to define uncertainty limits in Dempster-Shafer [43] [46] [47] which is not found in Bayesian inference. D-S theory is an evidential reasoning method where belief masses can be assigned to elements and sets, and on set of sets [43].

Capturing ignorance or uncertainty is another strong feature of evidential reasoning methods which is not achievable in probabilistic methods. It is not necessary to have a priori probabilities and data is provided only at the time when sensor reads them [43] [45] during observation. Dempster-Shafer theory of evidence finds widespread use in human-robot interactive (HRI) applications. A review of a few applications of HRI can be found in [48].

By using the power set as the frame of discernment beliefs can well represented. However, when the set is continuous the number of subsets cannot be measured and hence this is a significant limitation that is found in evidential reasoning methods [42] [43] that work well with discrete sets.

CHAPTER 3: SELECTIVE FUSION FRAMEWORK DESIGN

This chapter provides an overview of the selective fusion framework. The framework is designed for the robot to navigate an unknown terrain. The terrain map, features, and set of obstacles are not a priori information to the robot. The framework uses the concept of sensor fusion. It selectively utilizes required sensors associated with the task to determine the decisions required to execute the task successfully.

The focus of this work is the decision arrived by the framework to aid in navigation and not the navigation algorithm itself. Any path planning or navigation algorithm can be used, as the framework does not intervene with the navigation.

Several fusion frameworks found in the literature focus on using one of the signal level fusion algorithms mentioned in the Chapter 2 or its variants to filter out noise in the sensor signal. Other decision making frameworks utilize mostly soft data set and apply decision level fusion algorithms to arrive at a decision. This work presents a fusion framework that uses a combination of these algorithms at relevant stages to achieve higher level of cognition and increase confidence in autonomous decision making.

A robot is normally designed to execute a set of tasks. It also has sensors to gather the state of environment it is operating in. According to the framework as shown in Figure 11, a set of sensors are mapped to a task [49] [50]. Initially, sensors are selected and the raw sensor data undergoes a phase of preprocessing. This data then goes through the fusion process and a decision is arrived to execute a task or a sub task.

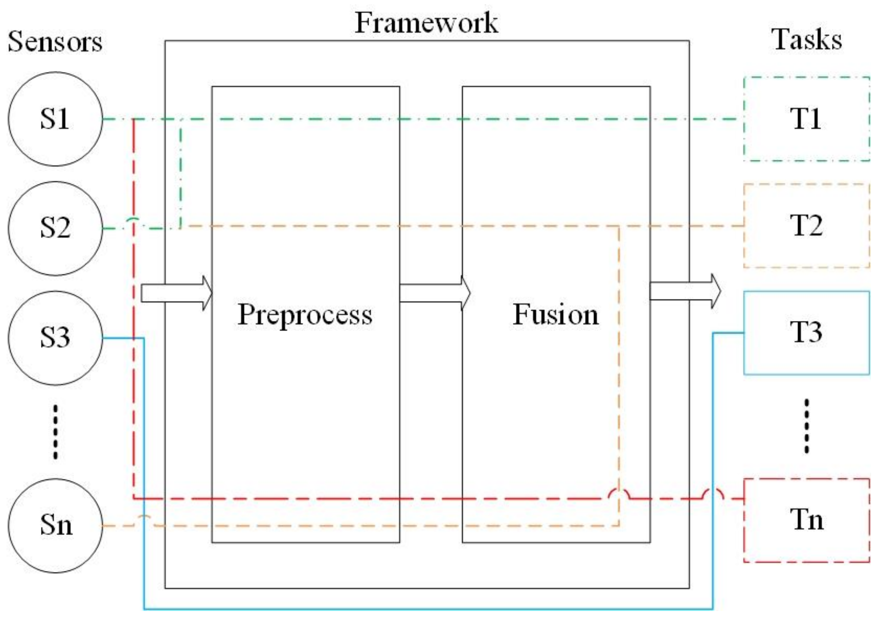


Figure 11: Selective sensor fusion framework

For a robot to navigate an unknown terrain, it is essential to analyze the terrain features and detect surface obstacles. LIDAR sensors are used for this purpose. Consider a robot with three sensors mounted at a 45 degree angle to the surface to get the terrain data and three more sensors mounted horizontal to the surface to get surface data as shown in Figure 19. Several such sensor configurations can be used. One sensor can be used for measuring surface obstacle and another one can be used to measuring terrain features. However, in this scenario, several sensors are used to demonstrate selection and scalability of the framework. The task to be taken up by the robot is movement. The robot used for the demonstration in this work is non-holonomic. So, movements are restricted to forward and turns. The movements are for fixed distances from one cell to another. The cell size can vary based on requirement, however, in this case, the terrain is divided into square cells of 1m length. So, the sensors and tasks are established, the actual fusion framework is explained in detail as follows. The block diagram of the framework is as shown in Figure 12.

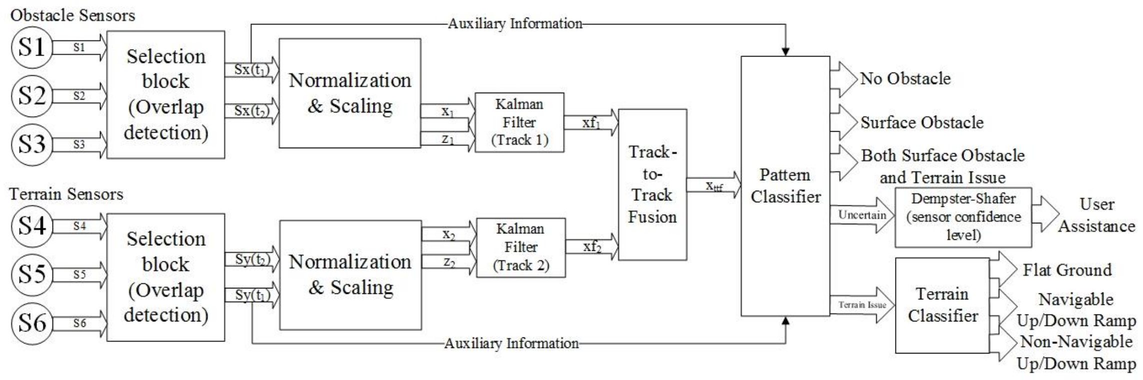


Figure 12: Fusion framework block diagram

The raw sensor data undergoes selection through the selection block. The selected sensor is then sampled again to get the next temporal sequence of data. This data is then sent to the pre-processing block for normalization and scaling. The normal data is sent through the Kalman filter for temporal fusion to minimize the effects of noise. This fused data from individual Kalman nodes is then sent to the track-to-track fusion node. The signal out of this node contains information of the state of the next cell under consideration. The pattern classifier block identifies the features of the environment and aids the decision level block to arrive at a decision for the robot to act on. The various parts of the framework are explained in detail below.

3.1 Overlap Detection

This block is essential to bring about spatial alignment of data. It also demonstrates sensor selection in the fusion process. The input to this block is the raw sensor data from six LIDAR sensors. Each sensor generates a stream of spatial and temporal sequence of data. Three of the horizontal LIDAR sensors are mounted such that there is a degree of overlap between the measurements i.e., a part of each LIDAR's data is a measure of the same point in space. The spatial overlap of data generates some amount of redundancy that is helpful to overcome some of the challenges mentioned in Section 2.3. Figure 13 shows the overlap region for horizontal sensors. This block uses the data from the region of overlap to increase confidence in the sensor's measurement. The sensor data with highest degree of confidence will be selected for the fusion process. Figure 15 shows the region of detection for the terrain sensors. The data set with the maximum deviation from the ideal flat ground LIDAR data sequence is considered for the fusion process. The next sequence of data from these sensors are also read.

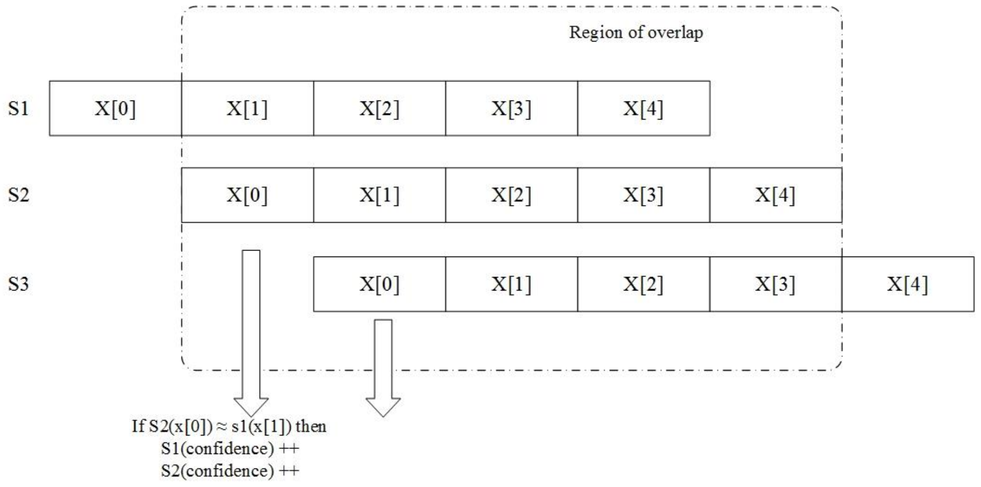


Figure 13: Region of spatial overlap for obstacle sensor

3.2 Preprocessor

The preprocessor brings the two data sets obtained from the overlap detection block to a common representation format. Every sensor measurement consists of the following parameters: spatial position, time, value, uncertainty of the given property as measured by the sensor. To achieve a common representation format, the sensor readings must be temporally and spatially aligned. This is taken care by the overlap detection block. The other requirement is that the values are normalized that is they are converted to a common scale. Normalization is achieved by a technique called the Min-Max method of normalization as defined below [21].

$$y = \frac{(x - a)}{(b - a)} \tag{15}$$

where, a = $\min_i(x_i)$, b = $\max_i(x_i)$.

Both the obstacle sensor and the terrain sensor are looking at the next cell in the path of the robot. The goal of the fusion process is to determine whether the cell is navigable or not. However, both these sensors are measuring different and independent attributes of the cell. Hence, there is heterogeneity in data. The obstacle sensor looking for a surface obstacle in the next forward cell is as shown in Figure 14 and the terrain sensor measuring the terrain irregularity in the next cell is as shown in Figure 15.

The boundary conditions for both the cases are used to determine the min and max values of the sensor data for normalization. The LIDAR sensors can measure up to a certain maximum distance based on its characteristics that is detailed in the data sheet of the sensor. The sensor used for this application can measure up to a maximum of 10m. This measure is taken as the max value for normalization for the obstacle sensor. The min value depends on the boundary of cell under consideration. As shown in Figure 14, the min value for a particular LIDAR range value is the boundary value of the cell at that angle.

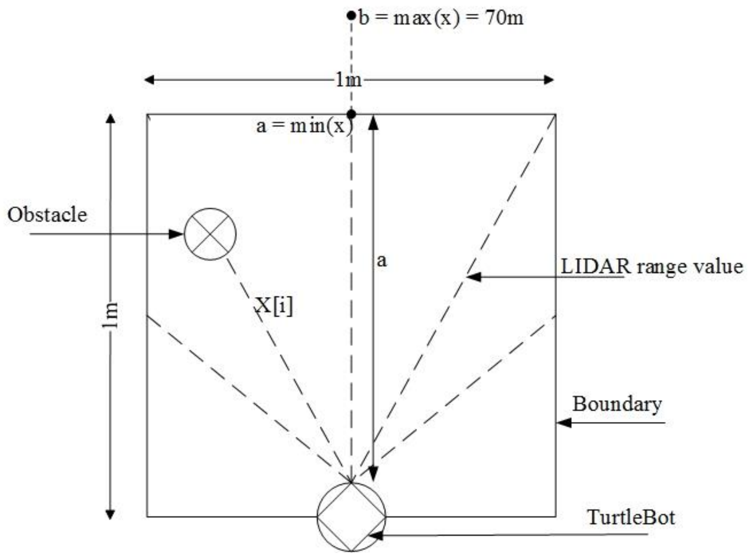


Figure 14: Obstacle sensor

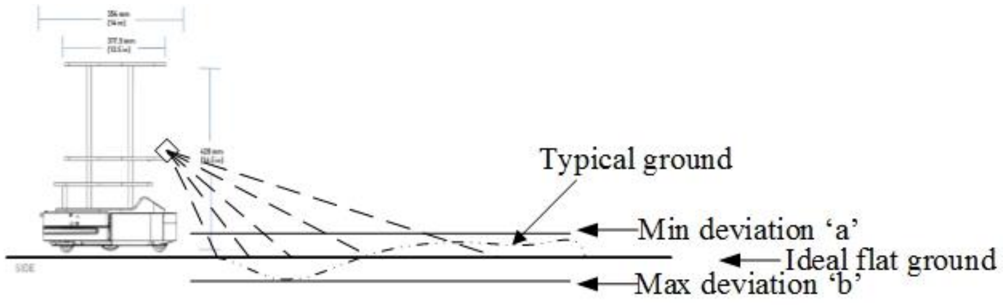


Figure 15: Terrain sensor

Figure 15 shows the LIDAR range values for the ideal flat ground. A deviation from the ideal flat ground value is used to determine the min and max values for the range values of terrain sensor. The obstacle sensor values have a higher difference between their min and max values compared to the terrain sensor. So, the obstacle sensor values are scaled after normalization.

3.3 Kalman Filter

The input to this block is the normalized data from the preprocessing stage. A temporal sequence of normalized data from the obstacle sensor and that from the terrain sensor are processed by this block. Detailed explanation of the working of Kalman filter and its use in sensor fusion is in Section 2.7.1.2.

This section focuses on implementation of Kalman filter for the fusion process. Two sets of temporal sequences of data obtained from the obstacle sensor are homogeneous in nature. Each set measures the same parameter of the environment and are from the same sensor. These are combined using Kalman fusion to obtain an estimate of the state that is free of sensor noise. This process is repeated for the data from terrain sensor too.

To implement the Kalman filter for fusion process, some of the parameters must be initialized. The state vector, x at time instant k-1 is the first of the temporal sequence of normalized LIDAR data. The size of the state vector depends on the resolution of sensor data. If ‘M’ range values are obtained in a LIDAR sweep, size of matrix x is [MxN] where, N = 1.

The state transition matrix A, is initialized to an identity matrix of size [MxM] because of the lack of knowledge on how the system varies. The measurement matrix, C is also an identity matrix of size [KxM] which indicates the sensor measurement contributing to observed value. The estimation covariance matrix P[MxM] is also initialized to an identity matrix.

The Kalman gain matrix G[MxK] is initialized to 0.5*I[MxK] which gets updated on every run. The covariance of process noise Q[MxM] and the covariance of measurement noise R[KxK] are initialized based on the expected levels of noise and are different for obstacle sensor and terrain sensor. Both these matrices are normally initialized to standard deviation of noise ie., σ*I[]. Since raw sensor values are not used for the fusion process, Q[MxM] for obstacle sensor is obtained by the following formulae:

$$Q[M{\times}M] = q1 * I[M{\times}M] \tag{16}$$

$$q1 = \frac{(\sigma - a_{avg})}{(b_{avg} - a_{avg})} \tag{17}$$

where, $a_{avg} = \sum min_i(x_i)/M,\ b_{avg} = \sum max_i(x_i)/M,$ σ is the standard deviation of process noise.

R[KxK] for obstacle sensor is obtained by the following formulae:

$$R[K{\times}K] = r1 * I[K{\times}K] \tag{18}$$

$$r1 = \frac{(\sigma - a_{avg})}{(b_{avg} - a_{avg})} \tag{19}$$

where, $a_{avg} = \sum min_i(x_i)/K,\ b_{avg} = \sum max_i(x_i)/K,$ σ is the standard deviation of sensor noise.

Q[MxM] for terrain sensor is obtained by the following formula:

$$\mathrm{Q[M{\times}M] = abs(q2) * \frac{I[M{\times}M]}{k}} \tag{20}$$

$$\mathrm{q2 = \frac{(\sigma - a_{avg})}{(b_{avg} - a_{avg})}} \tag{21}$$

where, $a_{avg} = \sum min_i(x_i)/M$, $b_{avg} = \sum max_i(x_i)/M$, σ is the standard deviation of process noise, k is the scaling factor.

R[KxK] for obstacle sensor is obtained by the following formulae:

$$\mathrm{R}[\mathrm{K}\times\mathrm{K}] = \mathrm{abs}(\mathrm{r2}) * \frac{\mathrm{I}[\mathrm{K}\times\mathrm{K}]}{\mathrm{k}} \tag{22}$$

$$\mathrm{r2} = \frac{(\sigma - \mathrm{a_{avg}})}{(\mathrm{b_{avg}} - \mathrm{a_{avg}})} \tag{23}$$

where, $a_{avg} = \sum min_i(x_i)/K$, $b_{avg} = \sum max_i(x_i)/K$, σ is the standard deviation of sensor noise, k is the scaling factor.

The values are scaled down here by a factor of ‘k’ because the difference between minimum average and maximum average value of obstacle sensor is higher than that of terrain sensor. So, if the values are not scaled, the terrain sensor will have higher impact on the results. The scaling factor considered in this study is 1000.

The observed state vector, z of the size [MxN] is the second in temporal sequence of normalized LIDAR data.

With the above mentioned initializations and boundary conditions in place, the Kalman Fusion runs separately for the obstacle sensor and the terrain sensor and obtains two state vectors in the update phase. The Kalman Fusion block takes signal in and provides signal out. This is in-line with the Dasarathy’s data flow model of fusion.

3.4 Track-to-Track Fusion

The working principle of track-to-track fusion process is explained in Section 2.7.1.3. The input to this block are the state vectors of obstacle sensor and terrain sensor from the Kalman fusion block. This is an extension to the Kalman fusion block and several parameters of the fusion equation are required in the track-to-track fusion block. Once the cross-covariance matrices of the two state vectors are obtained using Equation 10, the fused state estimate is obtained from Equation 9 that contains information about both obstacles and terrain features of the cell in front of the robot.

With the above-mentioned initializations and boundary conditions in place, the Kalman and track-to-track fusion are tested for a continuous 36 runs with random sensor values and different boundary conditions. The results of the test are tabulated in Appendix E. The first column contains the raw sensor value from obstacle sensor and the second column contains the next sequence of measurement from the same sensor and the same cell. The third column contains the measurements from the terrain sensor and the fourth column contains next sequence of data from the same sensor and for the same cell. All these data go through track-to-track fusion. The process is repeated with the same data for four different scaling factors in the preprocessing block i.e., n = 1, 10, 100, 1000 and the resultant vectors are stored in column five through eight. The scaling factor is essential to prevent bias of single sensor on the result. The track fusion block provides a signal out that needs to be analyzed to determine the state of the cell. The resultant vector of the fusion process holds information as to whether the cell is navigable or not. It can be observed from the eighth column in the table in Appendix E that all the elements of the resultant vector lie within the range 0 and 1 if there are no obstacles and terrain irregularities. Cell Є {navigable}, if $0<xh[i]<1$ for all xh[i]. Cell Є {non-navigable}, if $0>xh[i]$ or $xh[i]>1$ for any xh[i]. This is marked in the table in Appendix E with a red shade. The other shaded blocks on the sensor reading from column one through four indicate the presence of an obstacle or terrain irregularity. They are marked for convenience to correlate the navigability of a cell with the track-to-track fusion result values. Hence the fused state value classification is achieved.

The boundary conditions for the test run are as follows:

$\text{Obstacle}_{\text{min}}$ = {0.45, 0.55, 0.7, 0.85, 0.95}

$\text{Terrain}_{\text{min}}$ = {0.5, 0.6, 0.75, 0.9, 1}

$\text{Terrain}_{\text{max}}$ = {0.55, 0.65, 0.8, 0.95, 1.05}

The track-to-track fusion block is sensitive to minor irregularities in terrain. The boundary conditions assigned as min and max values determine the limit. So, based the track-to-track fusion results, the terrain parameter is further classified in the terrain classifier block.

3.5 Pattern Classifier

This is the feature classification phase of the fusion process. The information about navigability of the cell is obtained from the signal out of the track-to-track fusion as described above. The framework must accumulate confidence through various stages to achieve higher levels of cognition. The pattern classifier is a rule based system that takes the results of track fusion and auxiliary information from the previous stages to determine features of the cell under consideration. Table 5 contains the details of these features. In the truth table, ‘1’ indicates that the cell is non-navigable as determined by that source (named in the column heading) while ‘0’ indicates navigability. Any uncertainty in the determination is also accounted for in this stage.

Table 5: Pattern look-up table

Auxiliary information(Obstacle)	Auxiliary information(Terrain)	Track-to-track fusion result	Pattern
0	0	0	No obstacle
0	0	1	Uncertain
0	1	0	Uncertain
0	1	1	Terrain issue
1	0	0	Uncertain
1	0	1	Surface obstacle
1	1	0	Uncertain
1	1	1	Both obstacle and terrain issue

3.6 Terrain Classifier

This block further classifies the terrain based on its features. If the pattern classifier finds that the cell is navigable or that there is a surface obstacle, then this block is not consulted in arriving at a decision. If the pattern classifier suggests that there are both surface obstacle and terrain irregularity or there is only terrain irregularity, then this block is used to further classify the terrain features to make appropriate navigation decisions.

The data input to this block is the sensor reading from terrain sensor after selection and the control input is from the pattern classifier block as shown in Figure 16. This block takes into account the slopes of the terrain in various parts of the cell to determine if the cell is: overall flat, a navigable up ramp, a navigable down ramp, a non-navigable up ramp, a non-navigable down ramp.

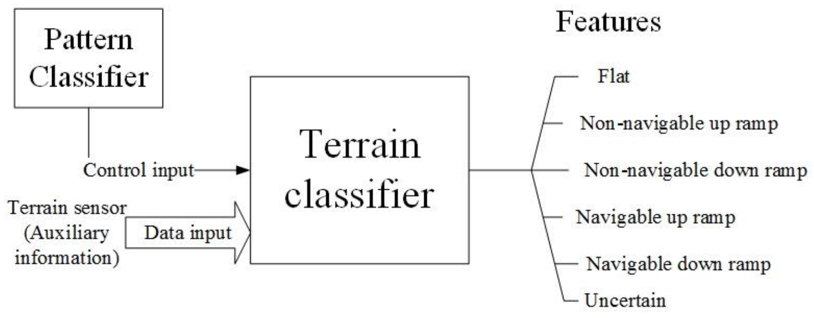


Figure 16: Terrain classifier block diagram

The terrain LIDAR sensor gives out [angle, range] values as marked by [θ,d] in Figure 17. The coordinates of points on terrain are obtained by the following formula:

$$x_i = d_i * \sin\left(\frac{(15 * i) * \pi}{180}\right) \tag{24}$$

$$y_i = y_h - \left(d_i * \cos\left(\frac{(15 * i) * \pi}{180}\right)\right) \tag{25}$$

where, (x_i, y_i) are the coordinates on the terrain, y_h is the distance of terrain sensor from the ground (mounting height), d_i is the range data. In this case, $\theta_i = 15*i$.

The slope between two points on the terrain are determined to measure the extent of irregularity in the terrain. It is obtained by the following formula.

$$m = \frac{y_2 - y_1}{x_2 - x_1} \tag{26}$$

The corresponding angle of ramps are obtained by the following formula:

$$\alpha^\circ = \left(\frac{(\tan^{-1} m) * \pi}{180}\right) \tag{27}$$

where, α is the angle of terrain surface. These angles are used to rank the cell surface for the features listed above.

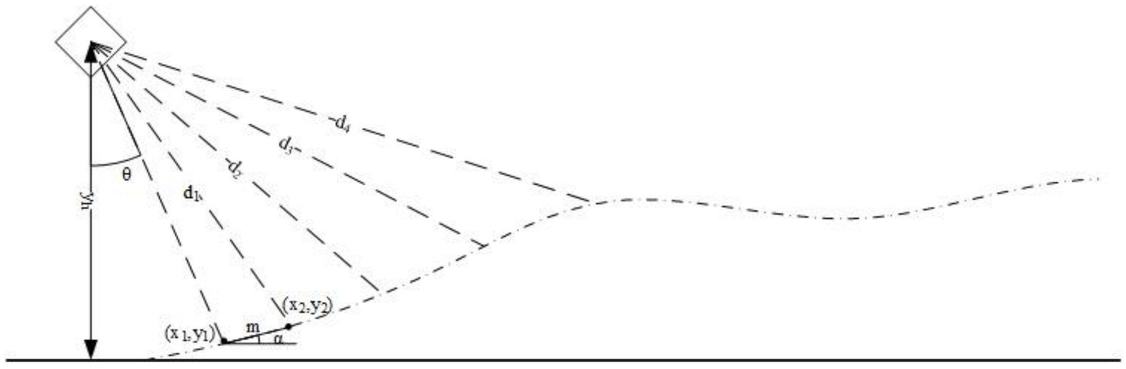


Figure 17: Terrain angle

3.7 Decision Block

The decision block arrives at a decision based on the evidence(features) determined by the pattern classifier and the terrain classifier. This block takes the features from the previous block as input. The mapping of feature set to the decision set is as shown in Table 6. The robot is programmed to act based on the result of the decision block. The decision to action mapping is tabulated in Table 7.

Table 6: Feature to decision mapping

Feature	Determination block	Decision
No obstacle	Pattern classifier	Navigable cell
Surface obstacle	Pattern classifier	Non-navigable cell
Terrain issue	Pattern classifier	Inspect the terrain further
Both surface and terrain issue	Pattern classifier	Non-navigable cell
Uncertain	Pattern classifier	Uncertain mode
Flat	Terrain classifier	Higher navigability index
Navigable up ramp	Terrain classifier	Higher navigability index
Non-navigable up ramp	Terrain classifier	Non-navigable cell
Navigable down ramp	Terrain classifier	Higher navigability index
Non-navigable down ramp	Terrain classifier	Non-navigable cell

Table 7: Decision set to action set mapping

Decision	Action
Navigable cell	Move forward by a cell
Non-navigable cell & previous action: move forward	Turn left by 90 degree
Non-navigable cell & previous action: turn left	Turn left by 180 degree; reset previous action

These sets can be expanded to account for more functionality in the robot. If a decision is dependent on a combination of several features, then a decision level fusion algorithm with higher level of cognition can be used. In this study, the parameters of action are held constant because the robot is traversing 1mx1m square cell. In a more complex scenario, exact distance of movement or turning angle can be a part of the action tuple. For instance, the turn angle of motors for a robotic arm.

Once the robot takes action based on the decision, the framework continues to re-sample the sensors and go through the fusion process to arrive at a new decision for the robot to execute.

The framework performs autonomously for all the features except for the uncertain feature. The robot enters an uncertain mode when the framework has not been able to build sufficient confidence on the state of the environment through its process. Any ambiguity in the findings of the framework about the environment that it is not able to resolve, results in this mode. Handling such uncertainty is an important aspect of intelligent autonomous systems. In the uncertain mode, the robot enters the assist phase where a user command is requested. The framework also calculates a confidence level in the sensors using the Dempster-Shafer rule of combination as stated in Equation 13 and Equation 14. The mass values are specified in Table 8 [51].

Table 8: Mass values for D-S theory

	0<x[i]<0.06	0.06<x[i]<max	x[i]>max
T(present)	0.3	0.8	0.2
T(not present)	0.3	0.2	0.7
T(uncertain)	0.4	0	0.1

The D-S theory accounts for uncertainty unlike the Bayesian theory or other probabilistic methods. The D-S theory makes the system scalable i.e., if there are many more hypotheses, they can be included to determine confidence. The mass table here is based on the sensor characteristics. The mass values can also be dynamically updated by the result obtained from previous stages in the framework.

In the assist phase, the framework requests for user assistance in the form of a keyboard input. The action and corresponding commands are tabulated in Table 9.

Table 9: User command and action set

User command	Action
f	Move forward
r	Turn right
l	Turn left

One of the following cases occur during execution of the framework:

Case a: The user enters a correct command. In this case, the entered command is executed and the framework returns to autonomous mode once executed.

Case b: The user enters an incorrect input. In this case, the robot takes no action and the user is requested again to enter a correct command.

Case c: The user does not enter an input. The robot waits for a set timeout period and enters the autonomous mode on timeout. The sensors readings are acquired again and they go through the fusion process hoping for better confidence in this run.

There is a bidirectional communication between the robot and user. The framework offers a way for user interaction.

CHAPTER 4: IMPLEMENTATION

The details of framework design and the principles that guide the development of sensor fusion framework are explained in the previous sections. This section introduces the actual implementation of the framework. The framework is developed using C, C++. The simulation environment and the robot are created using Gazebo (v 7.1), a robot simulation tool. Robot Operating System (ROS) (v Kinetic Kame), a collection of libraries that support robot software development is also used.

4.1 Gazebo

Gazebo is a dynamic 3D simulation engine that can simulate robots accurately in highly complex indoor and outdoor environments. It has a robust physics engine that offers physics simulation with high degree of fidelity and good quality graphic rendering. It offers interface for programs and users alike. The procedure to install Gazebo and related packages can be found in Appendix A. Figure 18 shows a basic Gazebo window. A list of components of Gazebo required to create a robot simulation:

1. World files (.world): The world description file contains all the elements in a simulation such as robots, lights, sensors and static objects. It is formatted using SDF (Simulation Description Format) and has a .world extension. The Gazebo server (gzserver) reads this file to generate and populate a world [52]. In this case, the robot along with its sensors is created as a model file and during ru

2. n time, “~/spawn_sdf_model : gazebo_msgs/SpawnModel” service is used to spawn the model in the Gazebo world.

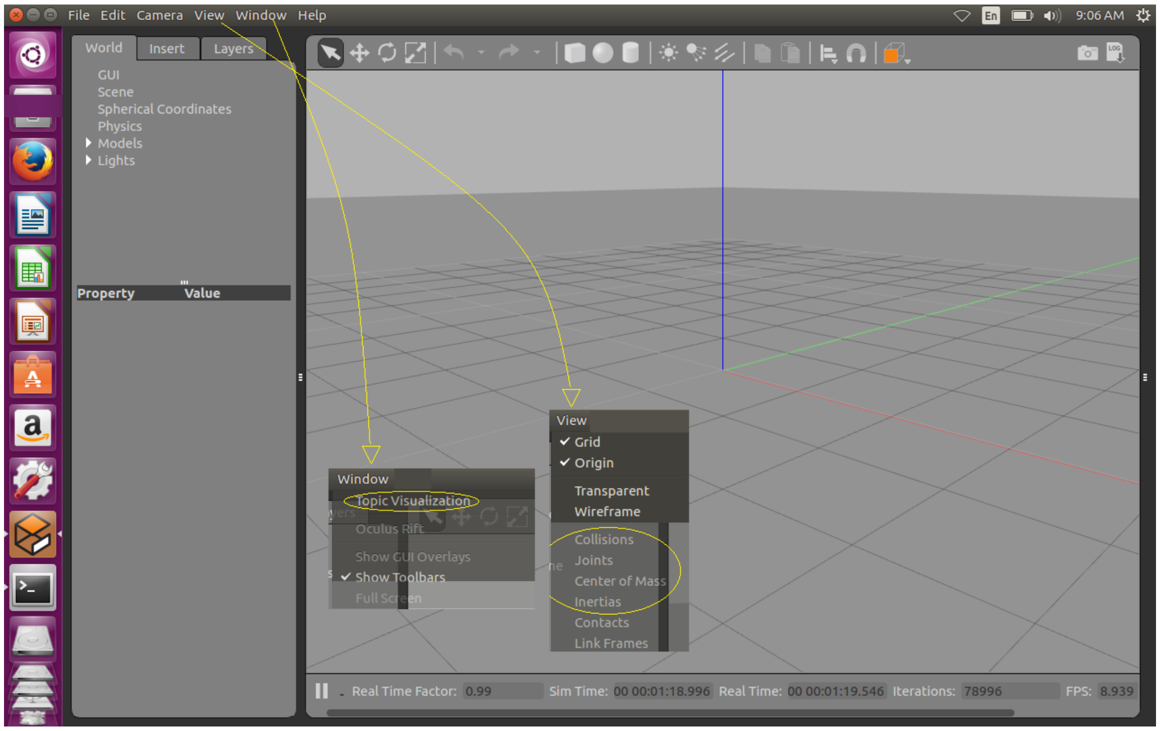


Figure 18: Gazebo window

3. Model files (.sdf): A model file also used the SDF format but contains only a single model that starts with a <model> tag and ends with a </model> tag. This facilitates model reuse and simplifies world files. Once a model file is created, it can be included in a world file using the following SDF syntax:

```
<include>
        <uri>model://model_file_name</uri>
</include>
```

4. Environment variables: Gazebo uses environment variables to locate files and set up communication between the server and clients. Default values are compiled in and they work for most cases. Some of the variables are as follows:

GAZEBO_MODEL_PATH, GAZEBO_RESOURCE_PATH, GAZEBO_MASTER_URI, GAZEBO_PLUGIN_PATH, GAZEBO_MODEL_DATABASE_URI. In this case, the model path variable is set to the workspace as explained in Appendix A.

5. Gazebo server: The server is the workhorse of Gazebo. It parses a world file given on command line and then simulates the world using a physics and sensor engine. However, the server does not include any graphics. It is started using gzserver filename.world command.
6. Graphical client: The graphical client connects to a running gzserver and visualizes the elements. It provides options to modify a running simulation. It is started using the gzclient command. The server and the client can be started using a single command gazebo filename.world
7. Plugins: Plugins provide a convenient way to interface with Gazebo. They can be specified on the command line or in the world/model file. Plugins specified on the command line are loaded before those specified in the world/model file. Some of the plugins used for this work are related to sensors and motors to drive the robot. These are specified in the model file. Other plugin to spawn the robot in Gazebo is issued on the command line. These plugins are from the gazebo_ros package. A set of ROS packages called the gazebo_ros_pkgs provide wrappers around standalone Gazebo to achieve integration with ROS. The details of this package are found in the weblink:
   http://gazebosim.org/tutorials?tut=ros_overview&cat=connect_ros

SDF is an XML format that describes objects and environments for robot simulators, visualization, and control [53]. The SDF specification can be found in this weblink: http://sdformat.org/spec

As discussed in the earlier sections, a robot model with six LIDAR sensors must be created on Gazebo to verify the framework. A robot using iRobot Create® as the base, akin to a TurtleBot v1 is created. Instead of the Microsoft Kinect sensor, this model has LIDAR sensors, three of them mounted on the top plate that look for surface obstacles and three others mounted on the mid plate at a tilt angle of 45 degrees to scan the terrain as shown in Figure 19.

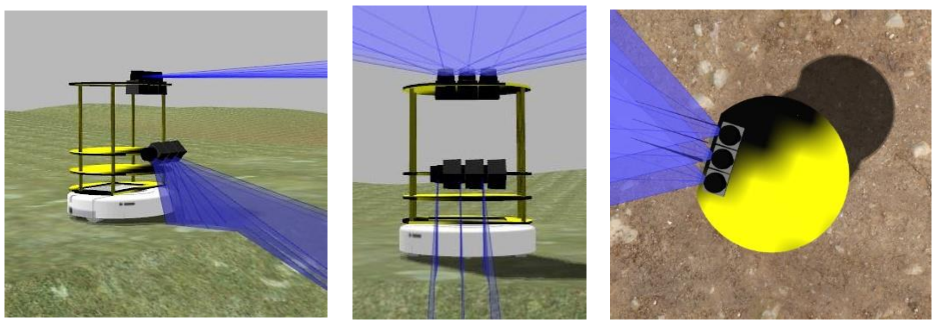


Figure 19: Robot model

4.1.1 Creation of Robot Model Using SDF

An SDF file is a collection of elements that describe the model so that the Gazebo server can generate the model. These models can range from simple static objects to complex robots. The elements are enclosed in a <model> tag and are described as follows [54]:

1. Link: It contains the physical properties of a part of the model. Normally parts with movable joints are considered as different links. Parts without movable joints are grouped under same link.

Collision: A collision element encapsulates a geometry that is used to collision checking. This can be a simple shape (which is preferred), or a triangle mesh (which consumes greater resources). A link may contain many collision elements.

Visual: This is used to visualize a part of the link. This element need not be present.

Inertial: It describes the dynamic properties of the link such as mass, rotational inertia matrix.

Sensor: This element collects data from the world.

2. Joint: It connects two links and one of the link is the parent link and the other child link. Other parameters such as joint limits and axis of rotation are mentioned in this element.
3. Plugin: A plugin is a third party shared library to control the model.

In this case, nine links are defined; one for the base itself (including chassis), one for each wheel (left and right) and one for each sensor (total of six). Each sensor link has the range and angular resolution, min and max values defined. The type of noise modeled by the sensor is Gaussian noise and the standard deviation of noise can be varied in the <noise> tag in the sensor link. The sensor link also has a plugin element that uses a library called the “libgazebo_ros_laser.so” to publish the sensor readings via ROS. A unique topic name is defined for every sensor and the reason for this is detailed in Section 4.3. A differential drive controller plugin called the “libgazebo_ros_diff_drive.so” controls the wheels with the help of topics such as cmd_vel, odom, base_footprint.

All of the above mentioned elements are put together in an model.sdf file (here, IRobotCreateHokuyo.sdf). This is placed in the ROS workspace (catkin_ws) as detailed in Appendix B and is launched using the “ROS Service Call Spawn Method”. [55]

4.1.2 Creation of Robot’s Environment Using Heightmap

Gazebo visualization has a flat surface by default, for the robot to navigate. It is essential to test the robot on an uneven terrain and check the framework’s behavior. In this case, a heightmap is used to generate a non-flat terrain in Gazebo. [56]

A heightmap is essentially an extrusion of 2D grayscale image to produce a 3D surface. Black color (hex: #000000) represents the lowest point in the plane whereas, white color (hex: #ffffff) represents the highest. The grayscale in between these two colors represents relative heights on the 3D terrain.

Figure 20 shows the heightmap used for this work and its resultant 3D terrain is in Figure 21. Gazebo requires the heightmap image to be a square image and has (2^n)+1 pixels, for instance, 129x129, 257x257 etc. Each pixel color represents a height value in the terrain. The heightmap image is a .png file without alpha channel.

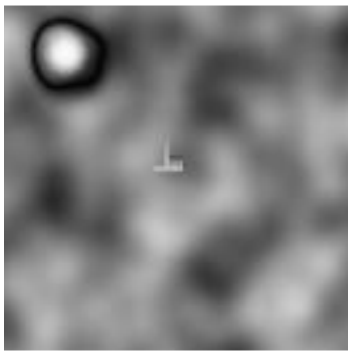


Figure 20: Heightmap grayscale image

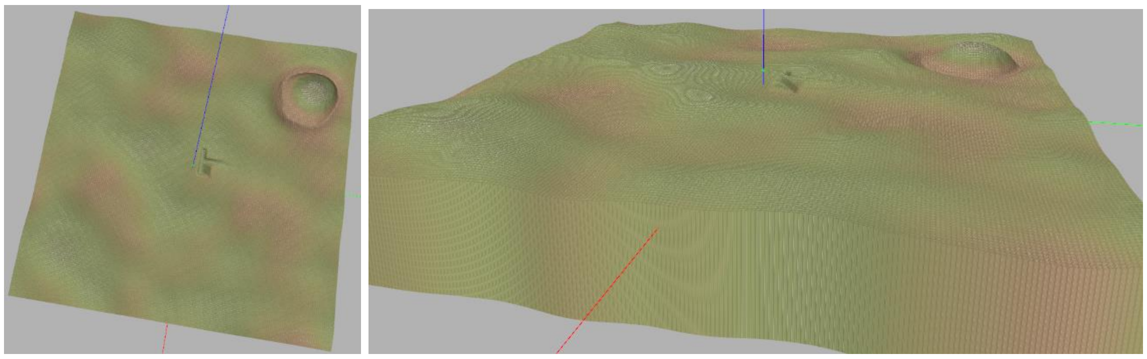


Figure 21: Resultant 3D extrusion

A .world file has to be created to load the heightmap on Gazebo. In this case, a model.world file is created that has a single link with a collision element and a visual element. Under a <heightmap> tag, the location of heightmap.png and the length, width and height of the surface is specified. In this case, it is 200mx200mx10m. This means that the highest point is 10m while the lowest being 0m. The visual element holds the various textures of the ground such as grass and dirt, and the blending indices. A lunch file is also created that opens up the model.world file using the roslaunch command as explained in Appendix B.

4.2 ROS

Robot Operating System (ROS) is a collection of libraries that support robot software development. It provides hardware abstraction, device drivers and message-passing mechanisms that enable robot application frameworks.

The procedure to install ROS is explained in Appendix A. A ROS workspace must be set up to install user programs. The procedure to set up the workspace (called the catkin) is also explained in Appendix A. The sequence of steps to install the custom fusion framework related packages are detailed in Appendix B. Every package in a catkin workspace consists of a source folder that has the source files, a CMakeLists.txt file and a package.xml file. The dependencies to build and run the package are listed in these two files. In this case, the package named “gazeboros” that consists of robot’s world, as explained in the previous section, has the following build and run time dependencies:

```
<build_depend>gazebo_msgs</build_depend>
<build_depend>gazebo_plugins</build_depend>
<build_depend>gazebo_ros</build_depend>
<build_depend>gazebo_ros_control</build_depend>
<build_depend>roscpp</build_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>
<run_depend>gazebo_msgs</run_depend>
<run_depend>gazebo_plugins</run_depend>
<run_depend>gazebo_ros</run_depend>
<run_depend>gazebo_ros_control</run_depend>
<run_depend>roscpp</run_depend>
<run_depend>rospy</run_depend>
<run_depend>std_msgs</run_depend>
```

The package named “lab2” that consists of actual fusion framework has the following dependencies: nav_msgs, sensor_msgs, laser_geometry, geometry_msgs, tf, roscpp, rospy, std_msgs, genmsg.

Since catkin should compile the framework code and create and executable, the details of the executable have to be mentioned in the CMakeLists.txt file.

The following are terminologies used to describe the working principle of ROS framework [57]:

1. Node: An executable that uses ROS to communicate with other nodes.
2. Master: Provides naming and registration services to rest of the nodes in ROS, facilitating communication. ROS master is started by the command “roscore”.
3. Topic: Nodes can publish messages to a topic or can subscribe to a topic to receive messages.
4. Message: ROS data type used for publishing and subscribing to a topic. Some of the message types used for this work are mentioned in Appendix D.

4.3 ROS Gazebo Interaction

The two main nodes, in this simulation running on Gazebo are “framework” and “gazebo”. Every sensor publishes data onto a unique topic specified in the IRobotCreate.sdf model file at a specified rate of 4Hz. The published data is in the form of ROS message type specified in Appendix D. The ROS computation graph is shown in Figure 22. It depicts the interaction of ROS and Gazebo during run time. All of the nodes run in parallel.

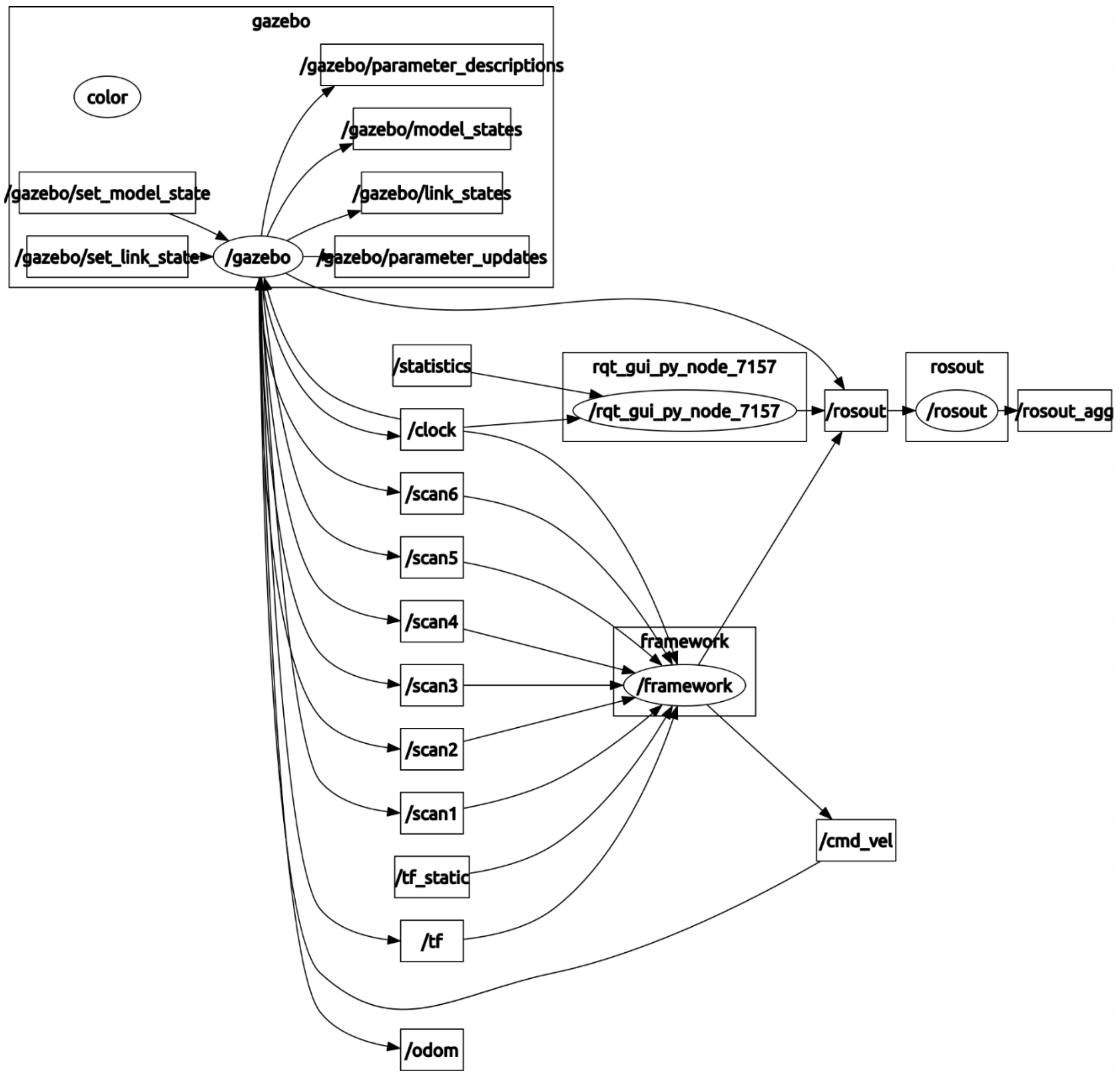


Figure 22: rqt graph

CHAPTER 5: SIMULATION

When the robot starts navigating, the state of the cells in the terrain are initially “unknown” (as shown in Figure 23) to the robot because there is no a priori information given to the robot. The framework is tested on a flat ground with surface obstacles as well as uneven ground with surface obstacles.

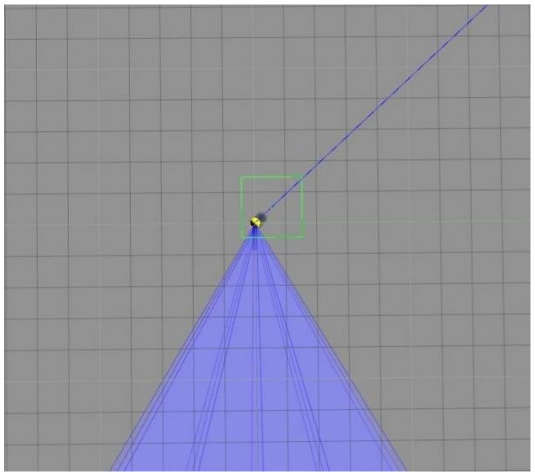


Figure 23: Terrain cells

On the flat ground, the frameworks ability to recognize presence of surface obstacle in a cell under consideration, irrespective of the state of the terrain is tested. The robot successfully navigates the flat ground with surface obstacles and the simulation at some of the key time instances are shown in Figure 24.

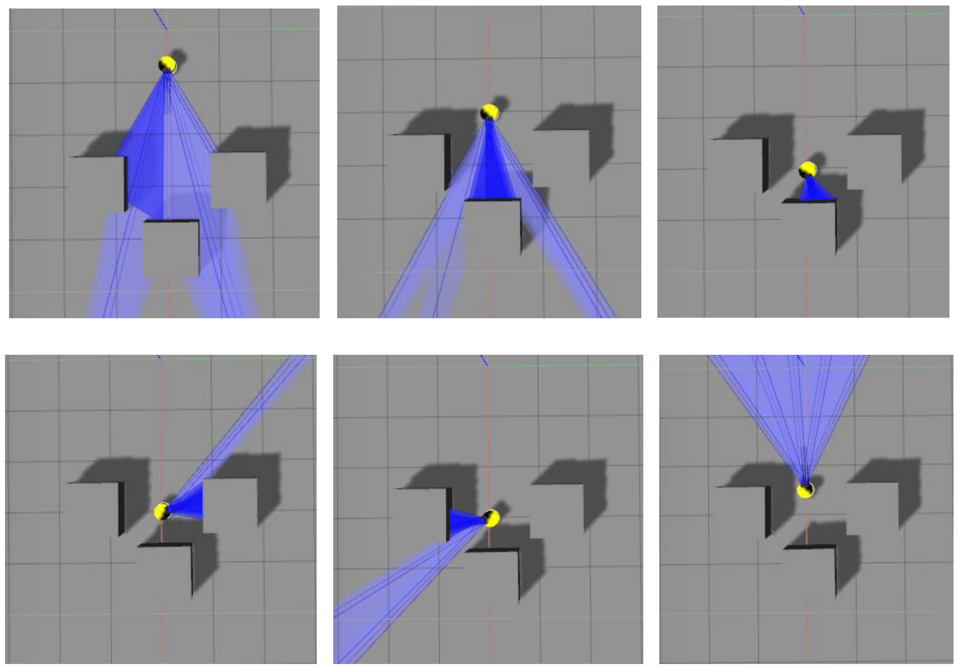
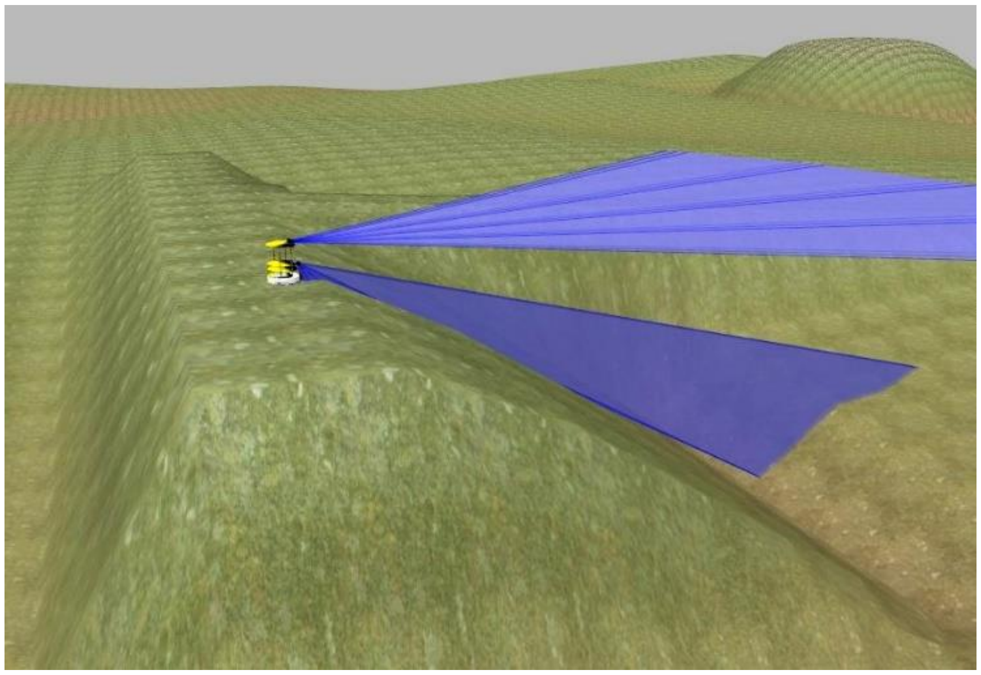


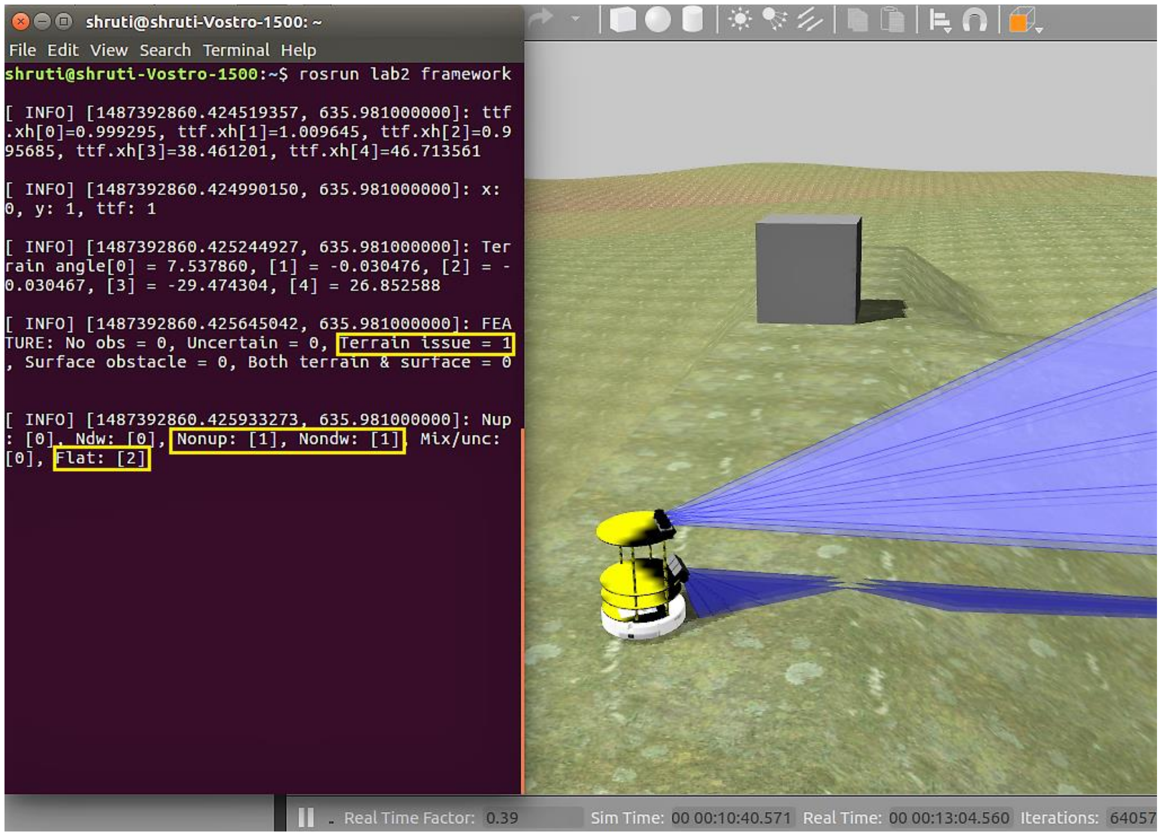
Figure 24: Robot navigating surface obstacles on a flat ground

On an uneven terrain, the robot must be able to detect terrain features along with surface obstacles. The irregular terrain created using heightmap as described in Section 4.1.2 is used to test the robot. The following are some of the scenarios encountered by the robot while navigating the terrain:

5.1 Non-Navigable Down Ramp

As marked in Figure 25, the robot detects a steep down ramp in the latter half of the cell. So, the cell is only partially flat and is not navigable.



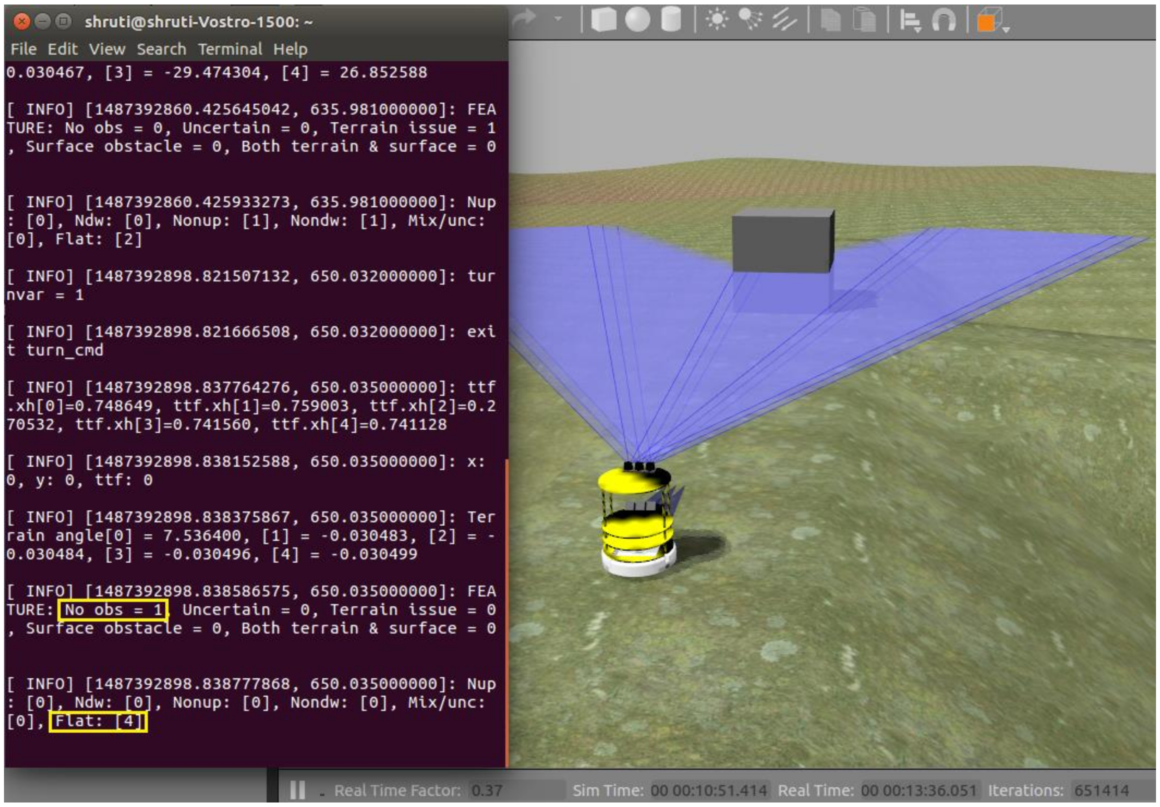

shruti@shruti-Vostro-1500: ~
File Edit View Search Terminal Help
Real Time Factor: 0.39
Sim Time: 00 00:10:40.571
Real Time: 00 00:13:04.560
Iterations: 64057

```
shruti@shruti-Vostro-1500:~$ rosrun lab2 framework

[ INFO] [1487392860.424519357, 635.981000000]: ttf
.xh[0]=0.999295, ttf.xh[1]=1.009645, ttf.xh[2]=0.9
95685, ttf.xh[3]=38.461201, ttf.xh[4]=46.713561

[ INFO] [1487392860.424990150, 635.981000000]: x:
0, y: 1, ttf: 1

[ INFO] [1487392860.425244927, 635.981000000]: Ter
rain angle[0] = 7.537860, [1] = -0.030476, [2] = -
0.030467, [3] = -29.474304, [4] = 26.852588

[ INFO] [1487392860.425645042, 635.981000000]: FEA
TURE: No obs = 0, Uncertain = 0, Terrain issue = 1
, Surface obstacle = 0, Both terrain & surface = 0


[ INFO] [1487392860.425933273, 635.981000000]: Nup
: [0], Ndw: [0], Nonup: [1], Nondw: [1], Mix/unc:
[0], Flat: [2]
```

Figure 25: Non-navigable down ramp detection

5.2 Flat Surface with No Obstacle

In Figure 26, although the LIDAR picks up the presence of a surface obstacle in the near vicinity, the framework decides that the next cell in the robot's path is flat and free of obstacles.



```
0.030467, [3] = -29.474304, [4] = 26.852588

[ INFO] [1487392860.425645042, 635.981000000]: FEA
TURE: No obs = 0, Uncertain = 0, Terrain issue = 1
, Surface obstacle = 0, Both terrain & surface = 0


[ INFO] [1487392860.425933273, 635.981000000]: Nup
: [0], Ndw: [0], Nonup: [1], Nondw: [1], Mix/unc:
[0], Flat: [2]

[ INFO] [1487392898.821507132, 650.032000000]: tur
nvar = 1

[ INFO] [1487392898.821666508, 650.032000000]: exi
t turn_cmd

[ INFO] [1487392898.837764276, 650.035000000]: ttf
.xh[0]=0.748649, ttf.xh[1]=0.759003, ttf.xh[2]=0.2
70532, ttf.xh[3]=0.741560, ttf.xh[4]=0.741128

[ INFO] [1487392898.838152588, 650.035000000]: x:
0, y: 0, ttf: 0

[ INFO] [1487392898.838375867, 650.035000000]: Ter
rain angle[0] = 7.536400, [1] = -0.030483, [2] = -
0.030484, [3] = -0.030496, [4] = -0.030499

[ INFO] [1487392898.838586575, 650.035000000]: FEA
TURE: No obs = 1, Uncertain = 0, Terrain issue = 0
, Surface obstacle = 0, Both terrain & surface = 0


[ INFO] [1487392898.838777868, 650.035000000]: Nup
: [0], Ndw: [0], Nonup: [0], Nondw: [0], Mix/unc:
[0], Flat: [4]
```

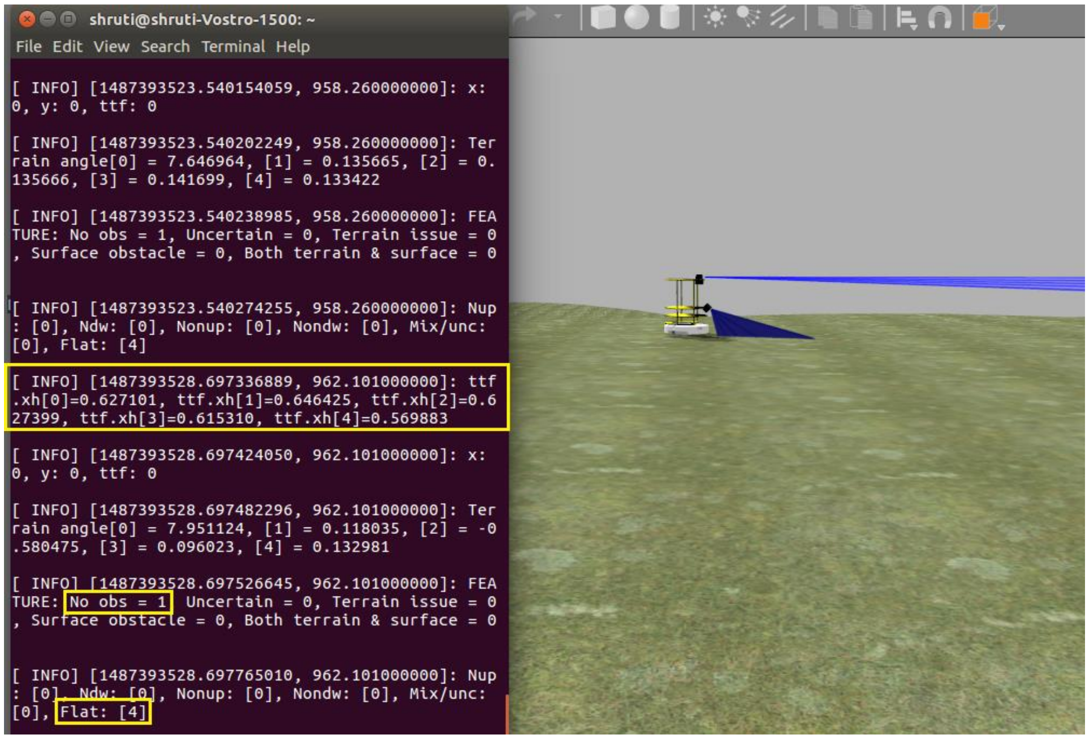
shruti@shruti-Vostro-1500: ~
File Edit View Search Terminal Help
Real Time Factor: 0.37
Sim Time: 00 00:10:51.414
Real Time: 00 00:13:36.051
Iterations: 651414

Figure 26: Robot navigating a flat surface with no obstacle

Figure 27 shows another scenario where the robot finds a flat surface without obstacles. It can be observed that the track-to-track fusion result is different from that in Figure 26, while they are both in the range for classification as "no obstacle".

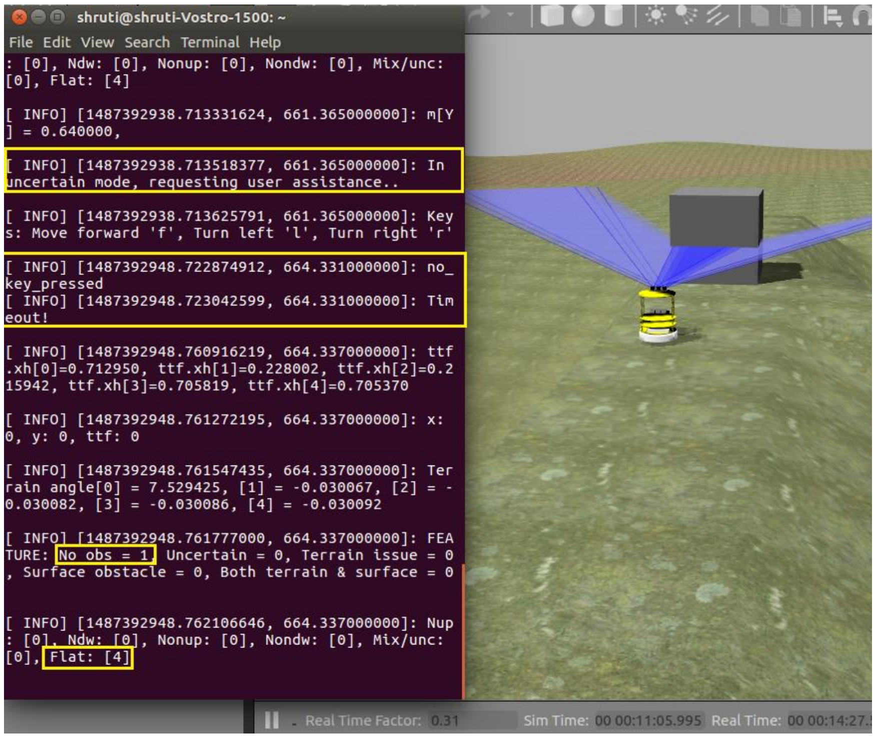
5.3 Uncertain Mode

When the robot enters an uncertain mode, the framework prompts for user assistance. As shown in Figure 28, the user does not respond within a preset time period. So, the robot takes sensor readings again to go through the fusion process. This time, the framework acquires higher confidence and estimates the state of the cell accurately as shown in Figure 28. In the second case, as shown in Figure 29, the user does not input a valid command. So, the framework prompts for a valid command again.


shruti@shruti-Vostro-1500: ~
File Edit View Search Terminal Help

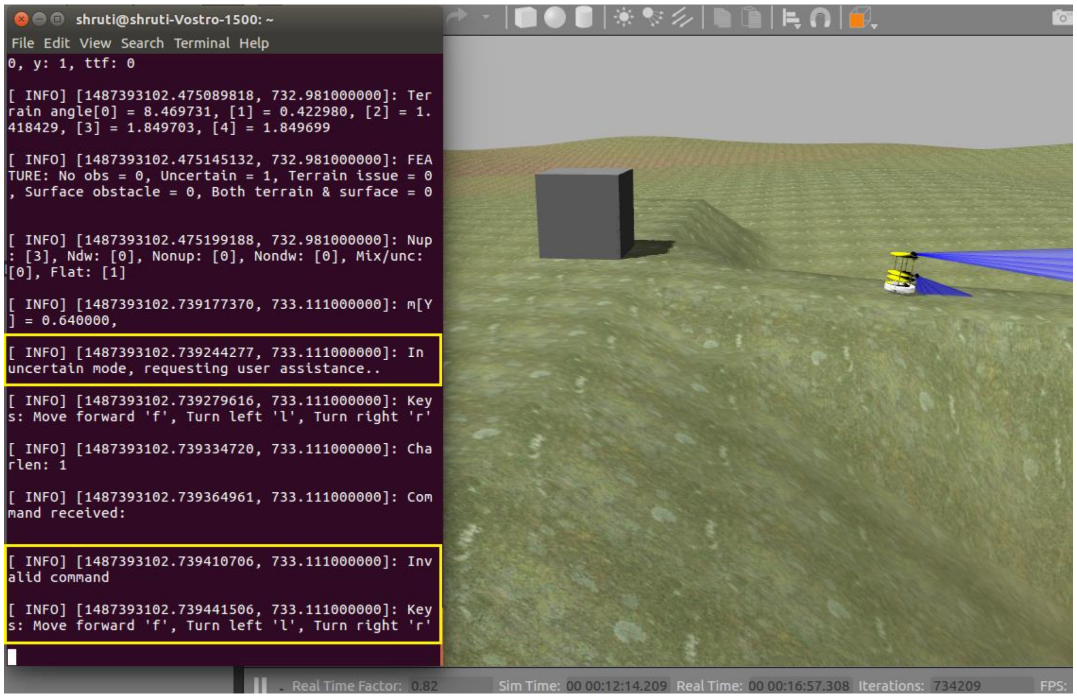
```
[ INFO] [1487393523.540154059, 958.260000000]: x:
0, y: 0, ttf: 0

[ INFO] [1487393523.540202249, 958.260000000]: Ter
rain angle[0] = 7.646964, [1] = 0.135665, [2] = 0.
135666, [3] = 0.141699, [4] = 0.133422

[ INFO] [1487393523.540238985, 958.260000000]: FEA
TURE: No obs = 1, Uncertain = 0, Terrain issue = 0
, Surface obstacle = 0, Both terrain & surface = 0


[ INFO] [1487393523.540274255, 958.260000000]: Nup
: [0], Ndw: [0], Nonup: [0], Nondw: [0], Mix/unc:
[0], Flat: [4]

[ INFO] [1487393528.697336889, 962.101000000]: ttf
.xh[0]=0.627101, ttf.xh[1]=0.646425, ttf.xh[2]=0.6
27399, ttf.xh[3]=0.615310, ttf.xh[4]=0.569883

[ INFO] [1487393528.697424050, 962.101000000]: x:
0, y: 0, ttf: 0

[ INFO] [1487393528.697482296, 962.101000000]: Ter
rain angle[0] = 7.951124, [1] = 0.118035, [2] = -0
.580475, [3] = 0.096023, [4] = 0.132981

[ INFO] [1487393528.697526645, 962.101000000]: FEA
TURE: No obs = 1  Uncertain = 0, Terrain issue = 0
, Surface obstacle = 0, Both terrain & surface = 0


[ INFO] [1487393528.697765010, 962.101000000]: Nup
: [0], Ndw: [0], Nonup: [0], Nondw: [0], Mix/unc:
[0], Flat: [4]
```

Figure 27: Robot navigating a flat surface


shruti@shruti-Vostro-1500: ~
File Edit View Search Terminal Help
Real Time Factor: 0.31
Sim Time: 00 00:11:05.995
Real Time: 00 00:14:27.

```
: [0], Ndw: [0], Nonup: [0], Nondw: [0], Mix/unc:
[0], Flat: [4]

[ INFO] [1487392938.713331624, 661.365000000]: m[Y
] = 0.640000,

[ INFO] [1487392938.713518377, 661.365000000]: In
uncertain mode, requesting user assistance..

[ INFO] [1487392938.713625791, 661.365000000]: Key
s: Move forward 'f', Turn left 'l', Turn right 'r'

[ INFO] [1487392948.722874912, 664.331000000]: no_
key_pressed
[ INFO] [1487392948.723042599, 664.331000000]: Tim
eout!

[ INFO] [1487392948.760916219, 664.337000000]: ttf
.xh[0]=0.712950, ttf.xh[1]=0.228002, ttf.xh[2]=0.2
15942, ttf.xh[3]=0.705819, ttf.xh[4]=0.705370

[ INFO] [1487392948.761272195, 664.337000000]: x:
0, y: 0, ttf: 0

[ INFO] [1487392948.761547435, 664.337000000]: Ter
rain angle[0] = 7.529425, [1] = -0.030067, [2] = -
0.030082, [3] = -0.030086, [4] = -0.030092

[ INFO] [1487392948.761777000, 664.337000000]: FEA
TURE: No obs = 1, Uncertain = 0, Terrain issue = 0
, Surface obstacle = 0, Both terrain & surface = 0


[ INFO] [1487392948.762106646, 664.337000000]: Nup
: [0], Ndw: [0], Nonup: [0], Nondw: [0], Mix/unc:
[0], Flat: [4]
```

Figure 28: Uncertain mode - no user input


shruti@shruti-Vostro-1500: ~
File Edit View Search Terminal Help
Real Time Factor: 0.82
Sim Time: 00 00:12:14.209
Real Time: 00 00:16:57.308
Iterations: 734209
FPS:

```
0, y: 1, ttf: 0

[ INFO] [1487393102.475089818, 732.981000000]: Ter
rain angle[0] = 8.469731, [1] = 0.422980, [2] = 1.
418429, [3] = 1.849703, [4] = 1.849699

[ INFO] [1487393102.475145132, 732.981000000]: FEA
TURE: No obs = 0, Uncertain = 1, Terrain issue = 0
, Surface obstacle = 0, Both terrain & surface = 0


[ INFO] [1487393102.475199188, 732.981000000]: Nup
: [3], Ndw: [0], Nonup: [0], Nondw: [0], Mix/unc:
[0], Flat: [1]

[ INFO] [1487393102.739177370, 733.111000000]: m[Y
] = 0.640000,

[ INFO] [1487393102.739244277, 733.111000000]: In
uncertain mode, requesting user assistance..

[ INFO] [1487393102.739279616, 733.111000000]: Key
s: Move forward 'f', Turn left 'l', Turn right 'r'

[ INFO] [1487393102.739334720, 733.111000000]: Cha
rlen: 1

[ INFO] [1487393102.739364961, 733.111000000]: Com
mand received:

[ INFO] [1487393102.739410706, 733.111000000]: Inv
alid command

[ INFO] [1487393102.739441506, 733.111000000]: Key
s: Move forward 'f', Turn left 'l', Turn right 'r'
```

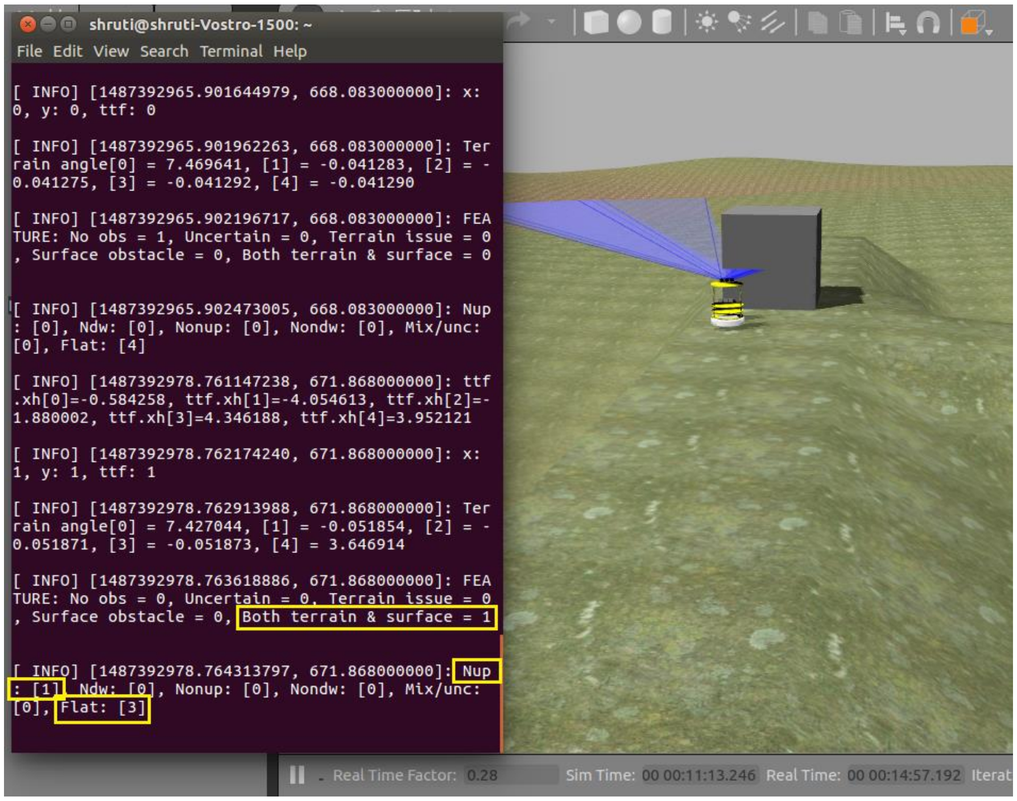
Figure 29: Uncertain mode - invalid input

5.4 Surface Obstacle Detection

In this case, the robot recognizes the surface obstacle in the next cell. The robot is closer to the obstacle, so the terrain sensor also detects the obstacle as an up ramp towards the end of the cell as shown in Figure 30.

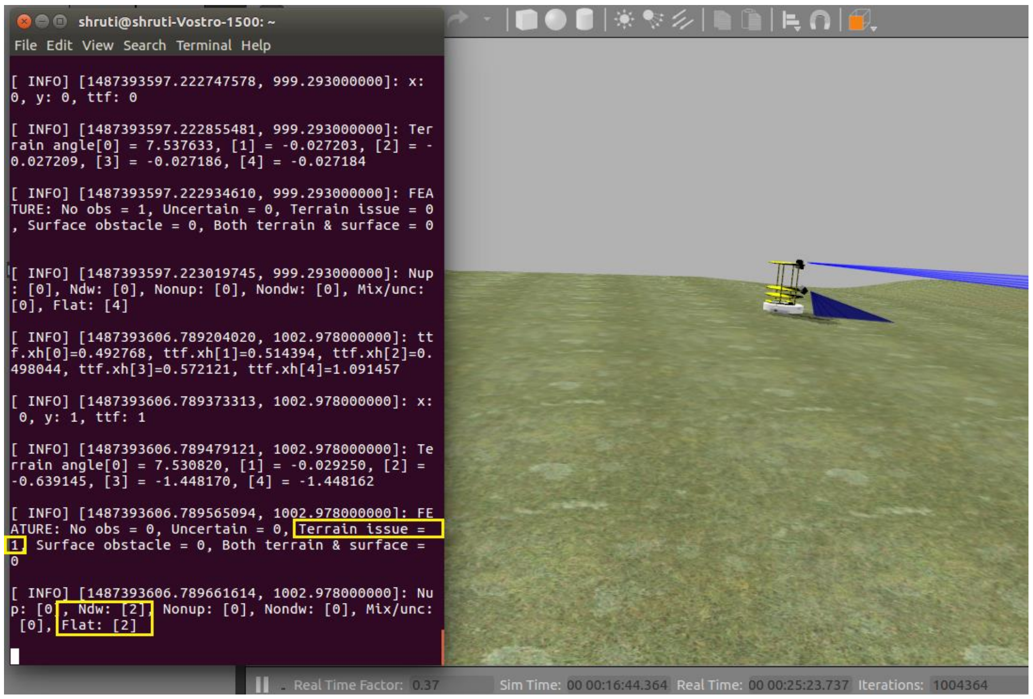
5.5 Navigable Down Ramp

Figure 31 explains this case where the robot encounters a navigable down ramp. A part of the cell is flat and then it slopes down as determined by the framework.


shruti@shruti-Vostro-1500: ~
File Edit View Search Terminal Help
Real Time Factor: 0.28
Sim Time: 00 00:11:13.246
Real Time: 00 00:14:57.192
Iterat

```
[ INFO] [1487392965.901644979, 668.083000000]: x:
0, y: 0, ttf: 0

[ INFO] [1487392965.901962263, 668.083000000]: Ter
rain angle[0] = 7.469641, [1] = -0.041283, [2] = -
0.041275, [3] = -0.041292, [4] = -0.041290

[ INFO] [1487392965.902196717, 668.083000000]: FEA
TURE: No obs = 1, Uncertain = 0, Terrain issue = 0
, Surface obstacle = 0, Both terrain & surface = 0


[ INFO] [1487392965.902473005, 668.083000000]: Nup
: [0], Ndw: [0], Nonup: [0], Nondw: [0], Mix/unc:
[0], Flat: [4]

[ INFO] [1487392978.761147238, 671.868000000]: ttf
.xh[0]=-0.584258, ttf.xh[1]=-4.054613, ttf.xh[2]=-
1.880002, ttf.xh[3]=4.346188, ttf.xh[4]=3.952121

[ INFO] [1487392978.762174240, 671.868000000]: x:
1, y: 1, ttf: 1

[ INFO] [1487392978.762913988, 671.868000000]: Ter
rain angle[0] = 7.427044, [1] = -0.051854, [2] = -
0.051871, [3] = -0.051873, [4] = 3.646914

[ INFO] [1487392978.763618886, 671.868000000]: FEA
TURE: No obs = 0, Uncertain = 0, Terrain issue = 0
, Surface obstacle = 0, Both terrain & surface = 1


[ INFO] [1487392978.764313797, 671.868000000]: Nup
: [1], Ndw: [0], Nonup: [0], Nondw: [0], Mix/unc:
[0], Flat: [3]
```

Figure 30: Surface obstacle detection

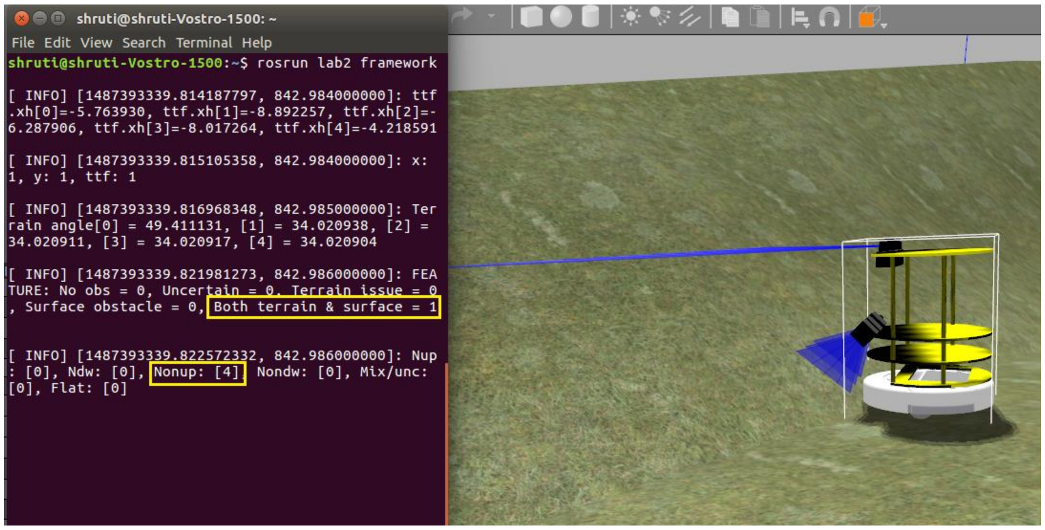

shruti@shruti-Vostro-1500: ~
File Edit View Search Terminal Help
Real Time Factor: 0.37
Sim Time: 00 00:16:44.364
Real Time: 00 00:25:23.737
Iterations: 1004364

```
[ INFO] [1487393597.222747578, 999.293000000]: x:
0, y: 0, ttf: 0

[ INFO] [1487393597.222855481, 999.293000000]: Ter
rain angle[0] = 7.537633, [1] = -0.027203, [2] = -
0.027209, [3] = -0.027186, [4] = -0.027184

[ INFO] [1487393597.222934610, 999.293000000]: FEA
TURE: No obs = 1, Uncertain = 0, Terrain issue = 0
, Surface obstacle = 0, Both terrain & surface = 0


[ INFO] [1487393597.223019745, 999.293000000]: Nup
: [0], Ndw: [0], Nonup: [0], Nondw: [0], Mix/unc:
[0], Flat: [4]

[ INFO] [1487393606.789204020, 1002.978000000]: tt
f.xh[0]=0.492768, ttf.xh[1]=0.514394, ttf.xh[2]=0.
498044, ttf.xh[3]=0.572121, ttf.xh[4]=1.091457

[ INFO] [1487393606.789373313, 1002.978000000]: x:
 0, y: 1, ttf: 1

[ INFO] [1487393606.789479121, 1002.978000000]: Te
rrain angle[0] = 7.530820, [1] = -0.029250, [2] =
-0.639145, [3] = -1.448170, [4] = -1.448162

[ INFO] [1487393606.789565094, 1002.978000000]: FE
ATURE: No obs = 0, Uncertain = 0, Terrain issue =
1, Surface obstacle = 0, Both terrain & surface =
0

[ INFO] [1487393606.789661614, 1002.978000000]: Nu
p: [0], Ndw: [2], Nonup: [0], Nondw: [0], Mix/unc:
 [0], Flat: [2]
```

Figure 31: Navigable down ramp

5.6 Non-Navigable Up Ramp

Figure 32 shows a scenario where the robot encounters a non-navigable up ramp. The framework's estimate is also marked.


shruti@shruti-Vostro-1500: ~
File Edit View Search Terminal Help

```
shruti@shruti-Vostro-1500:~$ rosrun lab2 framework

[ INFO] [1487393339.814187797, 842.984000000]: ttf
.xh[0]=-5.763930, ttf.xh[1]=-8.892257, ttf.xh[2]=-
6.287906, ttf.xh[3]=-8.017264, ttf.xh[4]=-4.218591

[ INFO] [1487393339.815105358, 842.984000000]: x:
1, y: 1, ttf: 1

[ INFO] [1487393339.816968348, 842.985000000]: Ter
rain angle[0] = 49.411131, [1] = 34.020938, [2] =
34.020911, [3] = 34.020917, [4] = 34.020904

[ INFO] [1487393339.821981273, 842.986000000]: FEA
TURE: No obs = 0, Uncertain = 0, Terrain issue = 0
, Surface obstacle = 0, Both terrain & surface = 1


[ INFO] [1487393339.822572332, 842.986000000]: Nup
: [0], Ndw: [0], Nonup: [4], Nondw: [0], Mix/unc:
[0], Flat: [0]
```

Figure 32: Non-navigable up ramp

CHAPTER 6: CONCLUSION AND FUTURE WORK

The simulation results discussed in Chapter 5, show that the framework can analyze LIDAR data to determine the state of cells in a terrain, aiding in autonomous robot navigation. The framework efficiently uses fusion algorithms (Kalman filter, track-to-track) to fuse the sensor signals and determine various features of the environment under consideration. The combination of methods used in the framework and the modular approach simplifies any customization efforts. It also makes the framework scalable to add more features, decisions and action sets. The framework works for various scenarios encountered in robot navigation as discussed in Chapter 5. It also accounts for human interaction while switching to autonomous mode if there is no response from the user. The framework is implemented on popular opensource platforms such as Gazebo and ROS.

There is a scope to develop the framework further by adding several other types of sensors like camera, IMUs, GPS and so on. Although fusion of camera and ranging technology (RADAR, LIDAR etc.) is a well explored area, it may be useful to include a variety of sensors depending on the application. A real-time decision evaluation would make the framework more robust and that data can be used to train the framework if such methods are considered. It would be useful to further classify the uncertain phase for the cause, for instance, sensor error, imbalance in weights in the fusion algorithm and so on. Robust fusion frameworks form the backbone of future autonomous system development.

REFERENCES

[1] S. S. Özer Çiftçioğlu, "Data Sensor Fusion for Autonomous Robotics," in *Serial and Parallel Robot Manipulators - Kinematics, Dynamics, Control and Optimization,* InTech, DOI: 10.5772/33139, 2012.

[2] R. Joshi and A. C. Sanderson, "Multisensor fusion: A minimal representation framework," *World Scientific,* vol. 11, 1999.

[3] R. C. Luo and C. C. Lai, "Multisensor fusion-based concurrent environment mapping and moving object detection for intelligent service robotics," *IEEE Transactions on Industrial Electronics,* vol. 61, no. 8, pp. 4043-4051, 2014.

[4] C. Axenie and J. Conradt, "Cortically inspired sensor fusion network for mobile robot egomotion estimation," *Robotics and Autonomous Systems,* vol. 71, pp. 69-82, 2015.

[5] F. Matía and Jiménez, "A. Multisensor fusion: an autonomous mobile robot," *Journal of Intelligent and robotic systems,* vol. 22, no. 2, pp. 129-141, 1998.

[6] B. Khaleghi, A. Khamis, F. O. Karray and S. N. Razavi, "Multisensor data fusion: A review of the state-of-the-art," *Information Fusion,* vol. 14, no. 1, pp. 28-44, 2013.

[7] A. Dallil, M. Oussalah and A. Ouldali, "Sensor fusion and target tracking using evidential data association," *IEEE sensors journal,* vol. 13, no. 1, pp. 285-293, 2013.

[8] Y. Bar-Shalom, P. K. Willett and X. Tian, *Tracking and data fusion,* YBS publishing, 2011.

[9] T. P. Banerjee and S. Das, "Multi-sensor data fusion using support vector machine for motor fault detection," *Information Sciences,* vol. 217, pp. 96-107, 2012.

[10] A. P. James and B. V. Dasarathy, "Medical image fusion: A survey of the state of the art," *Information Fusion,* vol. 19, pp. 4-19, 2014.

[11] N. E. El Faouzi, H. Leung and A. Kurian, "Data fusion in intelligent transportation systems: Progress and challenges–A survey," *Information Fusion,* vol. 12, no. 1, pp. 4-10, 2011.

[12] H. Cho, Y. W. Seo, B. V. Kumar and R. R. Rajkumar, "A multi-sensor fusion system for moving object detection and tracking in urban driving environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1836-1843, 2014.

[13] E. Cardarelli, L. Sabattini, C. Secchi and C. Fantuzzi, "Multisensor data fusion for obstacle detection in automated factory logistics," in *2014 IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp. 221-226.

[14] W. Wan, F. Lu, Z. Wu and K. Harada, "Teaching robots to do object assembly using multi-modal 3d vision," in *arXiv preprint arXiv:1601.06473*, 2016.

[15] J. Zhang, C. Song, Y. Hu and B. Yu, "Improving robustness of robotic grasping by fusing multi-sensor," in *IEEE Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pp. 126-131, 2012.

[16] J. Conrad, "ZapataBot - An Autonomous Robotic ATV," [Online]. Available: https://webpages.uncc.edu/~jmconrad/zapatabot.html.

[17] J. Dong, D. Zhuang, Y. Huang and J. Fu, "Advances in Multi-Sensor Data Fusion: Algorithms and Applications," *Sensors*, vol. 9, no. 10, pp. 7771-7784, 2009.

[18] B. Khaleghi et al., "Multisensor data fusion: A review of the state-of-the-art," *Informat. Fusion*, p. doi: 10.1016/j.inffus.2011.08.001, 2011.

[19] R. C. Luo and M. G. Kay, "Multisensor integration and fusion in intelligent systems," in *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5), 901-931, 1989.

[20] N. Xiong and P. Svensson, "Multi-sensor management for information fusion: issues and approaches," *Information fusion*, vol. 2, no. 3, pp. 163-186, 2002.

[21] H. B. Mitchell, "Multi-sensor data fusion: an introduction," *Springer Science & Business Media*, 2007.

[22] B. Chandrasekaran, S. Gangadhar and J. M. Conrad, "A Survey of Multisensor Fusion Techniques, Architectures and Methodologies," in *IEEE SoutheastCon 2017*, Charlotte, NC, 2017.

[23] H. F. Durrant-Whyte, "Sensor models and multisensor integration," *The international journal of robotics research*, vol. 7, no. 6, pp. 97-113, 1988.

[24] J. R. Raol, "Multi-Sensor Data Fusion with MATLAB®," *CRC Press,* 2009.

[25] D. L. Hall and J. Llinas, "An introduction to multisensor data fusion," *Proceedings of the IEEE,* vol. 85, no. 1, pp. 6-23, 1997.

[26] W. Elmenreich, "An introduction to sensor fusion," Vienna University of Technology, Austria, 2002.

[27] M. Markin, C. Harris, M. Bernhardt, J. Austin, M. Bedworth, P. Greenway, R. Johnston, A. Little and D. Lowe., "Technology foresight on data fusion and data processing," Publication, The Royal Aeronautical Society, 1997.

[28] B. V. Dasarathy, "Decision fusion," IEEE Computer Society Press, Los Alamitos, CA, 1994.

[29] O'Brien, J. and M. D. Bedworth, "The omnibus model: A new architecture for data fusion," in *Proceedings of the 2nd International Conference on Information Fusion (FUSION'99),* Helsinki, Finnland, July 1999.

[30] J. R. Boyd, "A discourse on winning and losing," Maxwell Air Force Base, AL: Air University. Library Document No. MU, 43947, 79, 1987.

[31] R. Alami, R. Chatila, S. Fleury, M. Ghallab and F. Ingrand, "An architecture for autonomy," *International Journal of Robotics Research,* vol. 17, no. 4, pp. 315-337, April 1998.

[32] F. Castanedo, "A review of data fusion techniques," *The Scientific World Journal,* 2013.

[33] M. A. Abidi and R. C. Gonzalez, *Data fusion in robotics and machine intelligence,* Academic Press Professional, Inc, 1992.

[34] S. Jingwei, Z. Yongjie, Z. Haiyun, Z. Tao, W. Leigang, R. Wei and L. Huifeng, "A multi-MEMS sensors information fusion algorithm," in *The 26th Chinese Control and Decision Conference (2014 CCDC),* pp. 4675-4680, IEEE, 2014.

[35] Y. Wang and S. Goodman, "Data Fusion with Neural Networks," in *International Conference on Systems Man and Cybernetics IEEE 1994,* vol. 1, pp. 640-645, Oct 1994.

[36] M. Kok., "Probabilistic modeling for sensor fusion with inertial measurements," Linköping studies in science and technology, Dissertations.No. 1814, ISBN 978-91-7685-621-5, 2016.

[37] N. Kovvali, C. Prior, K. Cizek, M. Galik, A. Diaz, E. Forzani and R. Tsui, "Least-squares based feature extraction and sensor fusion for explosive detection," in *IEEE International Conference on Acoustics, Speech and Signal Processing Proceedings. ICASSP*, pp. 2918-2921, 2010.

[38] G. D. Hager, S. P. Engelson and S. Atiya, "On comparing statistical and set-based methods in sensor data fusion," in *Proceedings IEEE International Conference on Robotics and Automation,* pp.352-358, vol.2, Atlanta, GA, 1993.

[39] H. Chen, "Research on multi-sensor data fusion technology based on PSO-RBF neural network," in *Advanced Information Technology, Electronic and Automation Control Conference (IAEAC),* Dec 2015.

[40] Y. H. Liang and W. M. Tian, "Multi-sensor Fusion Approach for Fire Alarm Using BP Neural Network," in *International Conference on Intelligent Networking and Collaborative Systems (INCoS),* pp. 99-102, Ostrawva, 2016.

[41] Q. Liu, X. Wang and N. S. V. Rao, "Artificial neural networks for estimation and fusion in long-haul sensor networks," in *18th International Conference on Information Fusion (Fusion),* pp. 460-467, Washington, DC, 2015.

[42] M. Munz, K. Dietmayer and M. Mählisch, "Generalized fusion of heterogeneous sensor measurements for multi target tracking," in *13th Conference on Information Fusion (FUSION),* pp. 1-8, 2010.

[43] B. Siciliano and O. Khatib, *(Eds.). Springer handbook of robotics.* Springer Science & Business Media, 2008.

[44] H. Wu, M. Siegel, R. Stiefelhagen and J. Yang, "Sensor fusion using Dempster-Shafer theory [for context-aware HCI]," in *Proceedings of the 19th IEEE In Instumentation and Measurement Technology Conference. IMTC/2002.* vol. 1, pp. 7-12, 2002.

[45] S. K. D. Challa, "Bayesian and dempster-shafer fusion," *Sadhana,* 29(2), 145-174, 2004.

[46] P. Dempster, "A Generalization of Bayesian Inference," *Journal of the Royal Statistical Society B,* vol. 30, no. 2, pp. 205–247, 1968.

[47] A. Shafer, "Mathematical Theory of Evidence," Princeton University Press, Princeton, NJ, USA, 1976.

[48] B. Chandrasekaran and J. M. Conrad, "Human-robot collaboration: A survey," in *IEEE SoutheastCon 2015*, pp. 1-8, 2015.

[49] B. Chandrasekaran and J. M. Conrad, "Sensor fusion using a selective sensor framework to achieve decision and task execution," in *IEEE SoutheastCon 2016*, 2016.

[50] B. Chandrasekaran and J. M. Conrad, "Complete Coverage Planning: Achieving Human Interaction and Maximum Coverage During an Autonomous Robotic Vehicle Navigation of an Unknown Terrain," in *AAAI Workshops*, 2017.

[51] Z. Rubo, G. Guochang and Z. Guoyin, "AUV obstacle-avoidance based on information fusion of multi-sensors," in *IEEE International Conference on Intelligent Processing Systems*, 1997.

[52] "Gazebo Components," Open Source Robotics Foundation, 2014. [Online]. Available: http://gazebosim.org/tutorials?tut=components.

[53] [Online]. Available: http://sdformat.org/.

[54] "Gazebo: Make a model," [Online]. Available: http://gazebosim.org/tutorials?tut=build_model.

[55] [Online]. Available: http://sauravag.com/category/tutorial/.

[56] [Online]. Available: http://playerstage.sourceforge.net/doc/Gazebo-manual-svn-html/tutorial_terrain.html.

[57] [Online]. Available: http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes.

APPENDIX A: SIMULATION SETUP FOR TESTING THE FRAMEWORK

ROS Kinetic Kame distribution requires the Ubuntu Xenial (16.04) platform to run on. This version of ROS interfaces with Gazebo v 7.1.

A.1 ROS Installation

The installation guide and documentation for the “Desktop-Full Install” configuration can be found at http://wiki.ros.org/kinetic/Installation/Ubuntu. The commands are as follows:

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >
/etc/apt/sources.list.d/ros-latest.list'
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key
0xB01FA116
$ sudo apt-get update
$ sudo apt-get install ros-kinetic-desktop-full
$ sudo rosdep init
$ rosdep update
$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
$ sudo apt-get install python-rosinstall
```

A.2 Gazebo Installation

The installation guide for Gazebo 7 can be found at:

http://gazebosim.org/tutorials?tut=install_ubuntu&cat=install

```
$ sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable
`lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-stable.list'
$ wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
$ sudo apt-get update
$ sudo apt-get install gazebo7
$ sudo apt-get install libgazebo7-dev
```

The ROS package to connect Gazebo and ROS is present in “gazebo_ros_pkgs”. The installation instructions can be found at http://gazebosim.org/tutorials?tut=ros_installing&cat=connect_ros.

The required packages can be installed with the following command:

```
sudo apt-get install ros-kinetic-gazebo-ros-pkgs ros-kinetic-gazebo-ros-control
```

A.3 CATKIN Workspace Setup

Catkin is used to build packages for ROS. A set of detailed instructions to set up the Catkin workspace can be found at http://wiki.ros.org/ROS/Tutorials. The following commands will set up the Catkin workspace in /home/username folder.

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
$ cd ~/catkin_ws/
$ catkin_make
$ cd
$ gedit ~/.bashrc
```

The following commands are to be saved in the .bashrc file:

source /opt/ros/kinetic/setup.bash

source ~/catkin_ws/devel/setup.bash

Additional ROS packages specific for navigation, mapping, visualization and robot control can be installed using the following command: sudo apt-get install pkgname

A list of some of the popular ROS packages:

ros-distro-navigation

ros-distro-slam-gmapping

ros-distro-rviz

ros-distro-ros-control

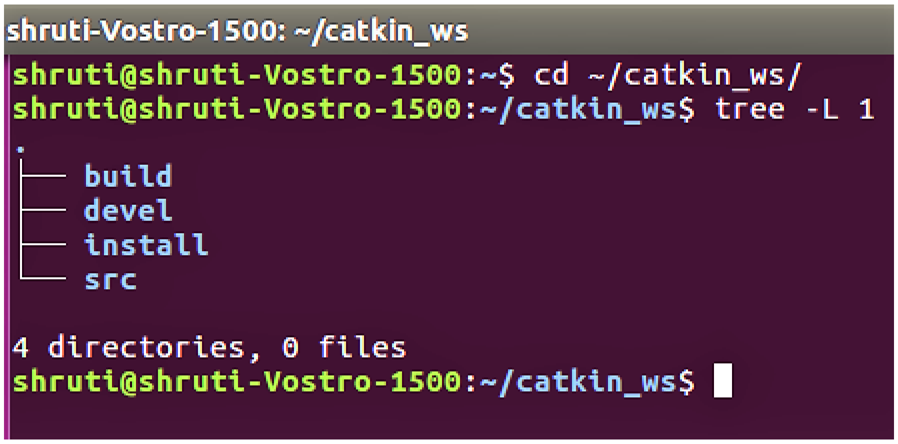
To understand the basic working of a ROS node, the following tutorial package will be useful: ros-distro-ros-tutorials

APPENDIX B: FRAMEWORK CODE AND GAZEBO SIMULATION

There are three main components required for the sensor fusion framework simulation. One, the robot itself, in this case a TurtleBot with 6 LIDARs. The description of the robot along with all its sensors is present in IRobotCreateHokuyo.sdf file. A robot has to navigate an irregular, unknown terrain. The description of the robot's world and terrain is in model.world file. It is launched by the custom_heightmap.launch file in Gazebo via ROS. That forms the second component. The third component is the sensor fusion framework itself. The source code for the framework is present in lab2/src folder in framework.cpp. Other dependencies are found in the same folder.

B.1 Basic Package Creation in ROS

ROS Catkin workspace has the following directory structure:



```
shruti-Vostro-1500: ~/catkin_ws
shruti@shruti-Vostro-1500:~$ cd ~/catkin_ws/
shruti@shruti-Vostro-1500:~/catkin_ws$ tree -L 1
.
├── build
├── devel
├── install
└── src

4 directories, 0 files
shruti@shruti-Vostro-1500:~/catkin_ws$
```

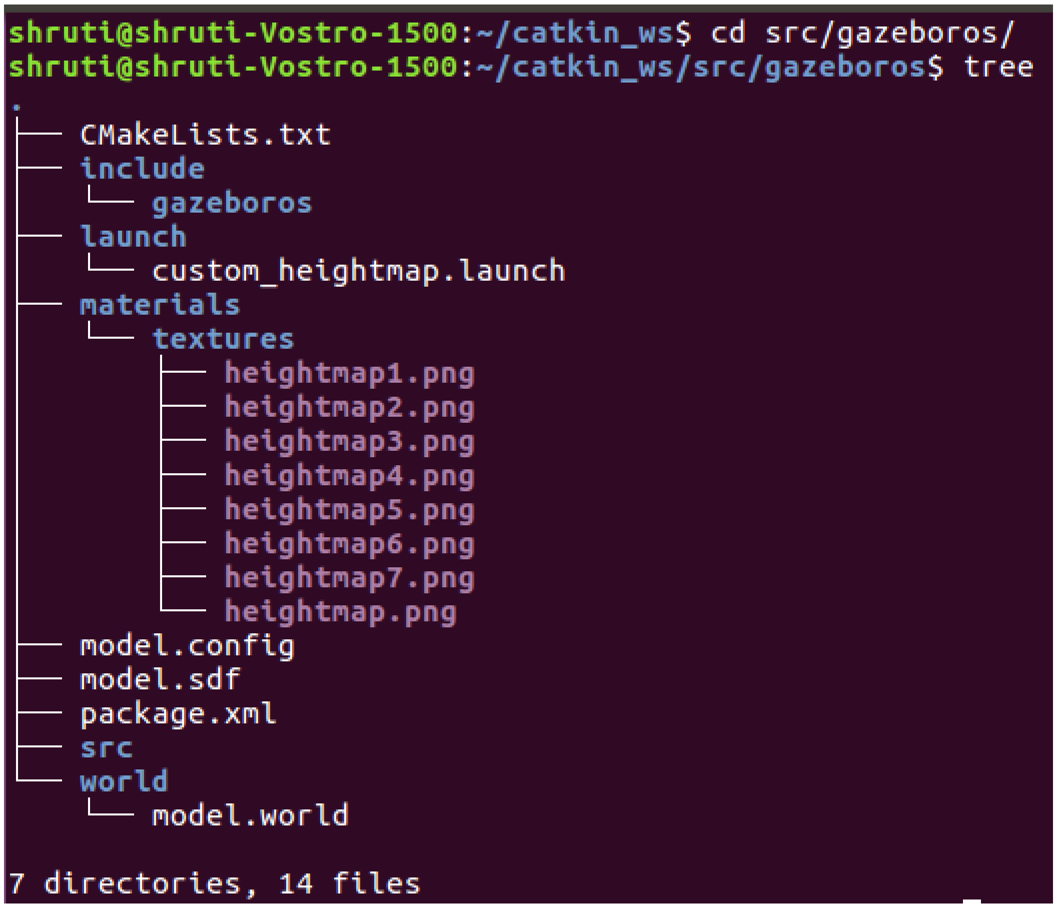
Figure 33: Catkin workspace directory structure

A new package is created in the ~/catkin_ws/src folder. Detailed procedure to create a package is explained in http://wiki.ros.org/ROS/Tutorials/catkin/CreatingPackage. The following commands will create the gazeboros package for making the robot's world:

$ cd ~/catkin_ws/src

$ catkin_create_pkg gazeboros gazebo_msgs gazebo_plugins gazebo_ros gazebo_ros_control std_msgs rospy roscpp

The above command creates a gazeboros folder in the catkin_ws/src directory that contains an include folder, a src folder, a Package.xml file and a CMakeLists.txt file. Package.xml contains the meta information whereas CmakeLists.txt contains the build information for the package.



```
shruti@shruti-Vostro-1500:~/catkin_ws$ cd src/gazeboros/
shruti@shruti-Vostro-1500:~/catkin_ws/src/gazeboros$ tree
.
├── CMakeLists.txt
├── include
│   └── gazeboros
├── launch
│   └── custom_heightmap.launch
├── materials
│   └── textures
│       ├── heightmap1.png
│       ├── heightmap2.png
│       ├── heightmap3.png
│       ├── heightmap4.png
│       ├── heightmap5.png
│       ├── heightmap6.png
│       ├── heightmap7.png
│       └── heightmap.png
├── model.config
├── model.sdf
├── package.xml
├── src
└── world
    └── model.world

7 directories, 14 files
```

Figure 34: gazeboros package file structure

Gazeboros package intends to include the information of the terrain that the robot navigates. The gazeboros package created by the catkin_create_pkg command does not hold this information. So, relevant files are included in the gazeboros package as shown in the file structure above. A world folder and a lunch folder is created and model.world and custom_heightmap.launch files are added respectively. A heightmap.png is placed in the materials/textures sub folder. model.sdf and model.config files are also added to the gazeboros folder. The Package.xml and CmakeLists.txt files are modified as required to include these changes. The following commands will build the new gazeboros package in the Catkin workspace:

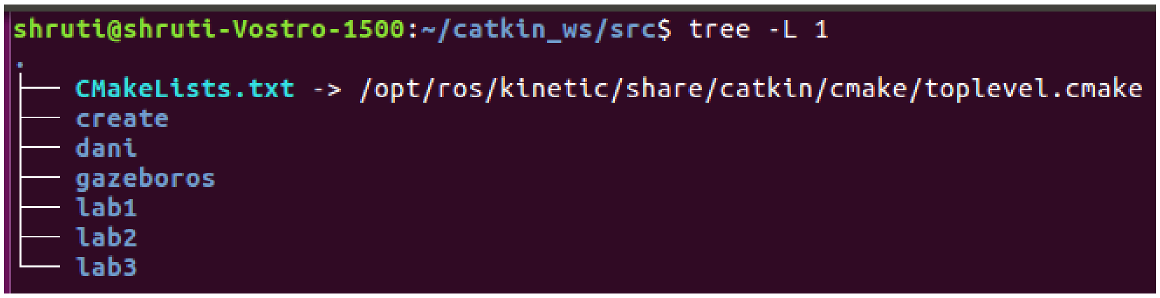
$ cd ~/catkin_ws

$ catkin_make install

The above mentioned procedure is repeated to install other parts of the sensor fusion framework package.

B.2 Package Directory

The directory structure for the entire framework in the catkin workspace is as shown below:



```
shruti@shruti-Vostro-1500:~/catkin_ws/src$ tree -L 1
.
├── CMakeLists.txt -> /opt/ros/kinetic/share/catkin/cmake/toplevel.cmake
├── create
├── dani
├── gazeboros
├── lab1
├── lab2
└── lab3
```

Figure 35: Package list

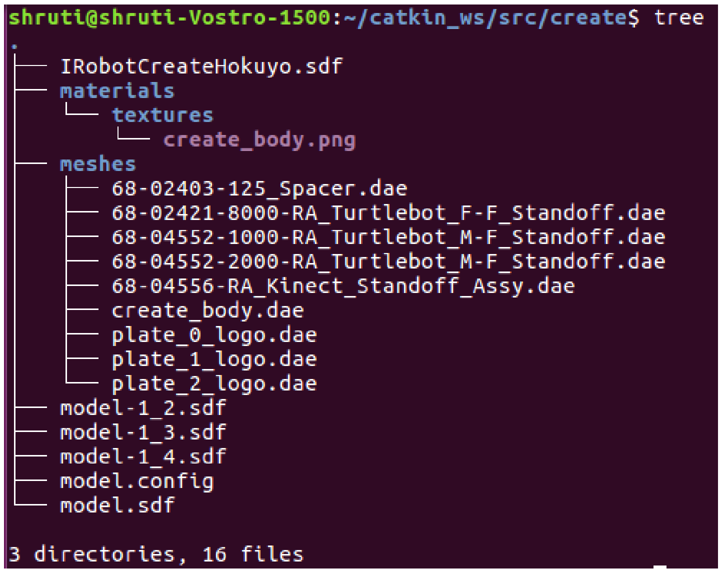
/ create – package that holds information about the robot model.

/lab2 – package that contains the entire sensor fusion framework code.

/gazeboros – package that contains files to launch a terrain in gazebo for the robot to navigate.

The file structure for some of the key packages of the framework:

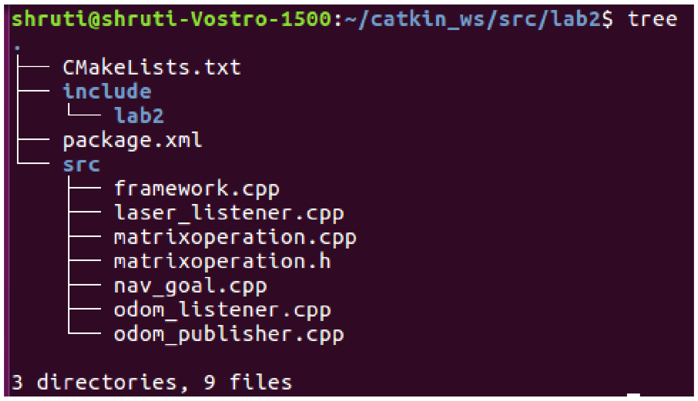
1. create



```
shruti@shruti-Vostro-1500:~/catkin_ws/src/create$ tree
.
├── IRobotCreateHokuyo.sdf
├── materials
│   └── textures
│       └── create_body.png
├── meshes
│   ├── 68-02403-125_Spacer.dae
│   ├── 68-02421-8000-RA_Turtlebot_F-F_Standoff.dae
│   ├── 68-04552-1000-RA_Turtlebot_M-F_Standoff.dae
│   ├── 68-04552-2000-RA_Turtlebot_M-F_Standoff.dae
│   ├── 68-04556-RA_Kinect_Standoff_Assy.dae
│   ├── create_body.dae
│   ├── plate_0_logo.dae
│   ├── plate_1_logo.dae
│   └── plate_2_logo.dae
├── model-1_2.sdf
├── model-1_3.sdf
├── model-1_4.sdf
├── model.config
└── model.sdf

3 directories, 16 files
```

Figure 36: Robot package

2. lab2



```
shruti@shruti-Vostro-1500:~/catkin_ws/src/lab2$ tree
.
├── CMakeLists.txt
├── include
│   └── lab2
├── package.xml
└── src
    ├── framework.cpp
    ├── laser_listener.cpp
    ├── matrixoperation.cpp
    ├── matrixoperation.h
    ├── nav_goal.cpp
    ├── odom_listener.cpp
    └── odom_publisher.cpp

3 directories, 9 files
```

Figure 37: Framework package

APPENDIX C: USER GUIDE

To run the simulation,

1. Initiate roscore in a terminal with the following command:

```
$ roscore
```

2. In another terminal, specify the path for the Gazebo files related to the robot environment model using the following command:

```
$ export GAZEBO_MODEL_PATH=/home/username/catkin_ws/src/
```

Then invoke Gazebo through roslaunch using the following command:

```
$ roslaunch gazeboros custom_heightmap.launch
```

This should launch Gazebo with the custom terrain.

3. In a third terminal, bring up the robot model at a specific coordinate position on the custom terrain with the following command:

```
$ rosrun gazebo_ros spawn_model -file
/home/username/catkin_ws/src/create/IRobotCreateHokuyo.sdf -sdf -x 0 -y 5 -z 5 -model
create
```

If coordinate is not mentioned, Gazebo will spawn the model at origin. The z coordinate must be mentioned so that the robot is on the custom terrain surface.

4. In the above terminal, run the framework package using rosrun to examine the sensor fusion framework.

```
$ rosrun lab2 framework
```

Note: “username” in the commands should be replaced with the actual user name of the machine.

APPENDIX D: ROS MESSAGES

D.1 Laser Scan Message for LIDAR Data

sensor_msgs/LaserScan.msg

```
# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior (e.g. a sonar
# array), please find or create a different message, since applications
# will make fairly laser-specific assumptions about this data

Header header              # timestamp in the header is the acquisition time of
                           # the first ray in the scan.
                           #
                           # in frame frame_id, angles are measured around
                           # the positive Z axis (counterclockwise, if Z is up)
                           # with zero angle being forward along the x axis

float32 angle_min          # start angle of the scan [rad]
float32 angle_max          # end angle of the scan [rad]
float32 angle_increment    # angular distance between measurements [rad]

float32 time_increment     # time between measurements [seconds] - if your
                           # scanner is moving, this will be used in interpolating
                           # position of 3d points
float32 scan_time          # time between scans [seconds]

float32 range_min          # minimum range value [m]
float32 range_max          # maximum range value [m]

float32[] ranges           # range data [m] (Note: values < range_min or >
                           # range_max should be discarded)
float32[] intensities      # intensity data [device-specific units].  If your
                           # device does not provide intensities, please leave
                           # the array empty.
```

std_msgs/Header.msg

```
# Standard metadata for higher-level stamped data types.
# This is generally used to communicate timestamped data
# in a particular coordinate frame.
#
# sequence ID: consecutively increasing ID
uint32 seq
# Two-integer timestamp that is expressed as:
# * stamp.sec: seconds (stamp_secs) since epoch (in Python the variable is called
# 'secs')
# * stamp.nsec: nanoseconds since stamp_secs (in Python the variable is called
# 'nsecs')
# time-handling sugar is provided by the client library
time stamp
# Frame this data is associated with
# 0: no frame
# 1: global frame
string frame_id
```

D.2 Command Velocity for the Robot

geometry_msgs/Twist.msg

```
# This expresses velocity in free space broken into its linear and angular parts.
Vector3  linear
Vector3  angular
```

geometry_msgs/Vector3.msg

```
# This represents a vector in free space.
# It is only meant to represent a direction. Therefore, it does not
# make sense to apply a translation to it (e.g., when applying a
# generic rigid transformation to a Vector3, tf2 will only apply the
# rotation). If you want your data to be translatable too, use the
# geometry_msgs/Point message instead.

float64 x
float64 y
float64 z
```

D.3 Odometry Information of the Robot

nav_msgs/Odometry.msg

```
# This represents an estimate of a position and velocity in free space.
# The pose in this message should be specified in the coordinate frame given by
# header.frame_id.
# The twist in this message should be specified in the coordinate frame given by the
# child_frame_id
Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

geometry_msgs/PoseWithCovariance.msg

```
# This represents a pose in free space with uncertainty.

Pose pose

# Row-major representation of the 6x6 covariance matrix
# The orientation parameters use a fixed-axis representation.
# In order, the parameters are:
# (x, y, z, rotation about X axis, rotation about Y axis, rotation about Z axis)
float64[36] covariance
```

geometry_msgs/Pose.msg

```
# A representation of pose in free space, composed of position and orientation.
Point position
Quaternion orientation
```

geometry_msgs/Point.msg

```
# This contains the position of a point in free space
float64 x
float64 y
float64 z
```

geometry_msgs/Quaternion.msg

```
# This represents an orientation in free space in quaternion form.

float64 x
float64 y
float64 z
float64 w
```

geometry_msgs/TwistWithCovariance.msg

```
# This expresses velocity in free space with uncertainty.

Twist twist

# Row-major representation of the 6x6 covariance matrix
# The orientation parameters use a fixed-axis representation.
# In order, the parameters are:
# (x, y, z, rotation about X axis, rotation about Y axis, rotation about Z axis)
float64[36] covariance
```

D.4 Transformation Configuration

geometry_msgs/Transformation.msg

```
# This represents the transform between two coordinate frames in free space.

Vector3 translation
Quaternion rotation
```

APPENDIX E: Track-to-Track Fusion Test Run

Obstacle sensor (r1)	Obstacle sensor (r2)	Terrain sensor (r1)	Terrain sensor (r2)	Track fusion w n=1	Track fusion w n=10	Track fusion w n=100	Track fusion w n=1000
70	71	0.5	0.492	0.717014	0.717014	0.717014	0.717014
70	70.3	0.58	0.57	0.603262	0.603262	0.603262	0.603262
70	72	0.76	0.771	0.867614	0.867614	0.867614	0.867614
70	75	0.91	0.922	0.894381	0.894381	0.894381	0.894381
70	70.8	1	1.014	0.824363	0.824363	0.824363	0.824363
70	70.99	0.6	0.595	1.052728	1.052728	1.052728	1.052728
70	68.4	0.5	0.499	0.063133	0.063133	0.063133	0.063133
70	70	0.5	0.5	-0.694836	-0.694836	-0.694836	-0.69483
70	70	0.5	0.501	-1.441198	-1.441198	-1.441198	-1.44119
70	70	0.5	0.48	-1.831603	-1.831603	-1.831603	-1.83160
70	74	0.5	0.517	0.667229	0.667229	0.667229	0.667229
70	65.89	0.7	0.686	1.373856	1.373856	1.373856	1.373856
70	73	0.88	0.893	1.337094	1.337094	1.337094	1.337094
70	72.86	1.2	1.219	2.149016	2.149016	2.149016	2.149016
70	70	1.5	1.483	3.336988	3.336988	3.336988	3.336988
71	70.5	0.5	0.504	0.706144	0.706144	0.706144	0.706144
72	70	0.58	0.573	0.680001	0.680001	0.680001	0.680001
70	70	0.76	0.758	0.774396	0.774396	0.774396	0.774396
73.8	71.54	0.91	0.917	0.774373	0.774373	0.774373	0.774373
75	73.44	1	1.01	0.713521	0.713521	0.713521	0.713521
71	70	0.6	0.605	8.741328	8.741328	8.741328	8.741328
72	70	0.5	0.488	7.848506	7.848506	7.848506	7.848506
70	70	0.5	0.499	7.020599	7.020599	7.020599	7.020599
73.8	70.21	0.5	0.485	6.400338	6.400338	6.400338	6.400338
75	72.09	0.5	0.501	5.828516	5.828516	5.828516	5.828516
71	70	0.5	0.501	1.476156	1.476156	1.476156	1.476156
72	72	0.7	0.699	1.98412	1.98412	1.98412	1.98412
70	70	0.88	0.886	2.083873	2.083873	2.083873	2.083873
73.8	72.9	1.2	1.185	3.069411	3.069411	3.069411	3.069411
75	74.44	1.5	1.517	3.911797	3.911797	3.911797	3.911797
70	74.01	0.5	0.513	0.507794	0.507794	0.507794	0.507794
3.4	3.41	0.58	0.578	0.025165	0.025165	0.025165	0.025165
70	70.02	0.76	0.747	0.717631	0.717631	0.717631	0.717631
9	10	0.91	0.914	0.179159	0.179159	0.179159	0.179159
70	73.08	1	0.989	0.635053	0.635053	0.635053	0.635053
70	70	0.6	0.58	0.313611	0.313611	0.313611	0.313611
3.4	3.38	0.5	0.52	-1.387129	-1.387129	-1.387129	-1.38712

70	71	0.5	0.509	-1.621879	-1.621879	-1.621879	-1.62187
9	9.01	0.5	0.501	-2.776353	-2.776353	-2.776353	-2.77635
70	74	0.5	0.501	-2.875135	-2.875135	-2.875135	-2.87513
70	69.87	0.5	0.509	0.584901	0.584901	0.584901	0.584901
3.4	3.391	0.7	0.697	1.649815	1.649815	1.649815	1.649815
70	72.34	0.88	0.887	1.927661	1.927661	1.927661	1.927661
9	8.99	1.2	1.202	3.712918	3.712918	3.712918	3.712918
70	72	1.5	1.495	5.749237	5.749237	5.749237	5.749237
2	2.8	0.5	0.511	0.573909	0.573909	0.573909	0.573909
70	70	0.58	0.565	0.335452	0.335452	0.335452	0.335452
70	69.5	0.76	0.769	0.645258	0.645258	0.645258	0.645258
70	72	0.91	0.898	0.645131	0.645131	0.645131	0.645131
3.4	3.6	1	1.013	0.573262	0.573262	0.573262	0.573262
2	1.99	0.6	0.598	2.883175	2.883175	2.883175	2.883175
70	70	0.5	0.514	0.787328	0.787328	0.787328	0.787328
70	70.7	0.5	0.48	-0.761037	-0.761037	-0.761037	-0.76103
70	70.7	0.5	0.518	-2.30888	-2.30888	-2.30888	-2.30888
3.4	3.4	0.5	0.519	-3.309816	-3.309816	-3.309816	-3.30981
2	2.01	0.5	0.502	-0.746924	-0.746924	-0.746924	-0.74692
70	70	0.7	0.698	0.253327	0.253327	0.253327	0.253327
70	70.36	0.88	0.878	0.562955	0.562955	0.562955	0.562955
70	72.41	1.2	1.187	2.317433	2.317433	2.317433	2.317433
3.4	3.05	1.5	1.504	4.412834	4.412834	4.412834	4.412834
70	70	0.5	0.495	0.733685	0.644195	-0.250708	-9.19974
0.2	0.1999	0.58	0.56	0.559998	0.477335	-0.349291	-8.61555
0.3	0.3054	0.76	0.741	0.869433	0.785001	-0.059318	-8.50250
70	75	0.91	0.912	0.836999	0.747509	-0.147395	-9.09642
70	72	1	1.004	0.733773	0.644283	-0.25062	-9.19965
70	69.28	0.6	0.591	2.63059	2.592554	2.212192	-1.59142
0.2	0.205	0.5	0.503	0.599365	0.568157	0.256077	-2.86472
0.3	0.4	0.5	0.504	-0.948944	-0.981912	-1.311598	-4.60846
70	70.33	0.5	0.48	-2.529881	-2.567917	-2.948279	-6.75189
70	73.22	0.5	0.483	-3.561908	-3.599944	-3.980306	-7.78392
70	71	0.5	0.484	-0.201502	-0.240792	-0.633689	-4.56266
0.2	0.28	0.7	0.709	0.863591	0.831137	0.506594	-2.73883
0.3	0.28	0.88	0.881	1.172959	1.138725	0.796387	-2.62699
70	74.05	1.2	1.212	2.894978	2.855688	2.462791	-1.46618
70	70.69	1.5	1.491	4.958947	4.919658	4.52676	0.597784
70	70	0.5	0.517	-15.03566	-15.52219	-20.38747	-69.0403
70	73.49	0.58	0.582	-15.24214	-15.72866	-20.59392	-69.2464
0	0.09	0.76	0.757	-14.89981	-15.38001	-20.18199	-68.2017
70	70	0.91	0.915	-14.93252	-15.41905	-20.28430	-68.9367
71	72.06	1	1.011	-15.03616	-15.52268	-20.38796	-69.0407
70	65	0.6	0.589	0.959364	0.9585	0.949859	0.86345

70	68	0.5	0.483	-1.106415	-1.107279	-1.11592	-1.20232
0	0.2	0.5	0.504	-2.621958	-2.616322	-2.559962	-1.99636
70	73.91	0.5	0.49	-4.204949	-4.205813	-4.214454	-4.30086
71	72.18	0.5	0.48	-5.238415	-5.239279	-5.24792	-5.33432
70	69.19	0.5	0.503	-0.461712	-0.482869	-0.694439	-2.81013
70	70	0.7	0.712	0.570478	0.549321	0.337751	-1.77794
0	0	0.88	0.879	0.912834	0.898007	0.749729	-0.73304
70	70.6	1.2	1.18	2.634478	2.613321	2.401751	0.286055
71	70.21	1.5	1.519	4.698512	4.677355	4.465785	2.350089
70	71	0.5	0.492	0.766627	0.766627	0.766627	0.766627
70	70.3	0.58	0.57	0.560189	0.560189	0.560189	0.560189
70	72	0.76	0.771	0.869986	0.869986	0.869986	0.869986
70	75	0.91	0.922	0.870026	0.870026	0.870026	0.870026
70	70.8	1	1.014	0.766787	0.766787	0.766787	0.766787
70	70.99	0.6	0.595	3.338835	3.338835	3.338835	3.338835
70	68.4	0.5	0.499	1.274676	1.274676	1.274676	1.274676
70	70	0.5	0.5	-0.27342	-0.27342	-0.27342	-0.27342
70	70	0.5	0.501	-1.821533	-1.821533	-1.821533	-1.82153
70	70	0.5	0.48	-2.853768	-2.853768	-2.853768	-2.85376
70	74	0.5	0.517	-1.301534	-1.301534	-1.301534	-1.30153
70	65.89	0.7	0.686	-0.269769	-0.269769	-0.269769	-0.26976
70	73	0.88	0.893	0.04013	0.04013	0.04013	0.04013
70	72.86	1.2	1.219	1.79471	1.79471	1.79471	1.79471
70	70	1.5	1.483	3.858574	3.858574	3.858574	3.858574
71	70.5	0.5	0.504	0.729088	0.729088	0.729088	0.729088
72	70	0.58	0.573	0.522101	0.522101	0.522101	0.522101
70	70	0.76	0.758	0.832724	0.832724	0.832724	0.832724
73.8	71.54	0.91	0.917	0.830978	0.830978	0.830978	0.830978
75	73.44	1	1.01	0.72725	0.72725	0.72725	0.72725
71	70	0.6	0.605	3.038085	3.038085	3.038085	3.038085
72	70	0.5	0.488	0.973315	0.973315	0.973315	0.973315
70	70	0.5	0.499	-0.573762	-0.573762	-0.573762	-0.57376
73.8	70.21	0.5	0.485	-2.123812	-2.123812	-2.123812	-2.12381
75	72.09	0.5	0.501	-3.156317	-3.156317	-3.156317	-3.15631
71	70	0.5	0.501	-1.408945	-1.408945	-1.408945	-1.40894
72	72	0.7	0.699	-0.377352	-0.377352	-0.377352	-0.37735
70	70	0.88	0.886	-0.066726	-0.066726	-0.066726	-0.06672
73.8	72.9	1.2	1.185	1.685844	1.685844	1.685844	1.685844
75	74.44	1.5	1.517	3.749707	3.749707	3.749707	3.749707
70	74.01	0.5	0.513	0.386843	0.386843	0.386843	0.386843
3.4	3.41	0.58	0.578	0.21223	0.21223	0.21223	0.21223
70	70.02	0.76	0.747	0.489896	0.489896	0.489896	0.489896
9	10	0.91	0.914	0.519449	0.519449	0.519449	0.519449
70	73.08	1	0.989	0.386656	0.386656	0.386656	0.386656

70	70	0.6	0.58	2.818546	2.818546	2.818546	2.818546
3.4	3.38	0.5	0.52	0.785978	0.785978	0.785978	0.785978
70	71	0.5	0.509	-0.793631	-0.793631	-0.793631	-0.79363
9	9.01	0.5	0.501	-2.313143	-2.313143	-2.313143	-2.31314
70	74	0.5	0.501	-3.373945	-3.373945	-3.373945	-3.37394
70	69.87	0.5	0.509	-0.585928	-0.585928	-0.585928	-0.58592
3.4	3.391	0.7	0.697	0.477402	0.477402	0.477402	0.477402
70	72.34	0.88	0.887	0.75579	0.75579	0.75579	0.75579
9	8.99	1.2	1.202	2.538968	2.538968	2.538968	2.538968
70	72	1.5	1.495	4.574401	4.574401	4.574401	4.574401
2	2.8	0.5	0.511	0.594133	0.594133	0.594133	0.594133
70	70	0.58	0.565	0.355717	0.355717	0.355717	0.355717
70	69.5	0.76	0.769	0.665513	0.665513	0.665513	0.665513
70	72	0.91	0.898	0.665385	0.665385	0.665385	0.665385
3.4	3.6	1	1.013	0.593487	0.593487	0.593487	0.593487
2	1.99	0.6	0.598	4.518887	4.518887	4.518887	4.518887
70	70	0.5	0.514	2.423045	2.423045	2.423045	2.423045
70	70.7	0.5	0.48	0.874682	0.874682	0.874682	0.874682
70	70.7	0.5	0.518	-0.673159	-0.673159	-0.673159	-0.67315
3.4	3.4	0.5	0.519	-1.674094	-1.674094	-1.674094	-1.67409
2	2.01	0.5	0.502	-0.330007	-0.330007	-0.330007	-0.33000
70	70	0.7	0.698	0.670244	0.670244	0.670244	0.670244
70	70.36	0.88	0.878	0.979872	0.979872	0.979872	0.979872
70	72.41	1.2	1.187	2.734349	2.734349	2.734349	2.734349
3.4	3.05	1.5	1.504	4.82975	4.82975	4.82975	4.82975
70	70	0.5	0.495	0.978172	0.956245	0.736969	-1.45578
0.2	0.1999	0.58	0.56	0.804484	0.789385	0.638386	-0.87159
0.3	0.3054	0.76	0.741	1.113919	1.09705	0.928359	-0.75855
70	75	0.91	0.912	1.081486	1.059558	0.840283	-1.35247
70	72	1	1.004	0.97826	0.956332	0.737057	-1.45569
70	69.28	0.6	0.591	15.601486	15.030738	9.323252	-47.7516
0.2	0.205	0.5	0.503	13.57028	13.006363	7.367196	-49.0244
0.3	0.4	0.5	0.504	12.021955	11.456274	5.799464	-50.7686
70	70.33	0.5	0.48	10.441041	9.870297	4.162858	-52.911
70	73.22	0.5	0.483	9.408993	8.838245	3.130768	-53.9439
70	71	0.5	0.484	0.449672	0.449412	0.446803	0.420723
0.2	0.28	0.7	0.709	1.516391	1.523137	1.590593	2.265153
0.3	0.28	0.88	0.881	1.825994	1.830916	1.880134	2.37231
70	74.05	1.2	1.212	3.548562	3.548301	3.545693	3.519612
70	70.69	1.5	1.491	5.614137	5.613876	5.611268	5.585187
70	70	0.5	0.517	0.644774	0.623722	0.4132	-1.69201
70	73.49	0.58	0.582	0.438278	0.417226	0.206704	-1.89851
0	0.09	0.76	0.757	0.780657	0.765943	0.6188	-0.85262
70	70	0.91	0.915	0.747897	0.726845	0.516323	-1.58889

71	72.06	1	1.011	0.644273	0.623221	0.412699	-1.69252
70	65	0.6	0.589	2.956377	2.93566	2.728494	0.656831
70	68	0.5	0.483	0.892201	0.871484	0.664318	-1.40734
0	0.2	0.5	0.504	-0.622959	-0.637334	-0.781079	-2.21853
70	73.91	0.5	0.49	-2.203929	-2.224646	-2.431812	-4.50347
71	72.18	0.5	0.48	-3.236582	-3.257299	-3.464465	-5.53612
70	69.19	0.5	0.503	-0.66732	-0.692992	-0.949704	-3.51682
70	70	0.7	0.712	0.364835	0.339164	0.082451	-2.48467
0	0	0.88	0.879	0.707146	0.687797	0.494309	-1.44057
70	70.6	1.2	1.18	2.428766	2.403095	2.146383	-0.42073
71	70.21	1.5	1.519	4.492732	4.46706	4.210348	1.643226