A SELECTIVE SENSOR FRAMEWORK USING SENSOR FUSION AND SENSOR MAPS TO ACHIEVE COMPLETE COVERAGE PLANNING OF A SEMI-AUTONOMOUS ROBOTIC VEHICLE

by

Balasubramaniyan Chandrasekaran

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Electrical Engineering

Charlotte

2017

Approved by:

_____
Dr. James Conrad

_____
Dr. Yogendra Kakad

_____
Dr. Hamed Tabkhi

_____
Dr. Aidan Browne

ABSTRACT

BALASUBRAMANIYAN CHANDRASEKARAN. A selective sensor framework using sensor fusion and sensor maps to achieve complete coverage planning of a semi-autonomous robotic vehicle. (Under the direction of DR. JAMES M. CONRAD)

Autonomous vehicles are increasingly used in many applications, such as elderly care, medical care, military, schools, and space exploration. In this research, we introduce the use of a framework for achieving the autonomy in a vehicle and describe the usage of sensor maps in managing the sensor space of a vehicle. Complete coverage planning is a new technique to achieve maximum coverage of a space during navigation. Path planning plays a key role in autonomous vehicle navigation and is achieved through several standard algorithms. In this research, a sensor fusion framework is formulated that utilizes the concept of complete coverage planning improvised it to include the feature of human interaction. Uncertainty during navigation is a common problem in robotic vehicle navigation and is neatly handled by including the human operator during vehicle's traversal, thus forming a closed loop between the robot and the operator. The improved framework also causes the vehicle to visit the cells previously not covered because of the presence of an obstacle and hence provides the user with much detailed information on the terrain data, such as location of trees, locations of ravines, and their spread across the cells in the field.

Robot cognition is achieved through the use of sensor map structures. These maps are very similar to the Penfield Maps or the Sensory Brain Maps, which are key concepts in human cognition. Sensor management and sensor data processing

are handled through these maps.

In this research, the two important implementations are the robot navigation in a space (i.e., a field) achieving maximum coverage with human interaction and the cognitive sensor model of the vehicle similar to the brain map. The framework incorporates the usage of signal, feature and decision fusion between the stages and allows the vehicle to handle all cases of obstacle presence and initiate user help only when the fusion stages identify an uncertain confidence level.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. James M. Conrad for his support and guidance throughout this project and motivation for my career in academia as well as Dr. Yogendra Kakad, Dr. Hamed Tabkhi, and Dr. Aidan Browne for serving on my committee.

I would like to thank my mom and dad whose prayers, support and encouragement has instilled courage and determination that has helped me come so far. Finally, and mostly importantly, I am thankful to God for giving me the opportunity, good health and mind to study and pursue the education that I have always wanted to.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Autonomous robots have been mostly used as a tool for performing tasks involving maneuverings and navigation in areas which are hazardous to humans or monotonous. Newer robots have now started to become more social and more involved in our everyday lives. Man-machine collaboration or Human-Robot Interaction (HRI) focuses on how machines can be made increasingly interactive with humans and how this communication paves way for better task performance of the robot and task reduction for human operator. For effective human-robot collaboration, it is imperative that the robot be capable of understanding and interpreting several communication mechanisms similar to the mechanisms involved in human-human interaction [1].

1.1 Motivation

Human robot interaction involves collaborative communication between the robot and the user. There are several modalities involved in the interaction, namely speech, gesture, bio-feedback sensor, and environmental influence. When a two-way interaction exists between both entities, there is a need for better decision making, heavily constrained on the robot side during the interaction process. This requires gathering the data from not just one source but from multiple sources which are streaming information. The sensor readings (hard data set) and the user-fed information (soft data set) need to be combined to help the robot make the right

decision while executing the current task [2]. Although HRI aims at making the robot highly autonomous to achieve an interaction similar to human-human interaction, the robot can still query the user for command in case the result of the data fusion seems ambiguous or during high levels of uncertainty.

Multi sensor fusion is advantageous over single sensor since the ability to identify multiple parameters is possible only by combining different sources of data. For instance, to identify a moving object, such as a airplane, requires the need to use information from infrared image sensor and pulsed radar [3] to get the range and angular direction of the airplane. A detailed description of the types of sensor fusion and architectures is described in Chapter 2. It is necessary to take into consideration the robot cognition while performing any robotic application. An overview of the types of learning that can be implemented on the vehicle is also described in the Introduction, Chapter 2.

1.2 Objective of this Work

In this research, the primary focus is on combining the data from multiple agents and improving the interaction process between the user and robot, thus increasing the efficiency of the interaction. We formulate a sensor framework that uses sensor fusion to achieve a decision before execution of a task and uses a sensor management interface through Sensory Maps to mimic the features of the Sensory Brain Maps [4, 5].

1.3 Contribution

The main contribution of this research is to achieve complete coverage planning and dynamic sensor maps. The navigation application implemented covers

maximum area of the field and invokes human involvement through a help phase. This increases the reliability of the system by reducing the possible collision of the vehicle with the obstacle. The usage of sensory maps to incorporate a dynamic sensor management and resource management is a novel approach to utilizing sensor space effectively and simulating human-brain like model for robot cognition as opposed to using neural networks.

The Complete Coverage application is implemented using Python and navigation is achieved on a terrain with surface and terrain obstacles. This applications does not use any path planning algorithm but only covers every reachable cell in the field. The application performs as expected by ensuring the robot to cover maximum area of the field while avoiding obstacles and calling the user at the right places for help. The sensor management implemented through machine learning works well by dynamically adjusting the space allocated for each sensor thereby handling the memory resources for each sensor effectively, simulating the human brain model.

For demonstration of the sensor fusion framework, the navigation application is implemented using Robot Operating System. The environment has obstacles, namely surface obstacles like blocks, trees etc. and ravines, and the framework is customized for the navigation application of the vehicle in such a terrain. The Sensor fusion process is heterogeneous and combines information from surface obstacle and ravines. Based on the type of obstacle the robot either continues moving or queries the user. Interaction with the human user will be prompted when the fusion algorithm signals an uncertain state to achieve safe

maneuvering of the vehicle in the field. The simulation of the vehicle navigation in the terrain with unknown obstacles demonstrates the functionality of the framework and its role in achieving bidirectional communication through the help phase under conditions of uncertainty. For sensor fusion, we need at least two sensors. If many sensors are used there is a possibility of physical interference between them. Thus, the choice of three LIDARs for terrain and three LIDARs for surface obstacle identification in the navigation application.

1.4 Organization

This thesis is divided into nine chapters. Chapter 2 gives an introduction to the various topics, such as Human Robot Interaction, multi sensor fusion, machine learning and the brain maps. Chapter 3 describes the various multi sensor fusion techniques and methods. Chapter 4 describes the Robot Operating System (ROS) and Gazebo. Chapter 5 describes the robot build using Gazebo and Chapter 6 presents the simulation of the sensor fusion framework for the robotic vehicle navigation on an unknown terrain. Chapter 7 describes the complete coverage planning algorithm and the modifications of the algorithm. Chapter 8 demonstrates the dynamic sensor management through the sensor maps for the modified navigation application. Chapter 9 summarizes the work completed and plans a course for future innovation and development.

CHAPTER 2: INTRODUCTION AND BACKGROUND

In this chapter, we review the basic concepts of

a) Human Robot Interaction (HRI) with examples
b) Multi sensor fusion architectures and types
c) Path planning
d) Sensor management
e) Machine learning – supervised and unsupervised classifications
f) Sensory brain maps

2.1 Human Robot Interaction

2.1.1 An Introduction

As the name suggests, HRI is combining user involvement during the execution of a robotic application. In the following section, we look into a few examples of human interactive robots that are used in several domains of our society [1].

2.1.2 HRI in the Society

Mobile robotics has been an area of interest for the past two decades.

a) Elderly and Aged

Robots have taken the role of a caretaker for the elderly people and those with disabilities. A good example can be found in a smart wheelchair, where the necessity of man machine collaboration is emphasized to

perform the task of mobility of the vehicle. A socially interactive robot called HOBBIT [8] caters the needs of elderly people at home.

Figure 2.1.2.1. HOBBIT Robot to provide elderly care [8]

b) Schools and Learning

In the area of teaching and providing assistance to teachers in schools, robots, are becoming significantly used. An example of such can be

found in the robots that were used with kindergarten children where such an interaction was found to promote geometrical thinking in children while playing with the robot [9]. It showed that children enjoyed learning and improved their geometrical thinking. In addition to assisting in teaching, the robot also provided performance statistics of the children over time. Such a positive interaction between humanoids and humans is instrumental and necessary for the development and enhancement of human robot interaction.

c) Bio-feedback Systems

Understanding and interpreting emotional signals in a human-human interaction is necessary to ensure an effective communication. Similarly, robots should be made capable of recognizing and responding to the emotional levels of the human operator. A good example of such an integration is where bio-feedback sensors are worn on the user and the user's physiological signals like anxiety levels were measured and these reading were then input to the robot [10]. In addition to normal operations, such as obstacle detection or wall following, based on the readings fed to the robot, actions can triggered such as to move closer or to start a conversation with the operator. This is a massive approach towards collaborating humans and robots communication and interaction.

d) Medicine

Medicine is another area where HRI is increasingly used. The use of

robotic assistants in hospitals has seen an increase in the recent years. An example of such, 'Gestonurse' helps the surgeon in the operating room (OR) by passing surgical instruments. The operations of the robot can be controlled via Speech and gesture [11]. Another application of HRI in medicine involves robots to treat patients affected with Autistic Disorders. In [12], techniques that were used to treat children with autistic disorders using robots are highlighted. Techniques were targeted on the areas of weakness found in these children, such as making a conversation, initiating social interactions, comprehending emotions through facial expressions, joint attention and motor skills.

e) Industry

Robots are increasing been used in several industrial applications. Human robot collaboration is important to reduce the burden of the operators in industrial assembly tasks, such as part acquisition, manipulation, and operations. LOCOBOT, the European project that involves developing customized robot co-workers in industrial assembly lines at lower costs [13] is an example of the use of HRI in industry.

f) Space Exploration

Space Exploration has seen increasing use of Robotics in the recent years. One such example, a free-flying robot is referenced in [14] that operates inside the International Space Station (ISS).The purpose of the robot was to offload the tasks for the astronaut, such as logistics management, equipment tracking, and handle an emergency situation

(such as a smoke detection).

g) Military

Robots are also used in military operations. In [15], a reference is mentioned to a robot named BEAR [Battlefield Extraction Assist Robot] (Vecna Technologies Cambridge Research Laboratory) for rescue missions. The BEAR robot can carry heavy objects and people and is capable of navigating through uneven environments. Figure. 2.1.2 [15] shows the BEAR carrying a human. The robot is designed to have a human-like form that provides the necessary flexibility and versatility while operating in environments such as hospitals, building etc.

Figure 2.1.2.2 BEAR Robot used in the military for rescue operations [15]

2.2 Multi Sensor Fusion

2.2.1 An Introduction

In order to produce reliable and more accurate information during an application, it becomes necessary to combine the data from a variety of sources such as voice, camera, touch and other sensors presently mounted on the robot. Such a combination requires the use of a fusion architecture and depending on the type of data involved (such as signal, information, decision or image) several classifications of sensor fusion exist.

2.2.2 Sensor Fusion Classes

There are various classes of Data fusion as highlighted by Dasarthy [16] as shown in Figure 2.2.2.

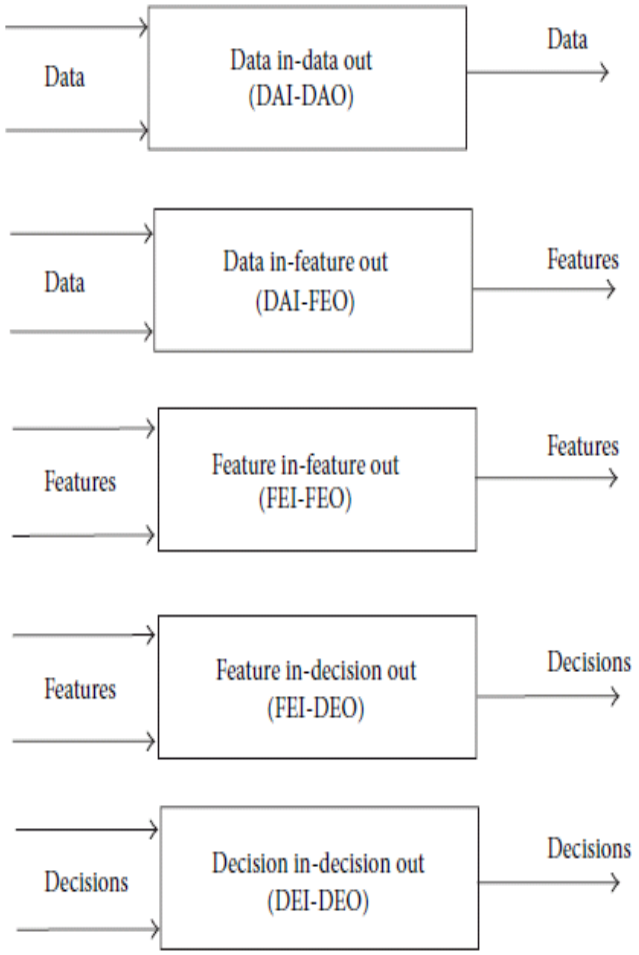

Figure 2.2.2 Dasarthy Classification [16]

2.2.3 Sensor Fusion Architectures

Sensor fusion system can also be categorized into Centralized, De-centralized, Distributed and Hierarchical [16, 17]. Figure 2.2.3 shows the various architectures [17].

In a centralized system, all the sensor data is streamed to a single node to execute the fusion process on the contrary, decentralized systems have nodes that autonomously perform the fusion process based on the local information received

by its sensors and from its neighboring nodes. Distributed architecture has a fusion node that receives the information from other source nodes. The source nodes have the ability to process the data associativity and state estimation on its sensor inputs before streaming it to the fusion node which hold the global view of the fused data.

Hierarchical architectures include nodes which perform data fusion at various levels of hierarchy, nodes are decentralized and distributed in the system.

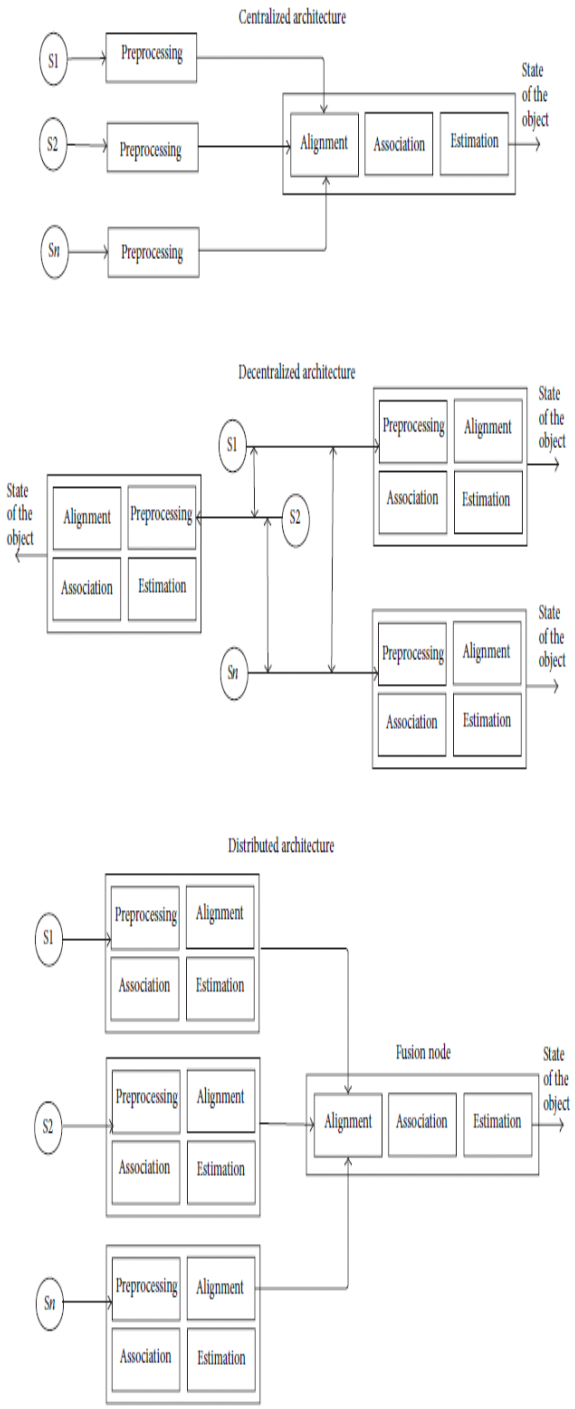

Figure 2.2.3 Types of Data Fusion Architectures [17]

2.2.4 Dempster-Shafer Theory of Evidence

A variety of Data Fusion techniques exist such as Bayesian Inference, Evidential Reasoning, Interval Calculus et al [17, 18]. While probabilistic methods are used widely there exist inherent limitations as outlined in [18], such as assigning aprori probabilities, increased complexity in presence of multiple hypothesis, and handling uncertainties while making decisions. In order to interact with humans, a fusion method that can handle uncertainties is necessary. Dempster-Shafer theory of evidence [6, 7, 18] is used to handle incomplete knowledge and uncertainty. This data fusion method is described as follows.

In this reasoning method [6, 7, 17, 18], all possible mutually exclusive events of the similar kind are listed in a set S. Each of the sensors based on its observation will assign belief mass over S. The elements of the set 2^S are the hypotheses. The sum of the mass function of all hypotheses is one. Belief function is used to express inaccurate beliefs.

The conditions on the belief function are: belief(S) =1, belief (null) =0 and 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 [18, 19, 72]:

$$m_1 \oplus m_2(\emptyset) = 0 \quad \text{(Equation 2.2.4.1)}$$

$$m_1 \oplus m_2(H) = \frac{\sum_{X \cap Y = H} m_1(X) m_2(Y)}{1 - \sum_{X \cap Y = \emptyset} m_1(X) m_2(Y)} \quad \text{(Equation 2.2.4.2)}$$

Thus, it is not necessary to have aprori probabilities and data is provided only at the time when sensor reads them. This is a significant advantage of evidential reasoning over probabilistic methods such as Bayesian inference [18, 19, 72].

2.3 Path Planning

2.3.1 Introduction

Path planning, as the name indicates is the process of formulating a route for the vehicle to navigate between any two or more locations. The overall path could be further dissected into global and local paths depending on how the path planning algorithm is executed. In the following section we discuss one such path planning technique used in vehicle navigation. Some of the path planning techniques are described in [18, 20].

2.3.2 Complete Coverage Planning

In [20], the concept of complete coverage planning was introduced. The vehicle navigates through the field column by column following a global or regular path. On encountering an obstacle, it formulates a local path to navigate around the obstacle and avoid the obstacle. Figure 2.3.2 shows the algorithm used for path planning [20]. The algorithm uses stacks to store the left and the right local paths. The tables 2.3.2.1 and 2.3.2.2 show the calculation of the local paths when the vehicle faces an obstacle [20].

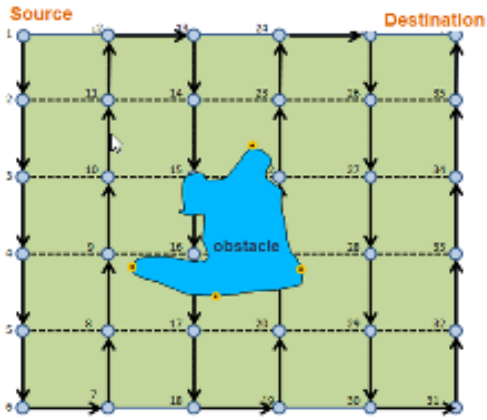

Figure 2.3.2 Obstacle

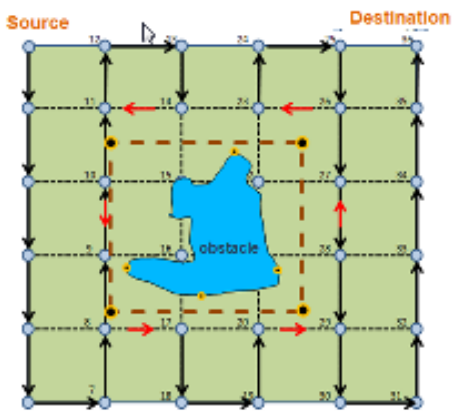

Figure.2.3.2 Local path

Table 2.3.2.1 : Case table for left local path [20]

T = Top, L = Left, B = Bottom, R = Right

Main case 1	**Ysource < Ytarget**	
Case No	**Condition**	**Sequence**
1.1	(SourceY < AltGoalY) && (SourceX == AltGoalX)	TLBR
1.2	(SourceY < AltGoalY) && (SourceX > AltGoalX)	TLBR
1.3	(SourceY < AltGoalY) && (SourceX < AltGoalX)	LBRT
1.4	(SourceY == AltGoalY) && (SourceX > AltGoalX)	RTLB
1.5	(SourceY == AltGoalY) && (SourceX < AltGoalX)	LTBR
1.6	(SourceY > AltGoalY) && (SourceX == AltGoalX)	BLRT
1.7	(SourceY > AltGoalY) && (SourceX > AltGoalX)	TRLB
1.8	(SourceY > AltGoalY) && (SourceX < AltGoalX)	BRLT

Table 2.3.2.1 : (continued)

Main case 2	**Ysource > Ytarget**	
2.1	(SourceY < AltGoalY) && (SourceX == AltGoalX)	TLRB
2.2	(SourceY < AltGoalY) && (SourceX > AltGoalX)	RBLT
2.3	(SourceY < AltGoalY) && (SourceX < AltGoalX)	TLRB
2.4	(SourceY == AltGoalY) && (SourceX > AltGoalX)	RBTL
2.5	(SourceY == AltGoalY) && (SourceX < AltGoalX)	LTBR
2.6	(SourceY > AltGoalY) && (SourceX == AltGoalX)	LTRB
2.7	(SourceY > AltGoalY) && (SourceX < AltGoalX)	LTRB
2.8	(SourceY > AltGoalY) && (SourceX > AltGoalX)	BLTR

Table 2.3.2.2 : Case table for right local path [20]

T = Top, L = Left, B = Bottom, R = Right

Main case 1	**Ysource < Ytarget**	
Case No	**Condition**	**Sequence**
1.1	(SourceY < AltGoalY) && (SourceX == AltGoalX)	TRBL
1.2	(SourceY < AltGoalY) && (SourceX > AltGoalX)	TLBR
1.3	(SourceY < AltGoalY) && (SourceX < AltGoalX)	TRBL
1.4	(SourceY == AltGoalY) && (SourceX > AltGoalX)	RTLB
1.5	(SourceY == AltGoalY) && (SourceX < AltGoalX)	LTRB
1.6	(SourceY > AltGoalY) && (SourceX == AltGoalX)	BLRT
1.7	(SourceY > AltGoalY) && (SourceX > AltGoalX)	TRLB

1.8	(SourceY > AltGoalY) && (SourceX < AltGoalX)	BLTR

Table 2.3.2.2 : (continued)

Main case 2	**Ysource > Ytarget**	
2.1	(SourceY < AltGoalY) && (SourceX == AltGoalX)	TLRB
2.2	(SourceY < AltGoalY) && (SourceX > AltGoalX)	RBLT
2.3	(SourceY < AltGoalY) && (SourceX < AltGoalX)	TLBR
2.4	(SourceY == AltGoalY) && (SourceX > AltGoalX)	RBTL
2.5	(SourceY == AltGoalY) && (SourceX < AltGoalX)	LBRT
2.6	(SourceY > AltGoalY) && (SourceX == AltGoalX)	RTLB
2.7	(SourceY > AltGoalY) && (SourceX < AltGoalX)	BRTL
2.8	(SourceY > AltGoalY) && (SourceX > AltGoalX)	BLTR

2.4 Sensor Management

2.4.1 Introduction

Sensor management is the heart of the entire multi sensor fusion system [21]. Sensor management provides a lot of freedom for the architect to formulate and deploy their designs and investigate existing architectures. In the following sub sections, we describe the several such sensor management architectures and their relevant applications.

2.4.2 Sensor Management Architectures

As described in [21, 23], there are two types of sensor management architectures. They are described as below [21]. In [22, 50], several implementations

of the sensor fusion architectures in MATLAB is presented. It is necessary to have a robust sensor management system to achieve a better performing multi sensor network.

2.4.2.1 Information-Theoretic

In this technique, the goal of the sensor manager/management is to decrease the measure of uncertainty in a sensor observation. This approach aims at maximizing the information gain of the sensor network to improve the overall performance [21].

2.4.2.2 Bayesian Decision Making

In contrast to maximizing the information, this method aims at minimizing the cost function associated with a decision. This approach utilizes apriori probabilities of the sensors and requires choosing "proper thresholds", that will be used during the decision making process [21, 23]. It is more computationally intensive than the information centric approach.

2.5 Machine Learning

2.5.1 Introduction

In order to incorporate automation into the vehicle or the robot, it becomes necessary to train the vehicle to self-adjust or self-modify its behavior when reacting to situations. Machine learning is a field in Artificial Intelligence [24] that caters to this purpose. From simple automated voice systems to the Self-driving cars, machine learning has made it possible to realize intelligent systems. Learning with experience is the key idea behind developing machine learning algorithms.

2.5.2 Types of Classifications

To achieve the desired autonomous and intelligent system for the application we as the user need to train the robot with possible sets of stimuli known as the training set. Depending on the application, the user might know the corresponding outputs and define them or might need to let the system categorize the outputs by itself. Hence, there are three possible types of classifications to the end user requirements [25, 26, 27].

a) Supervised Classifiers

The user knows the output patterns for the input sets and trains the system. When the system encounters a particular input pattern it recognizes it and fits it with an output label used during the training. Inductive learning [28] is a category of supervised learning. Naïve Bayes, Decision Trees etc. are a few examples of classifiers.

b) Unsupervised Classifiers

In Unsupervised learning, the system has to learn without a trainer i.e., the system has to deal with the environment and self-train since there are no explicit output patterns defined for the possible inputs. As mentioned in [25], the best way to train the system is through experiments, the system must possess memory and information retrieval abilities.

Moreover, this technique requires heavy interaction of the system with the user and environment. Clustering is a commonly used unsupervised classifier.

c) Reinforced Classifiers

Unlike supervised and unsupervised learning, reinforcement incorporates the

reward or punishment that the system will face during its experience. The interaction of the system with the environment produces feedback signals that are categorized as, reward and punish labels, hence the learner adapts its responses according to these labels. Some of the examples are Q learning and temporal difference learning [29].

2.6 Sensor Maps

2.6.1 Introduction

Dr. Ramachandran, in his book [4] has described the concept of brain maps. Brain maps or Penfield maps as shown in Figure 2.6.1 [4], are representation of the various sensory stimuli of the human body. As the figure shows, every sensory input has a corresponding slot or position on the map. These slots are of varying sizes and located in the sequence shown.

Figure 2.6.1 A portion of the Penfield map of the skin surface [4]

2.6.2 Dynamic Nature of the Maps

The idea that the brain is not malleable in learning and that the brain maps are fixed in size and position was a long held belief in neuro science. However, in [5], Dr. Doidge, illustrates several examples and case studies of patients who were physically affected by pain due to trauma and other medical conditions such as brain injury. These patients were able to recover from their physical problems not because of the treatments proposed and highlighted by Dr. Doidge, but through the dynamic nature of the brain maps that changed as the treatments were practiced on the patients. The plastic nature of the brain (or the sensory brain maps!), is a remarkable finding that is inspires confidence into patients and gives hope that they can lead a normal life like their friends and peers.

When a sensory input is not recognized for a long time due to a physical injury or under-utilization, the corresponding slot in the map starts to reduce, as described in the book by Dr. Ramachandran. By therapies or training, such a slot can be re-allocated back to the sensory input as described in [5, 30]. Similar to how brain exercises can help rewire the brain, learning disorders cured through training and several other methods of therapy to revitalize the brain, differentiation of the brain map can be achieved. This is the use-it-or-lose-it principle of the brain.

2.7 System Integration

The previous sections provided an introduction to each of the topics that are used in the research work. Here, we integrate them all into our framework that synthesizes these elements. The framework we formulate is very similar to the sensor fusion framework described in [21]. The main idea is to develop a robot navigation

application and utilize the machine learning techniques to implement the dynamic sensor map of the vehicle. We will deep dive into the implementation and results achieved so far in the following chapters.

CHAPTER 3: MULTI SENSOR FUSION

3.1 Introduction

Sensor fusion involves combining data from several sensors to obtain better information for perception. Humans and animals process multiple sensory data to reason and act and the same principle is applied in multi-sensor data fusion. Multi-sensor fusion combines data from different sensors into a common representation format [31, 32]. In developing robotic systems, multi-sensor fusion plays a crucial role since interaction with the environment is instrumental in successful execution of the task. Significant applications of multi-sensor fusion can be found in applications such as mobile robots [32, 33,34, 35], defense systems (such as target tracking [32, 36, 37, 38]), medicine [39, 40], transportation systems [41, 42] and industry [43, 44, 45]. The motivation for sensor fusion is discussed in section 3.2. Section 3.3 describes the various types of sensor fusion proposed in literature. The various topologies and models for sensor fusion is covered in sections 3.4 and 3.5. Sections 3.6, 3.7 provide an overview of signal and decision level fusion.

3.2 Motivation

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 [46, 47, 48]. Combining the data from a single sensor at different time intervals can also produce this effect [48].

3.3 Sensor Fusion Categories

Depending upon the sensor configuration, there are three main categories of sensor fusion: Complementary, Competitive and Co-operative [49]. These are described below as follows:

a) 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 [49, 50].

b) 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 [49, 50].

c) 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 [49, 50].

Several other types of sensor networks exist such as corroborative, concordant, redundant etc. [48]. Most of them are derived from the above mentioned

sensor fusion categories.

Dasarthy [51, 52] classified sensor fusion types depending upon the input/output characteristics. Figure 3.3 [51], shows the various sensor fusion types. Only a few combinations are allowed in Dasarthy's scheme for the inputs and outputs.

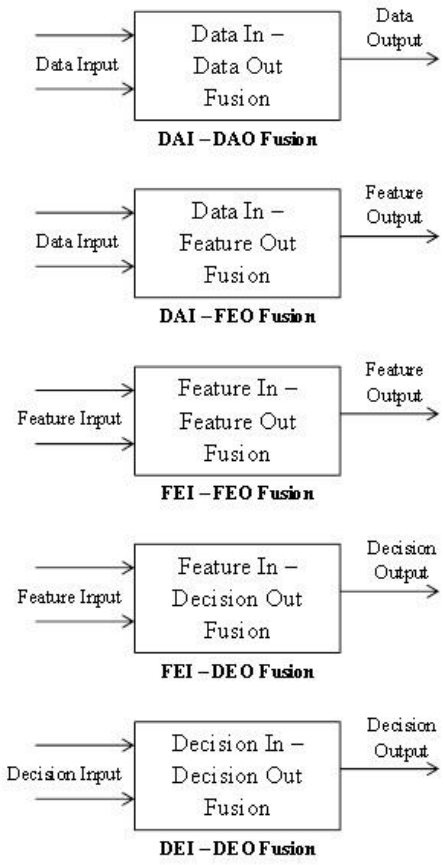

Figure 3.3 Dasarthy's classification of multi-sensor fusion [51].

3.4 Sensor Fusion Topologies

There are different topologies namely, Centralized, Decentralized and Hybrid [48, 50, 53, 54]. Each of these is described as follows:

a) Centralized Architecture

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 3. 4. 1 shows a typical centralized architecture [48, 50].

b) Decentralized Architecture

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. [50]. Figure 3.4.2 shows a typical decentralized architecture [48, 50].

c) Hierarchical Architecture

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 [50, 53].

A simple comparison between the centralized and decentralized topologies is shown below in Table 3.4 [48, 50, 17].

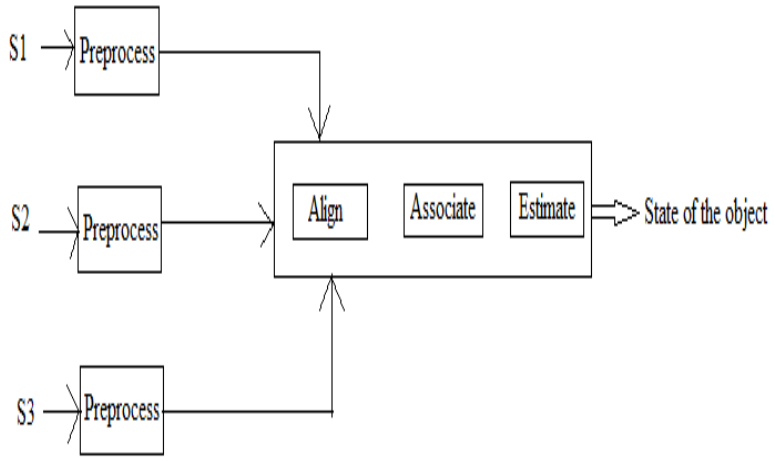

Figure 3.4.1 Centralized Topology [17, 53]

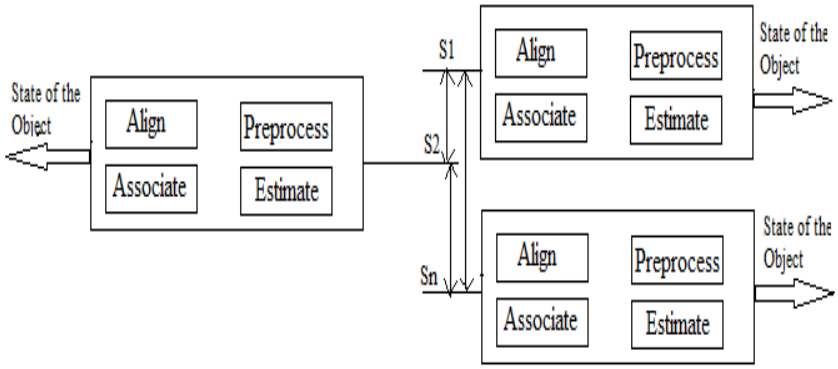

Figure 3.4.2 Decentralized Topology [17, 53].

Table 3.4: Centralized and Decentralized topologies [48, 50, 17]

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

3.5 Multi-Sensor Fusion Models

The application that uses the sensor fusion plays a vital role in determining the type of architecture. Hence there is no specific model or architecture that is definitive for all applications [55, 56, 57].

In this section, the two most widely used architectures namely, the JDL Fusion architecture and the Waterfall Fusion Process Model are discussed.

a) JDL Fusion Architecture

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 [50].

The model uses five levels for data processing and a database. These

components can communicate through a bus interface [50, 54, 56]. The JDL model is shown in Figure 3.5.1 [54, 56]. These levels could be executed sequentially or concurrently during the application.

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 [54, 56]. 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 considered to be composed of kinematic and identity fusion [50]. 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 [50, 54]. 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 [54].

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 [50, 54, 56]. The Database Management System helps monitor, update, add and provide information to the fusion process [50, 54, 56].

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. It is abstract and interpretation could be difficult [54, 56].

Table 3.5.1 [54] highlights the summary of various components used in JDL model.

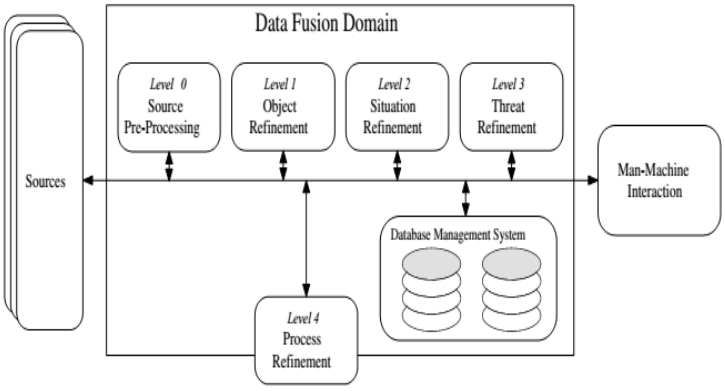

Figure 3.5.1 JDL Fusion Model [54, 56].

Table 3.5.1: Summary of JDL process components [54]

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.

b) Waterfall Fusion Process Model

The Waterfall fusion process model (WFFM) deals with the low level processing of data and is shown in Figure 3.5.2 [54, 58]. 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 [54, 56, 58] and the comparison is shown in Table 3.5.2.

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 [50] 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 3.5.3 [50]. Several other fusion models exist such as the Omnibus model [59], Boyd or OODA model [60], LAAS Architecture [61].

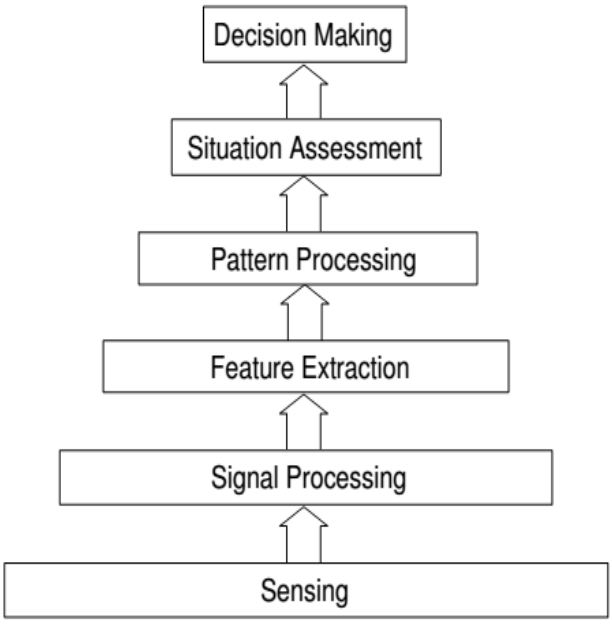

Figure 3.5.2 Waterfall Fusion Process Model [58]

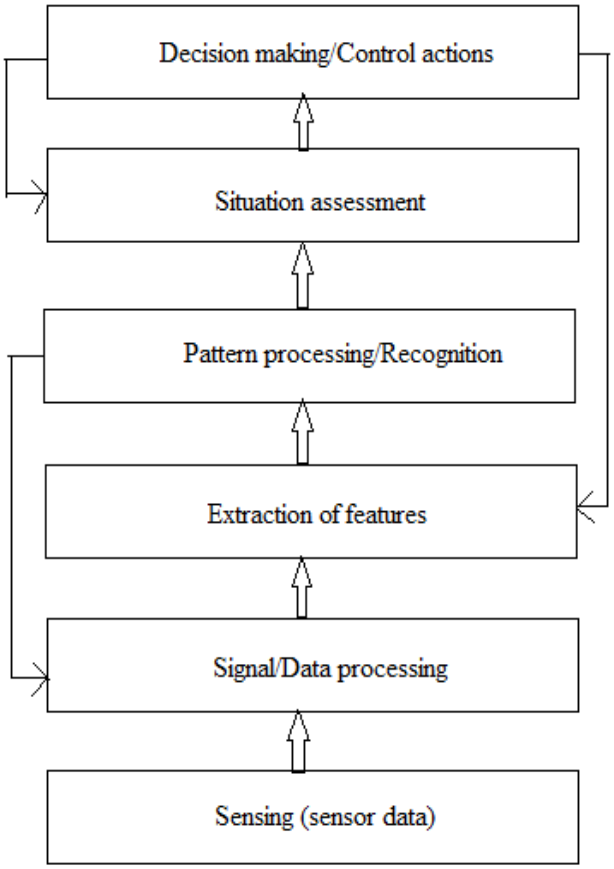

Figure 3.5.3 Modified Waterfall Fusion Model [50].

Table 3.5.2: JDL and Waterfall fusion models [54, 56, 58]

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

3.6 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:

Obtain a higher quality version of the input signals i.e. higher signal to noise ratio [62]. Sensor measurements from several sensors which have same physical properties are combined to determine the parameter being measured, more accurately [48]. 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.

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 [54].

For sensor data to undergo signal level fusion, it is essential to condition the signals in the signal preprocessing phase. The signals have to be in a common representation format [48]. The stages involved in this process, as shown in Figure 3.6.1, include but not limited to: Signal alignment, normalization and scaling [48].

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.

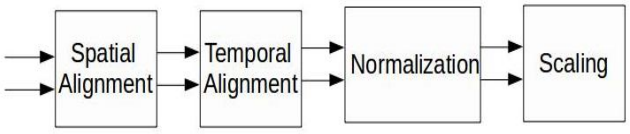

Figure 3.6.1 Common representation format functions [48].

The following are some of the commonly used signal fusion methodologies:

a) 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. [62]

$$x_{fused} = \sum_{i=0}^{n} w_i x_i \qquad (1)$$

where, w_i = f(variance)

b) 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:

$$x(k) = F\,x(k-1) + B\,u + G\,w$$
$$z(k) = H\,x(k) + v$$

where, x: state vector, F: state transition matrix, B: Input transition matrix, u: Input vector, G: Process noise transition matrix, w: process noise vector, H: 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:

$$\hat{x}_k = A\,\hat{x}_{k-1} + B\,u_k \quad (2)$$

$$P_k = A\,P_{k-1}\,A^T \quad (3)$$

Update phase:

$$K_k = P_k C^T (C P_k C^T + R)^{-1} \quad (4)$$

$$\hat{x}_k = \hat{x}_k + K_k(z_k - C\,\hat{x}_k) \quad (5)$$

$$P_k = (1 - K_k C)P_k \quad (6)$$

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 of them 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. [62]

c) 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 3.6.2 [63]. 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 [63].

$$\hat{x}_{k|k} = \hat{x}^1_{k|k} + \left[P^1_{k|k} - P^{12}_{k|k}\right]\left[P^1_{k|k} + P^2_{k|k} + P^{12}_{k|k} + P^{21}_{k|k}\right]^{-1} \left(\hat{x}^2_{k|k} - \hat{x}^1_{k|k}\right) \tag{7}$$

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

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

$$P^{12}{}_{k|k} = \left(1 - K^1_k H^1_k\right) F_{k-1} P^{12}_{k-1|k-1} F^T_{k-1} \left(1 - K^2_k H^2_k\right)^T + \left(1 - K^1_k H^1_k\right) G_{k-1} Q_{k-1} G^T_{k-1} \left(1 - K^2_k H^2_k\right)^T \tag{8}$$

This configuration can be extended for multiple sensors. A modified track-to-track fusion and three fusion algorithm are explained in detail in [63].

There are other ways to define the track fusion algorithm such as taking confidence weighted averaging of the tracks based on variance [63].

d) 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. [64]

Unlike Kalman filters, neural networks offer non-linear transfer functions and parallel processing capabilities. This can help in performing image fusion. Figure 3.6.3 shows a basic structure of three layer neural network with nonlinear mapping.

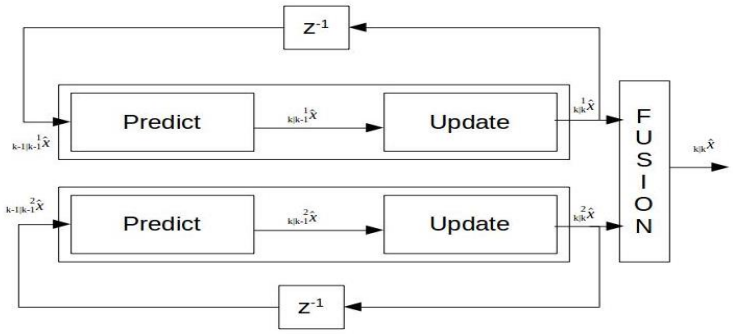

Figure 3.6.2 Track to track fusion architecture [63].

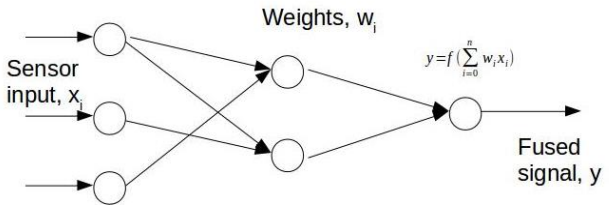

Figure 3.6.3 Neural network structure for sensor fusion [64].

The fused output is a combination of input signal and corresponding weights calculated by the equation [64]:

$$y = \sum_{i=0}^{n} w_i x_i \qquad (9)$$

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 aforementioned signal fusion methods.

The probabilistic approach for sensor fusion includes the use of joint probability distributions and Gaussian distributions [68]. Other fusion methods include Bayesian, least-squares for feature extraction [69] and some statistical approaches. [48, 62, 70]. In [65, 66, 67] the authors explain various approaches for modeling sensor fusion architecture using neural networks.

3.7 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 [71, 72]. Identity and Knowledge based methods form the two categories of decision fusion [50, 72]. Table 3.7 [50, 72] lists few of the decision fusion methods or AI techniques for each category.

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 [71, 72, 75, 76]. We describe in detail the D-S theory in the following sub-section followed by a comparison with Bayesian inference which is another widely used decision fusion technique.

a) Dempster-Shafer Theory of evidence

D-S theory is a generalization of the probability theory [6, 7, 71, 75]. In this method, a frame of discernment Ω is defined which is set of elementary hypotheses:

$$\Omega=\{a_i\},i=1, ..., n \qquad (10)$$

Table 3.7: Decision Fusion Models [50, 72]

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

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 [6, 7]:

$$m_1 \oplus m_2(\emptyset) = 0 \tag{11}$$

$$m_1 \oplus m_2(H) = \frac{\sum_{X \cap Y = H} m_1(X) m_2(Y)}{1 - \sum_{X \cap Y = \emptyset} m_1(X) m_2(Y)} \tag{12}$$

b) Dempster-Shafer and Bayesian fusion comparison

Although both these methods are widely used in inference engines there are few differences between them [72, 76]. The main difference being the concept of support and plausibility to define uncertainty limits in Dempster-Shafer [72, 6, 7] 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 sets of sets [72]. 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 prori probabilities and data is provided only at the time when sensor reads them [72, 76] 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 [1].

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 [71, 72] that work well with discrete sets.

In the current research work the framework formulated is similar to that described in [73, 74]. The framework uses sensor fusion, machine learning and the robot operating system to navigate an unknown terrain with surface and terrain obstacles.

3.8 Conclusion

In this chapter a brief overview of the various concepts of multi-sensor fusion was presented. The types of sensor fusion, the sensor fusion topologies and architectures were reviewed. A survey of sensor fusion types can be found in [77]. Signal level and Decision level fusion was also covered highlighting the methods used to achieve each of them.

CHAPTER 4: ROBOT OPERATING SYSTEM

4.1 Introduction

In this chapter we describe the basics of the Robot Operating System (ROS). It is a robotic platform for running simulations. ROS can be defined as [78, 79] a open source meta operating system that provides features similar to an operating system. These features include physical layer abstraction, low level device control, communication between processes and package management. There is also provision to obtain, build, write and run code across multiple computers [78] through different tools and libraries. ROS works together with the operating system currently installed and doesn't replace it.

4.2 Advantages of ROS

Using ROS has some significant advantages that are summarized below [78].

a) Distributed Computation

Several robotic systems have multiple processes that are executed on different computers. These processes could control one or more sub-systems of the robot such as sensors or actuators. Even a single robot could have its features written as a set of independent modules or processes that is easy to control and configure. This is called complexity via composition. The modules coordinate together in-order to

achieve the goal. Also, ROS helps multiple computers to co-ordinate and work on a shared task and helps in the communication between these computers. Most importantly, ROS provides mechanism to include human users to control and send commands to the robot from any electronic device such as laptop, mobile device etc. All of these are dependent on the communication between the processes and ROS helps to achieve them through message passing and through services [78].

b) Software Reusability

One of the key elements in software engineering is the ability to develop software that is adaptable to changing business or project requirements. Robotic applications require the use of path planning algorithms, navigation, motion planning and several others. ROS helps is modifying and applying these algorithms through its libraries depending upon the context and the application that is sought for. This eliminates the need to re-implement the algorithms for every new context or application that is being developed [78].

Also, since ROS uses message passing as its means of communication. It integrates easily with the latest hardware and implementations of several frequently used applications such as navigation stacks, motion control etc, are readily available [78, 79].

c) Rapid Testing

While software development can be time consuming it is often more important to test the software modules being developed. In robotics software development testing is often time consuming [78] because of the non-availability of physical robots. The testing can be still time consuming in the presence of physical

robots because of their slowness and other physical level difficulties.

With ROS it is possible to record and playback the sensor data and other similar messages through the use of rosbag [78, 79]. This helps to test the different ways by which the sensor data is processed by the system. ROS also separates the physical level abstraction and the decision making (high level processing) components of the robotic system. By doing this the low level processing or hardware can be substituted with a simulator and testing can be performed on the higher levels of the system [78].

Although ROS offers such significant advantages it is not the only robotics platform. However, in our research work ROS was preferred because of the vast amounts of information available through the robotics community [79, 80, 81] which helps is faster development and debugging of the project.

ROS is not a programming language but has its programs written using C++. There is also support for other libraries that are written using the languages such as python, lisp etc. ROS includes server, command-line tools, graphical interface and a build system alongside several client libraries [78, 79]. ROS is not an integrated development environment like eclipse etc.

4.3 ROS Distributions and Installation

ROS versions are named as distributions. Some of the distributions are hydro, groovy, electric, fuerte, jade etc. In this research work we have used ROS indigo and its corresponding libraries. The various steps to install and configure ROS indigo [78, 79, 80].

4.4 Gazebo Simulator

Gazebo is a robotics simulator that aids in testing of robotic algorithms, designing robots, testing and training of AI based robotic systems [82]. Gazebo is a highly robust physics engine that offers capabilities of high quality graphics and programmable interfaces [82]. It was developed by the Open Source Robotics Foundation (OSRF) and is supported by Windows, Linux and MacOS. It provides support for various families of robots such as unmanned aerial vehicles (UAV), ground vehicles, underwater robots, humanoids, robotic arms, robotic hands and human avatars [80, 82]. It also provides support for various sensors such as odometry, IMU, collision, GPS, monocular and stereo cameras, depth cameras, 2D and 3D laser scanners [80, 82].

4.4.1 Gazebo Components

In this section the different components of gazebo are defined from [82].

a) World File

   This file contains the details of the various elements used in the simulation such as the sensors, robots and the other static objects. Format of this file follows the Simulation Description Format (SDF). The world file has the extension ".world".

b) Model File

   This file helps to facilitate model reusability and simplifies the world file described above. It uses the same format as the world file. Gazebo features an online database of models and any model can be inserted from this database and downloaded at runtime.

c) Environment Variables

There are several environmental variables that need to be set up to locate necessary files and to establish communication between the server and client. The list of these variables are:

GAZEBO_MODEL_PATH: directories where Gazebo searches for models

GAZEBO_RESOURCE_PATH: directories where Gazebo searches for world and media files.

GAZEBO_MASTER_URI: Gazebo master URI. This is the IP and port of the server.

GAZEBO_PLUGIN_PATH: directories where Gazebo searches at runtime for shared libraries.

GAZEBO_MODEL_DATABASE_URI: Online model database URI.

d) Gazebo Server

The server parses the ".world" file and uses the physics and sensor engine to simulate the physical world.

e) Graphical Client

The client connects to the server and visualizes the elements. The simulation can be modified using the graphical client.

f) Plugins

Plugins can be used to interface with gazebo. They can be given through command-line or through the ".world" file. The server or client also have the capability to load and run the plugins.

4.4.2 Robot Model using Gazebo

In this section we describe a robot model built using gazebo also known as SDF Model Object [82]. The SDF model could be a simple shape such as rectangle, square, circle etc or a complex robot. The various components of an SDF model are as follows:

a) Links

The physical properties of one body of the model is described using links. It could be a wheel or even a join chain link. For better performance and stability it is necessary to keep the links as less as possible. Links could also contain collision and visual elements which are described as follows. Collision elements encapsulates a geometric structure to detect any collisions. The visualization of the parts of the link are done through a visual element. A link may contain many collision elements and 0 or more visual elements. The mass and rotational inertia matrix are dynamic properties of the link and are described using the inertial element. The sensor element collects data from the environment and there could be 0 or more sensor elements in a link.

b) Joints

Joints are used to connect links.

c) Plugins

This can be used to control a model and is a shared library.

4.4.3 Gazebo and ROS Integration

To integrate stand-alone Gazebo with ROS a set of wrappers called

"gazebo_ros_pkgs" provide wrappers around the gazebo. Figure 4.4.3 gives an overview of the gazebo_ros_pkgs interface [82].

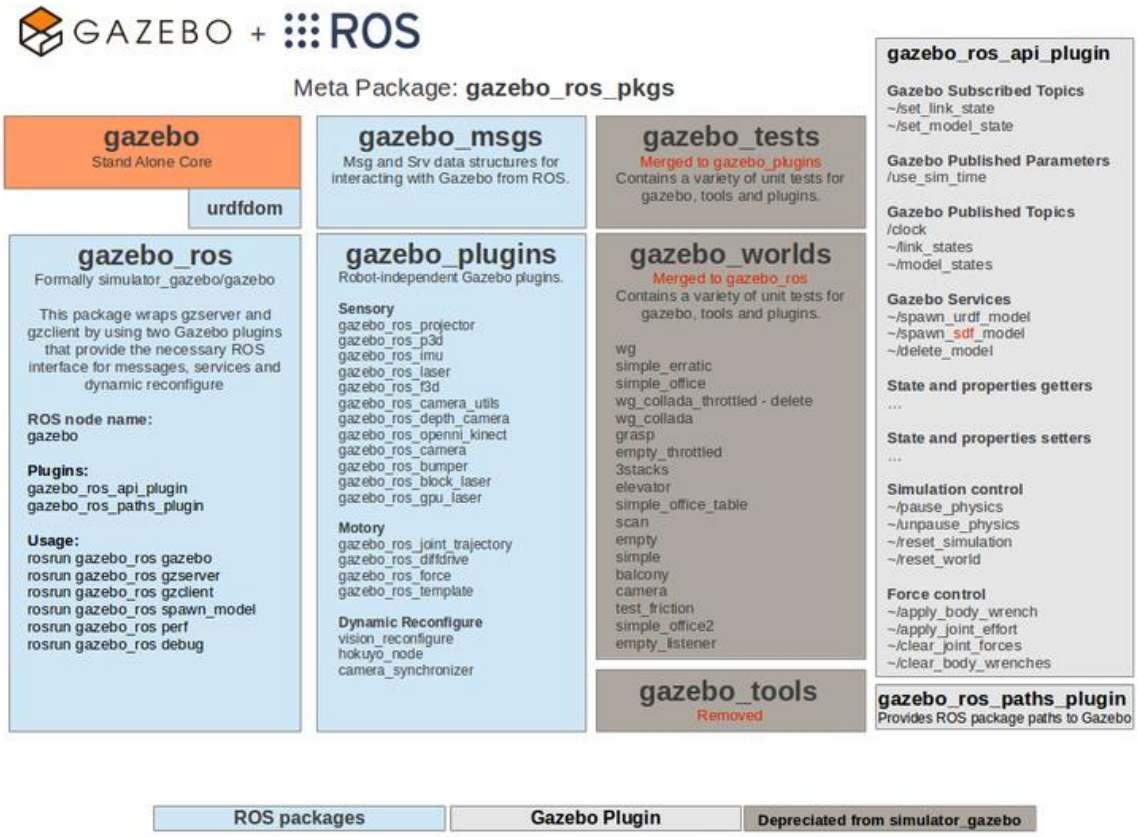

Figure 4.4.3 Gazebo ROS package interface [82].

CHAPTER 5: ROBOT MODEL USING GAZEBO

5.1 Introduction

We describe in this chapter the robot build using Gazebo. The basics of the gazebo simulator was described in the previous chapter. The version of gazebo used is 7.1 and the version of Ubuntu used is 16.04 with ROS Kinetic.

5.2 Robot Model

In the research work we are using a robot similar to turtlebot [80, 83]. In order to represent the turtlebot and use extra sensors it is necessary to create a new model file for the simulation and the exiting model files cannot be used directly because of the required modifications.

The model file follows the Simulation Description Format (SDF). There are three sensors mounted horizontally to acquire readings of obstacle in front of the vehicle and there are three sensors mounted at 45 degree to identify and detect any ravine or slopes in the terrain.

5.2.1 Robot Model SDF

The figure 5.2.1 shows the snippet from the robot model specifying the parameters for the sensors. The code is repeated for every sensor in the system making it occur 6 times in the robot model SDF file [82].

```
652
653 <!-- HORIZONTAL S1 -->
654     <link name="hokuyo_link1">
655       <pose>0.095 -0.05 0.408457 0 0 -0.0174533</pose>
656       <inertial>
657         <inertia>
658           <ixx>1e-6</ixx>
659           <ixy>0</ixy>
660           <ixz>0</ixz>
661           <iyy>1e-6</iyy>
662           <iyz>0</iyz>
663           <izz>1e-6</izz>
664         </inertia>
665         <mass>1e-5</mass>
666       </inertial>
667       <collision name="collision">
668         <pose>0 0 0.5 0 0 0</pose>
669         <geometry>
670           <box>
671             <size>0.1 0.1 0.1</size>
672           </box>
673         </geometry>
674       </collision>
675       <visual name="visual">
676         <geometry>
677           <mesh>
678             <uri>model://hokuyo/meshes/hokuyo.dae</uri>
679           </mesh>
680         </geometry>
681       </visual>
682       <sensor name="laser" type="ray">
683         <pose>0.01 0 0.0175 0 -0 0</pose>
684         <ray>
685           <scan>
686             <horizontal>
687               <samples>5</samples>
688               <resolution>1</resolution>
689               <min_angle>-0.523599</min_angle>
690               <max_angle>0.523599</max_angle>
691             </horizontal>
692           </scan>
693           <range>
694             <min>0.08</min>
695             <max>10</max>
696             <resolution>0.01</resolution>
697           </range>
698         </ray>
699         <plugin name="laser" filename="libgazebo_ros_laser.so"> <!-- libRayPlugin.so -->
700       <topicName>/scan1</topicName>
701       </plugin>
702           <always_on>1</always_on>
703           <update_rate>4</update_rate>
704           <visualize>true</visualize>
705       </sensor>
706     </link>
707
708      <joint name="hokuyo_joint1" type="revolute">
709       <parent>base</parent>
710       <child>hokuyo_link1</child>
711       <axis>
712         <xyz>0 1 0</xyz>
713         <limit>
714           <upper>0</upper>
715           <lower>0</lower>
716         </limit>
717       </axis>
718     </joint>
719 <!-- HORIZONTAL S1 END-->
```

Figure 5.2.1 Robot Model SDF [82]

5.2.2 Robot Model SDF parameters for sensors

From figure 5.2.1 we can see there are several parameters that need to be configured while adding a sensor. Below we describe some of the important parameters [82].

a) Pose: This defines the spatial location of the sensor. From the figure the (x,y,z) for the sensor is (0.095 -0.05 0.408457).

b) Inertia: This sets the values to balance the effects due to gravity.

c) Sensor name: Identifies the nature of the sensor. The sensor here is of kind laser.

d) Scan: This defines the number of samples needed per sweep, the resolution, the minimum and maximum angle of the laser. Here we set the number of samples to be recorded to 5 and the angle of the laser is from -30 to +30 degrees.

e) Topic name: This specifies the name that will be used in the ROS to publish the scan messages.

f) Plugin filename: This specifies the file that establishes the connection between ROS and gazebo. Gazebo publishes the messages to be accessed by ROS through this file.

5.3 Robot View

Figures 5.3.1 to 5.3.3 show the robot that is built from different angles.

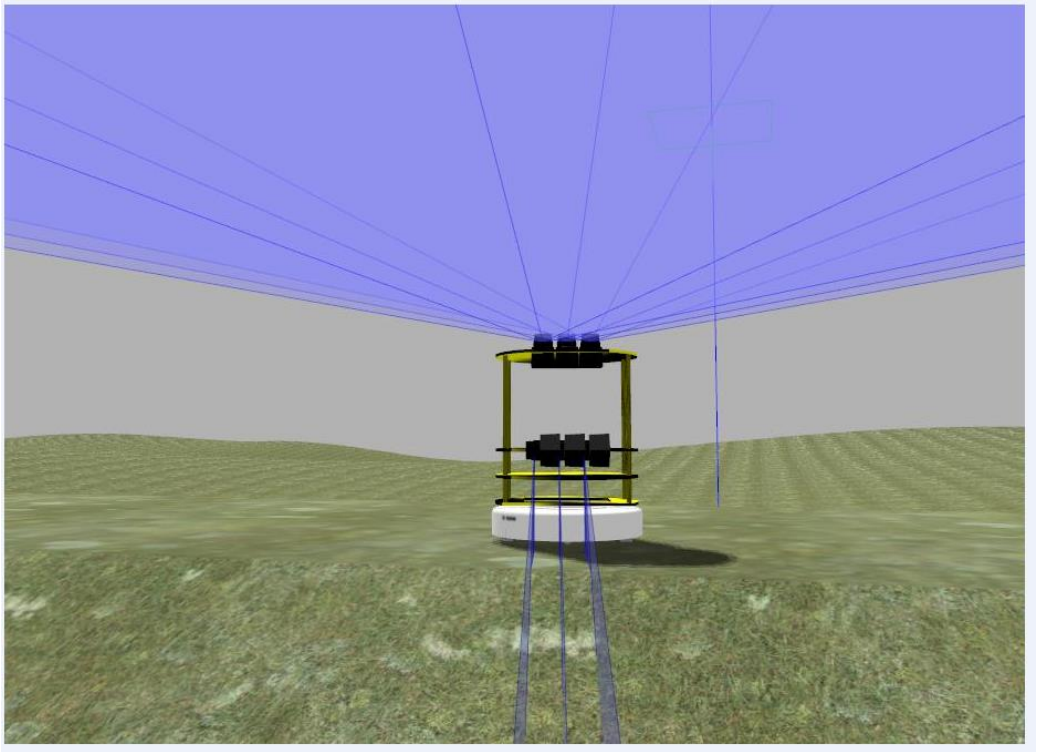

Figure 5.3.1 Front view of the robot

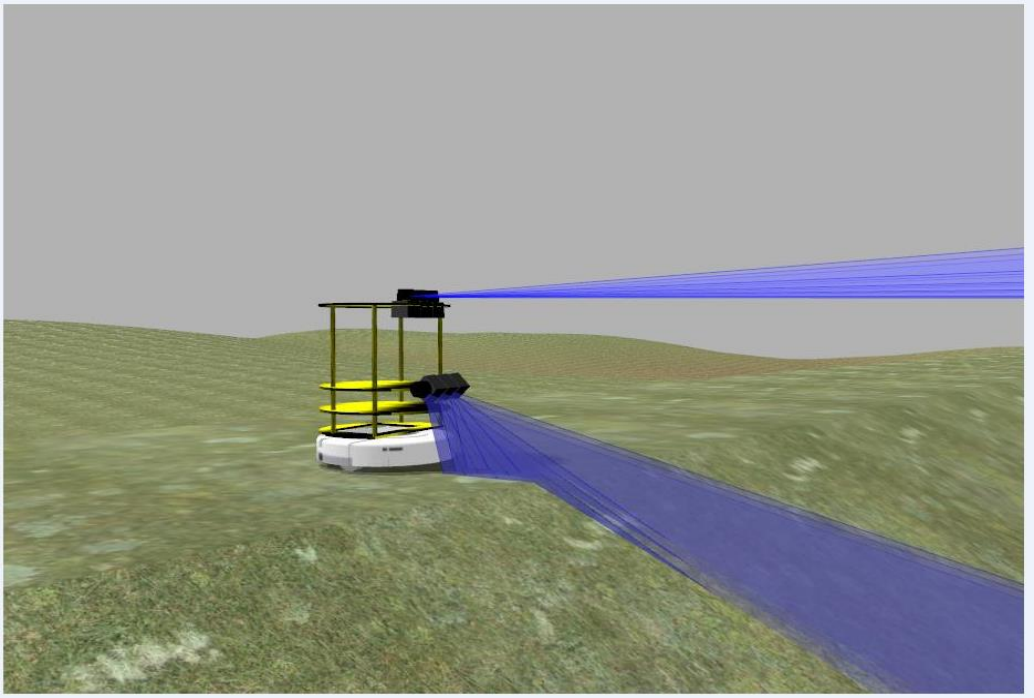

Figure 5.3.2 Side view of the robot

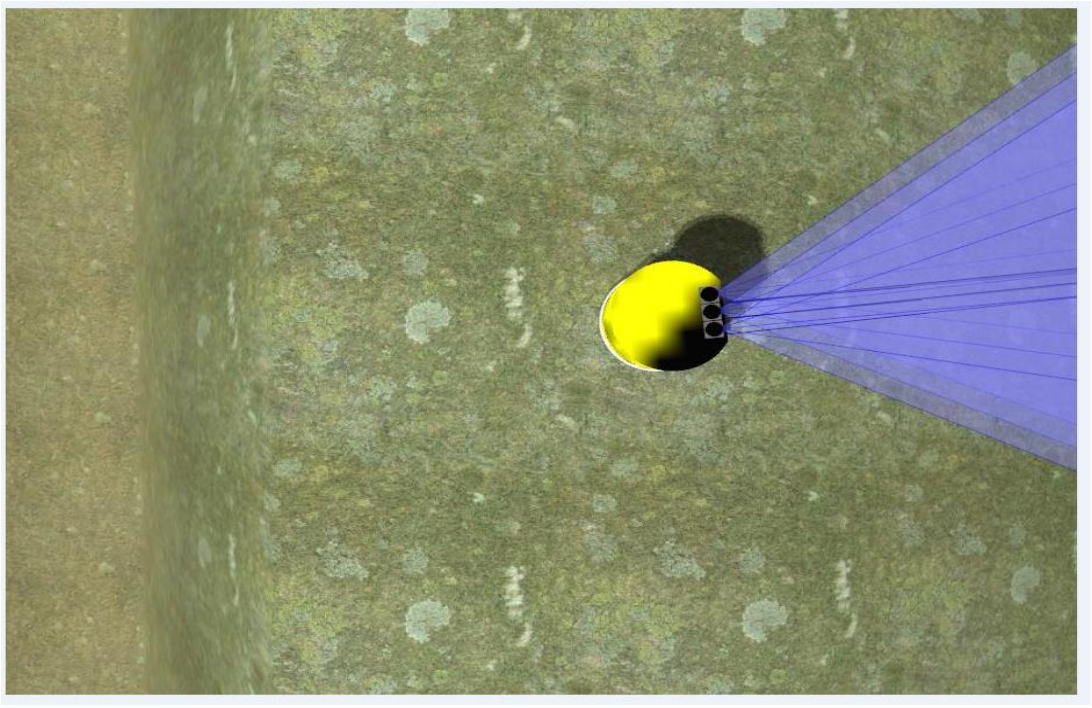

Figure 5.3.3 Top view of the robot

CHAPTER 6: ROS AND GAZEBO SIMULATION

6.1 Introduction

In this chapter we describe the simulation of the sensor fusion framework [73] using ROS and Gazebo. ROS and Gazebo were described in detail in Chapter 4. The robot model that we use for the simulation was covered in detail in Chapter 5. The sensor fusion framework is the basis of this thesis. In a completely human-centered approach of robot control; the operator serves as the master guiding the robot at each step of the process. Here we consider the human-operator as another input that is providing information and not as a controller except when the robot is unable to make a decision on an action [73, 85]. Human-robot interaction (HRI) is the key theme of this framework and the process by which the framework achieves it depends entirely on the type of application the framework is intended for.

In the present research the application is the navigation of a robotic vehicle using LIDAR sensors [84] in an unknown terrain. The robot is built using Gazebo and is coupled with ROS to achieve the physical layer abstractions and high level processing. The following section describes the various blocks/stages of the framework in detail similar to what was described in [73, 85].

6.2 Block Diagram of the Framework

The block diagram of the framework is shown in Figure 6.2.

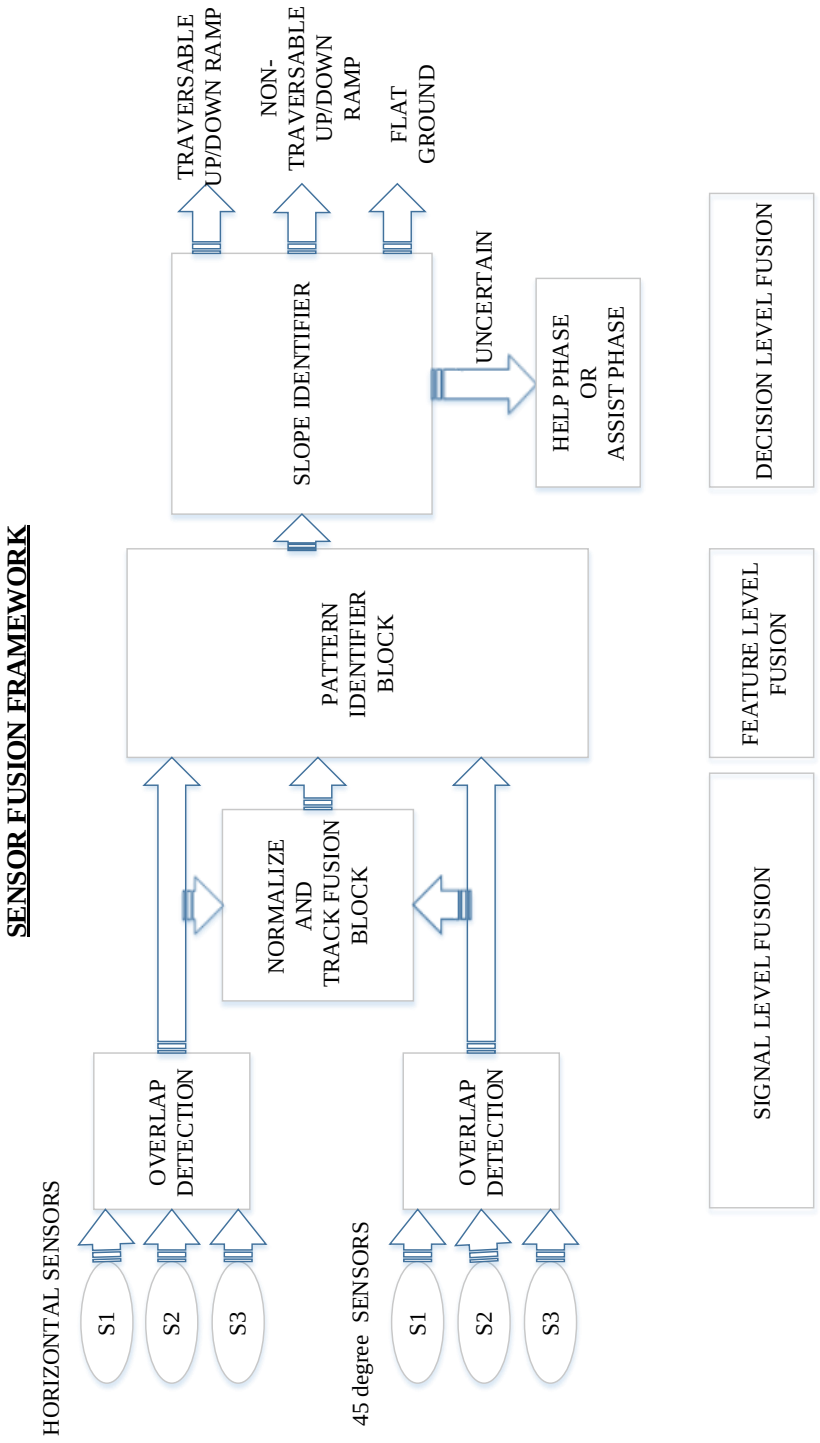

Figure 6.2 The block diagram of the entire system.

6.3 Input Unit

The input unit consists of the sensory data. The sensors used for the application are LIDARs [84]. There are three LIDARs that are mounted horizontally to gather information on obstacle in front of the vehicle. There are three more LIDARs that are mounted at 45 degree angle to acquire the data for any ravine or depth of the ground.

6.4 Overlap Detection

The purpose of pre-processing the data from the LIDARs is to acquire more specific information on the location of the obstacle in the cell in front of the vehicle. The sensors used for the application are LIDARs (lasers). There are three LIDARs that are mounted horizontally to gather information on obstacle in front of the vehicle. There are three more LIDARs that are mounted at 45 degree angle to acquire the data for any ravine or depth of the ground. Figure 6.4.1 shows the overlap detection happening in the current application.

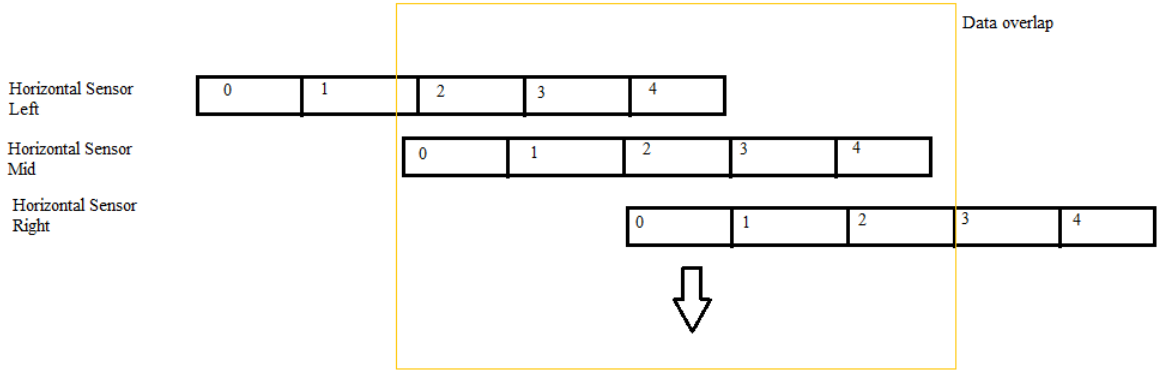

Figure 6.4.1 Overlap between the three horizontal sensors.

As shown in Figure 6.4.1, each sensor stores five values of readings in the five elements of the array. The overlap detection helps in identifying the more precise location of the obstacle in the cell. Also, when there are multiple sensors measuring the same attribute it is necessary to handle reliability between the sensors.

The first overlap happens between left sensor's 2^{nd} and middle sensor's 0^{th} reading in the array. When both these sensors detect obstacle their confidence value is incremented by 1. Similarly for the left sensor's 3^{rd} and middle sensor's 1^{st} values. When the overlap happens between all the three sensors i.e., left sensor 4^{th}, mid sensor's 2^{nd} and right senor's 0^{th} values then each of the sensors confidence is incremented by 0.5 except the mid sensor which is incremented by 1. The reason left and right sensor confidence is only incremented by 0.5 is to distribute evenly the confidence between all the sensors. Similarly for the confidence increments between the mid sensor and the right most sensor values.

At the end of the overlap detection we have the sensor whose value is the most reliable. Similar to the horizontal sensor the overlap detection is performed with the sensors that are mounted at 45 degree angle. The sensor with the highest confidence value will serve as the choice for the track-to-track fusion algorithm that will combine the data between the horizontal sensor and the 45 degree tilted sensor. The output of the overlap detector is the value of the sensor that is considered most reliable among the three sensors. Using such an overlap detection, the system can be extended to incorporate more sensors and identify the most reliable among them based on their current readings and reduces the complexity of using averaging or voting methods and improves the fault tolerance of the system. Figure 6.4.2 shows

the sensor values overlap through “rays”.

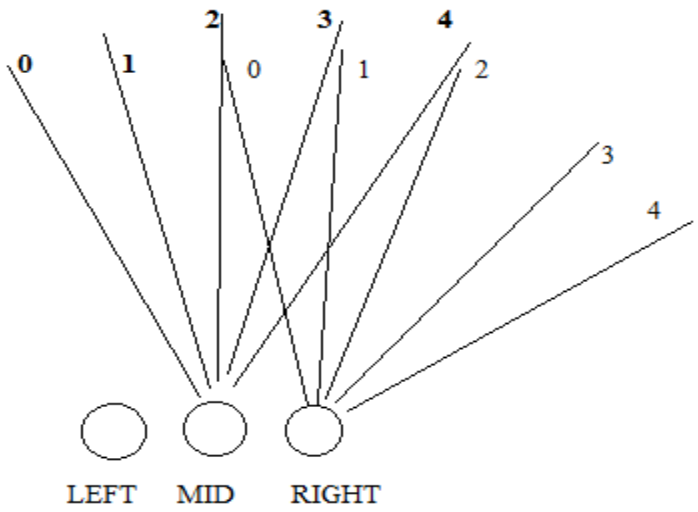

Figure 6.4.2 Overlap between the three horizontal sensors illustrated through “rays” from the sensors.

6.5 Track-to-Track Fusion block

The sensor with the best confidence, i.e., with highest value is chosen. One sensor value from the horizontal sensor group and one sensor value from the 45 degree tilted sensor group is input into the track to track fusion block.

Track to track fusion was described in Chapter 3. 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 3.6.2 of Chapter 3 and equations 7, 8 [63]. 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 [63]. The superscript 1, 2 refer to the values of the sensor 1 (horizontal) and sensor 2 (45 degree tilted sensor) respectively. These two sensors are measuring different attributes of the environment i.e., obstacle in front (could be a block, tree etc.) and ravine (or a ramp) on the terrain. The fusion takes place using the equation for $P^{12}_{k|k}$

$$\hat{x}_{k|k} = \hat{x}^1_{k|k} + \left[P^1_{k|k} - P^{12}_{k|k}\right]\left[P^1_{k|k} + P^2_{k|k} + P^{12}_{k|k} + P^{21}_{k|k}\right]^{-1} \left(\hat{x}^2_{k|k} - \hat{x}^1_{k|k}\right) \tag{7}$$

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

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

$$P^{12}{}_{k|k} = \left(1 - K^1_k H^1_k\right) F_{k-1} P^{12}_{k\text{-}1|k\text{-}1} F^T_{k-1} \left(1 - K^2_k H^2_k\right)^T + \left(1 - K^1_k H^1_k\right) G_{k-1} Q_{k-1} G^T_{k-1} \left(1 - K^2_k H^2_k\right)^T \tag{8}$$

6.6 Pattern Identifier block

The sensor with the best confidence i.e., with highest value is chosen. One sensor value from the horizontal sensor group and one sensor value from the 45 degree tilted sensor group is input into the track to track fusion block. Based on the values given by the track to track fusion block we have two scenarios, one for obstacle present and other for obstacle not present. When the track fusion result is between 0 to 1 it is classified as no obstacle and we set a variable (or flag) to 0. When the value returned by the track fusion block is outside this range we consider that as

a sign of obstacle presence and we set the variable/flag to 1. Now, in order to improve the overall confidence, we use more evidence to identify the type of feature the obstacle could be from among tree/block, ramp or both these obstacle features.

A truth table is generated using the track to track fusion result with the data provided by the physical sensors (horizontal sensor and 45 degree tilted sensor) observing the two attributes of the environment. The purpose of such an approach is to increase the reliability of the system by using more evidence present and to make the system scalable. Truth table can be altered easily by adding more columns to include features at more granular level. Also, when certain features don't contribute to the decision the respective columns could be removed or considered as a don't-care similar to that found in digital logic systems. The rule based truth table is shown in Table 6.6. Sensor 1 (s1) is the horizontal sensor and sensor 2 (s2) is the tilted sensor. TF (t) is the track fusion block output variable. Boolean equation for uncertainty:

uncertain = (s2 AND NOT(t)) OR (s1 AND NOT(t)) OR (s1 AND NOT(s2) AND t)

Table 6.6: Pattern Identifier

Sensor 1	Sensor 2	TF	Feature
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

6.7 Ramp Identifier

The type of feature is detected by the pattern identifier block. If there is a terrain issue, there are two possibilities. One is a ramp upwards/uphill and the other is a ramp downwards/downhill. For each of these ramps there exists two more possibilities. A traversable ramp up and non-traversable ramp up. There might be a traversable ramp down and a non-traversable ramp down. Table 6.7 will describe these more neatly.

Table 6.7: Possible Ramp Scenarios in the terrain

RAMP Type	Angle	Navigability
Up	1<Angle<=4	Yes
Up	Angle>4	No
Down	-4<Angle<-1	Yes
Down	Angle<=-4	No

The slope of the ramp determines whether the ramp can be traversed or not. The angle can be set by the user depending on the environment and the configuration of the robot.

6.8 Dempster-Shafer (D-S) for Uncertainty

The D-S decision method [6, 7] was covered in detail under Chapter 2 and 3. By using the properties of the sensors we can create a mass table with values as shown in Table 6.8. The rule of combination is used to calculate the uncertainty percentage. Although, in this research Dempster-Shafer is used for only calculating the uncertainty percentage it is possible to extend the decision block to include more

features and calculate their confidence percentages. In Table 6.8 the notations mass 1 and mass 2 refer to the mass values of sensors 1 and 2 respectively. This concept is similar to the method used in [86]. If D is the effective distance for the sensor.

Table 6.9: Mass Values [86]

Mass Value (m)	D = 0	0.2 < D <=5	D > 5
m(obstacle present)	0.3	0.8	0.2
m(no obstacle)	0.3	0.2	0.7
m(obstacle present, no obstacle)	0.6	0	0.1

6.9 Human Input under Uncertainty

Help phase is initiated when the vehicle is unable to identify the outcome of the situation and queries the user for assistance. In this framework, at the pattern identification stage we can flag uncertain situation and calculate the degree of uncertainty using D-S rule of combination. Here we achieve coordination between the user and the robot by communicating through help commands. To minimize the communication delay the framework uses single characters to trigger commands. Entering ‘r’ sends the command to turn right, ‘f’ commands the vehicle to move forward. A complete list of commands is shown in Table 6.9. Such a collaborative control is similar to what was described in [73, 85].

Table 6.9: Possible Commands for the Robot

Command	Code
Forward	f
Left	l
Right	r

Although the help phase alleviates the problem of the vehicle getting stuck during its navigation it is imperative from the user side to be equally responsible as well. Timely issue of commands facilitates faster operation and the vehicle can complete its task on time, if there is a time constraint. To ensure the robot doesn't wait indefinitely for the user prompt there is a time out embedded within the help phase. The time out can be configured based on the application or other requirements. A feature to include time out is available in ROS. If the time out occurs the robot can be made to abort the tasks to save energy and power and for safety.

Once the user initiates the commands and the vehicle is out of certainty the sensing process is restarted and the vehicle navigates as usual autonomously. This is the closed loop feedback that is established and is similar to the modified waterfall fusion model described under Chapter 4. Robot sensing, perception and control are integrated into the framework with human-interaction under uncertainty.

These are the different components of the framework and the next section describes the simulation of a vehicle navigation in Gazebo and ROS. The vehicle used is the model created and described from Chapter 5.

6.10 Simulation in Gazebo and ROS

The environment that the vehicle navigates is set up in gazebo. In order to create a terrain and include several features to the terrain a "world.sdf" file is created. The basics of gazebo was covered in detail in Chapter 4. Figure 6.6.1 shows the world.sdf file that was used to create the environment for simulation.

6.10.1 World SDF in Gazebo

The Figure 6.6.1 shows the snippet from the world model specifying the parameters for the environment.

W:\Documents\WRITE\model.world - Notepad++

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

model.world

```
<?xml version="1.0" ?>
<sdf version="1.4">

  <world name="default">
    <!-- A global light source -->
    <include>
      <uri>model://sun</uri>
    </include>

    <model name="heightmap">
      <static>true</static>
      <link name="link">

        <collision name="collision">
          <geometry>
            <heightmap>
              <uri>model://gazeboros/materials/textures/heightmap.png</uri>
              <size>200 200 10</size>
              <pos>0 0 0</pos>
            </heightmap>
          </geometry>
        </collision>

        <visual name="visual_abcedf">
          <geometry>
            <heightmap>
              <use_terrain_paging>false</use_terrain_paging>
              <texture>
                <diffuse>file://media/materials/textures/dirt_diffusespecular.png</diffuse>
                <normal>file://media/materials/textures/flat_normal.png</normal>
                <size>1</size>
              </texture>
              <texture>
                <diffuse>file://media/materials/textures/grass_diffusespecular.png</diffuse>
                <normal>file://media/materials/textures/flat_normal.png</normal>
                <size>1</size>
              </texture>
              <texture>
                <diffuse>file://media/materials/textures/fungus_diffusespecular.png</diffuse>
                <normal>file://media/materials/textures/flat_normal.png</normal>
                <size>1</size>
              </texture>
              <blend>
                <min_height>2</min_height>
                <fade_dist>5</fade_dist>
              </blend>
              <blend>
                <min_height>4</min_height>
                <fade_dist>5</fade_dist>
              </blend>
              <uri>model://gazeboros/materials/textures/heightmap.png</uri>
              <size>200 200 10</size>
              <pos>0 0 0</pos>
            </heightmap>
          </geometry>
        </visual>

      </link>
    </model>

  </world>
</sdf>
```

Figure 6.6.1 World Model SDF [82]

6.10.2 World Model SDF parameters

From Figure 6.6.1 we can see there are several parameters that need to be configured while creating a customized environment for the terrain. Below we describe some of the important parameters [82].

a) Model URI: By default the name assigned is sun.

b) Model name: This is section where most of the modifications happen. It is called as the “heightmap”.

c) Texture: This describes the type of element we want to add to our environment. The file has several textures added. There is greenery included by adding the “grass texture”, as shown in line 34 of figure 6.6.1.

6.10.3 Environment View

Figure 6.6.3 shows the environment where the vehicle navigates. The environment has uneven terrain with up/down ramps and flat ground.

6.11 Vehicle Simulation in Gazebo and ROS

Figures 6.11.1 to 6.11.9 show the screenshots of the vehicle navigation on an unknown terrain. The simulation world created is similar to that shown in Figure 6.6.3 having some peaks and valleys or up ramps and down ramps. Also a few blocks or surface obstacles are also introduced to illustrate the working of the sensor fusion framework described in Figure 6.2.

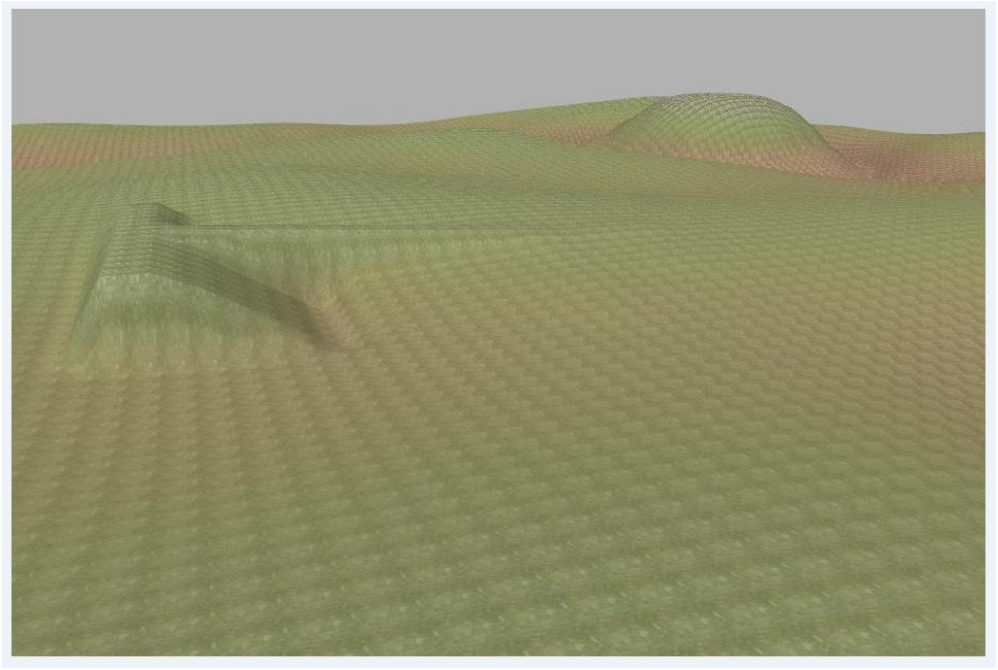

Figure 6.6.3 Simulation environment for the vehicle generated through gazebo world model.

Case 1: Surface Obstacle

Figure 6.11.1 shows the environment where the vehicle faces the presence of a block in the front. The terminal screen shows the variable for front obstacle set. Surface Obstacles could be rocks, blocks, trees or any object that is hindering the vehicle from moving forward to the next cell in the field. In the example shown the front obstacle is a black block in front of the vehicle.

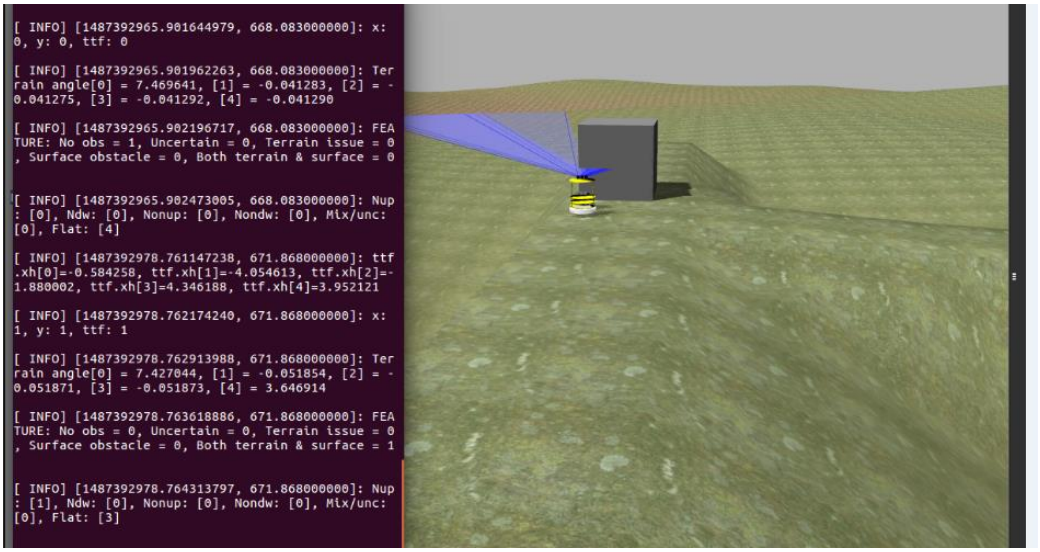

Figure 6.11.1 Handling an obstacle in front of the vehicle

Case 2: Surface Obstacle and Ramp close to each other

Figure 6.11.2 shows the environment where the vehicle faces the presence of a block in the front and then turns to its side and detects a ramp. On identifying it's a non-traversable ramp the vehicle turn again and resumes autonomous navigation. The terminal screen shows the variables for front and ramp obstacles set accordingly.

In order to turn backward, instead of doing a direct 180 degree turn the vehicle turns left twice and then proceeds forward.

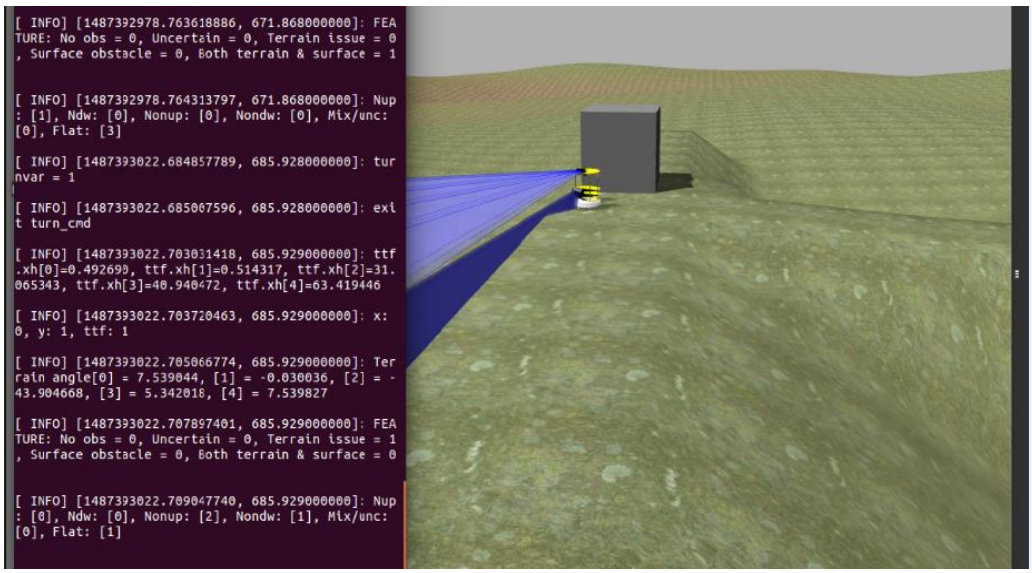

Figure 6.11.2 Handling an obstacle in front and at the side of the vehicle.

Case 3: Uncertainty handling and help phase

Figure 6.11.3 shows the environment where the vehicle is uncertain on moving forward or navigating. Such a situation could happen when the track fusion results and the value provided by the sensor is out of range or when there is a difference between them. In Figure 6.11.3 the vehicle is navigating near an obstacle in front and ramps beside it. So the system enters uncertain mode and initiates user help. On entering the command the vehicle navigates by moving forward. Since there is more area that can be traversed before the front obstacle the vehicle keeps moving forward until it is very close to the block. This is shown in Figure 6.11.4 where the command entered is shown on the terminal and the vehicle resuming autonomy is shown in Figure 6.11.5.

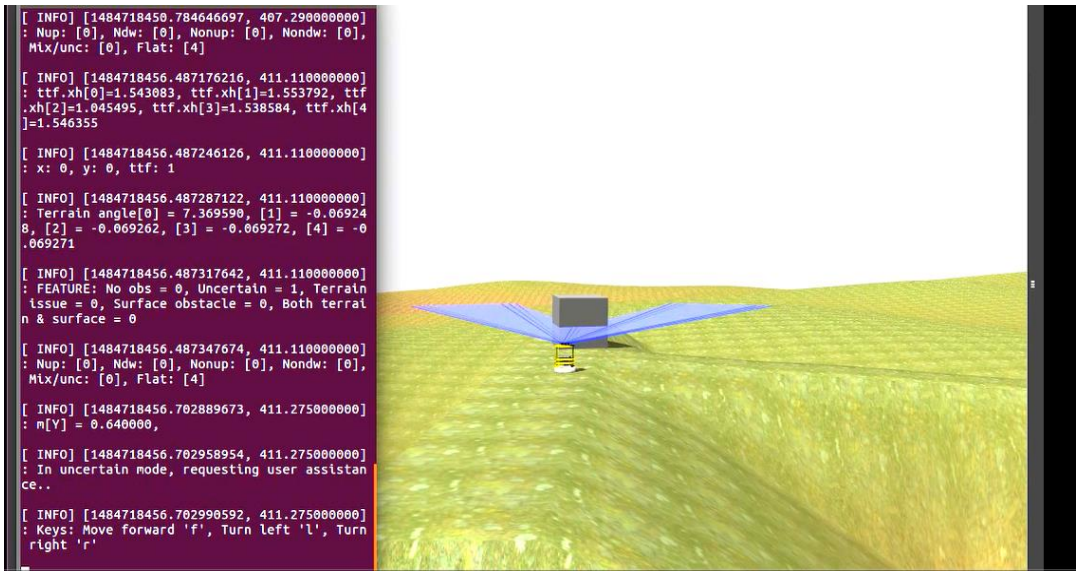

Figure 6.11.3 Handling uncertainty during navigation

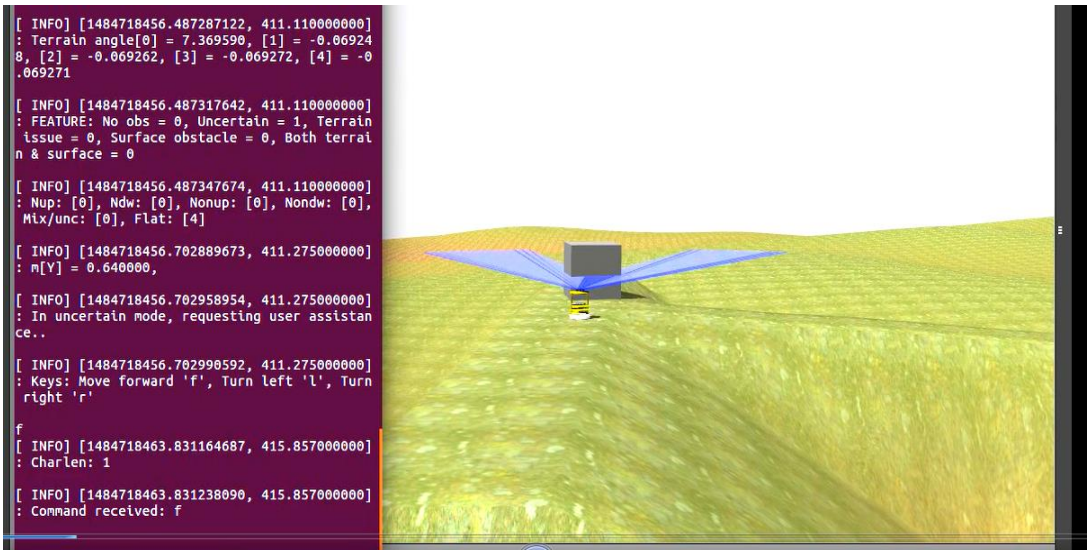

Figure 6.11.4 Help phase and user entering commands.

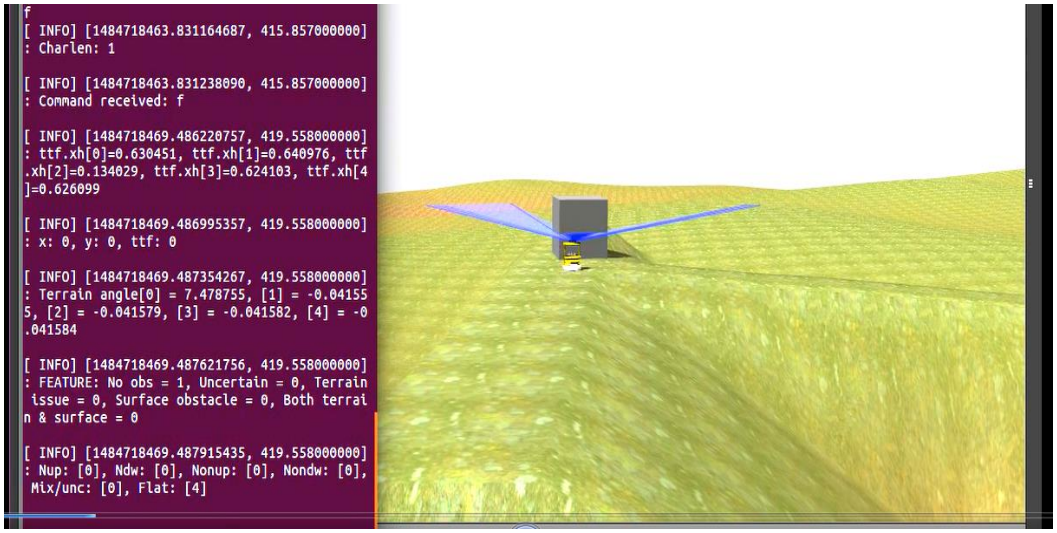

Figure 6.11.5 Vehicle resuming autonomous mode of navigation after help phase.

Case 4: Handling terrain issues: flat ground to non-traversable ramps

As described in ramp identifier section there are five possible scenarios for terrain issues. Figures 6.11.6 to 6.11.9 shows the environment where the vehicle faces these scenarios during navigation.

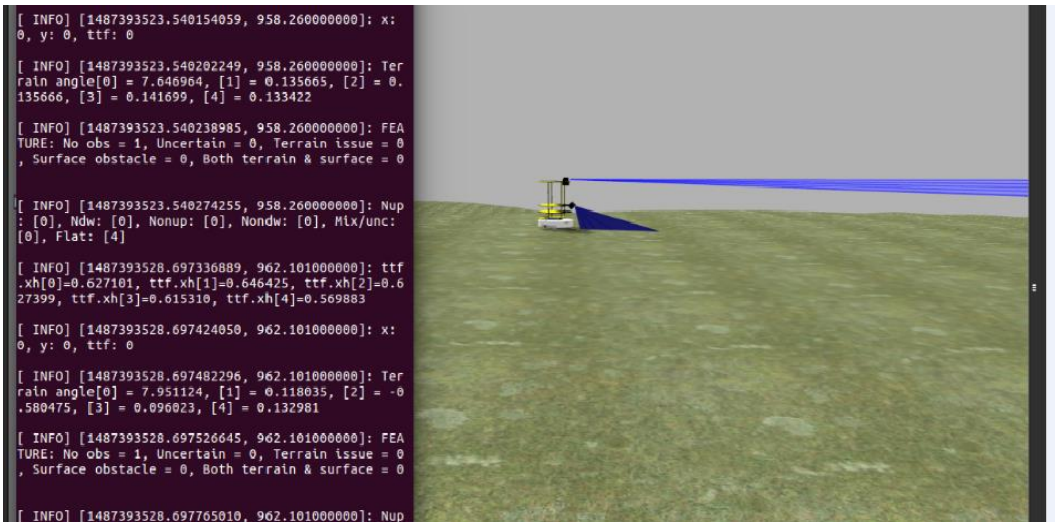

Figure 6.11.6 Navigating a flat terrain or flat ground

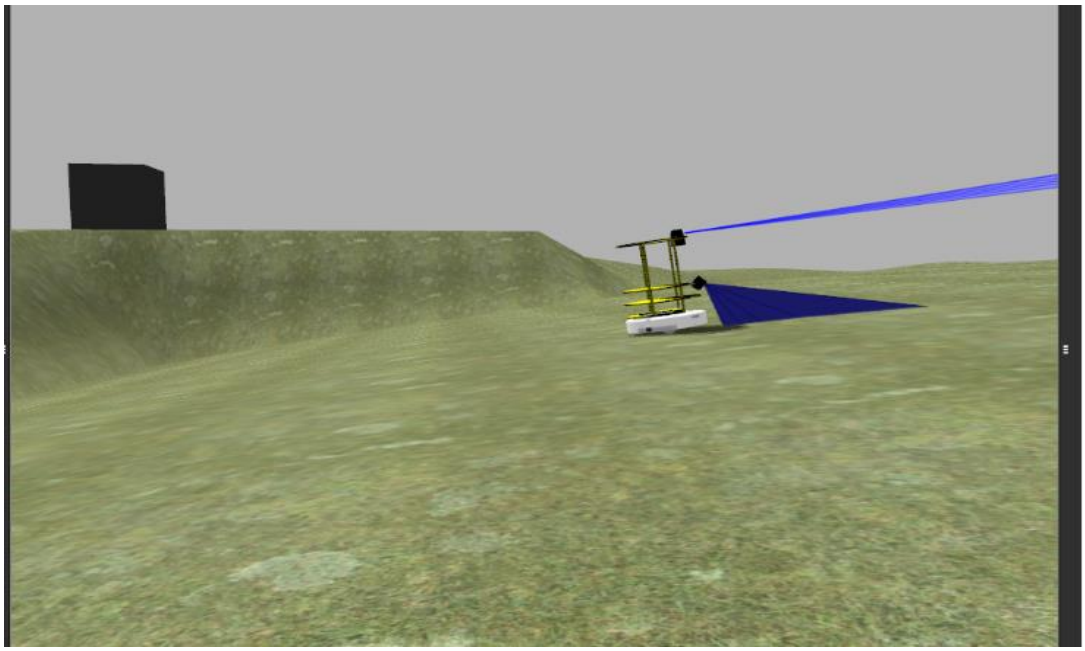

Figure 6.11.7 Navigation on a traversable up ramp

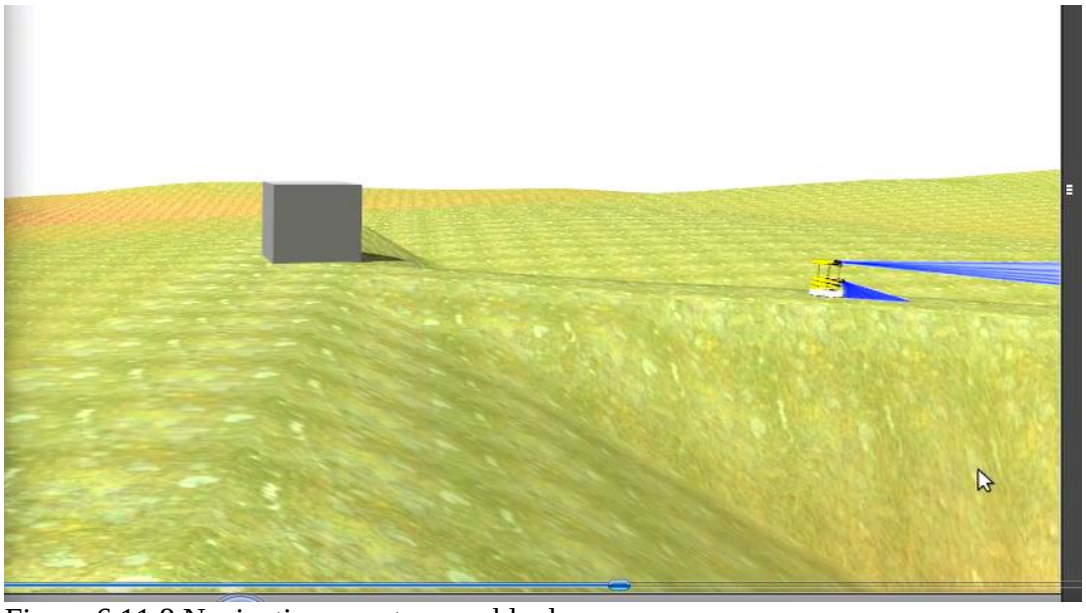

Figure 6.11.8 Navigation on a traversable down ramp

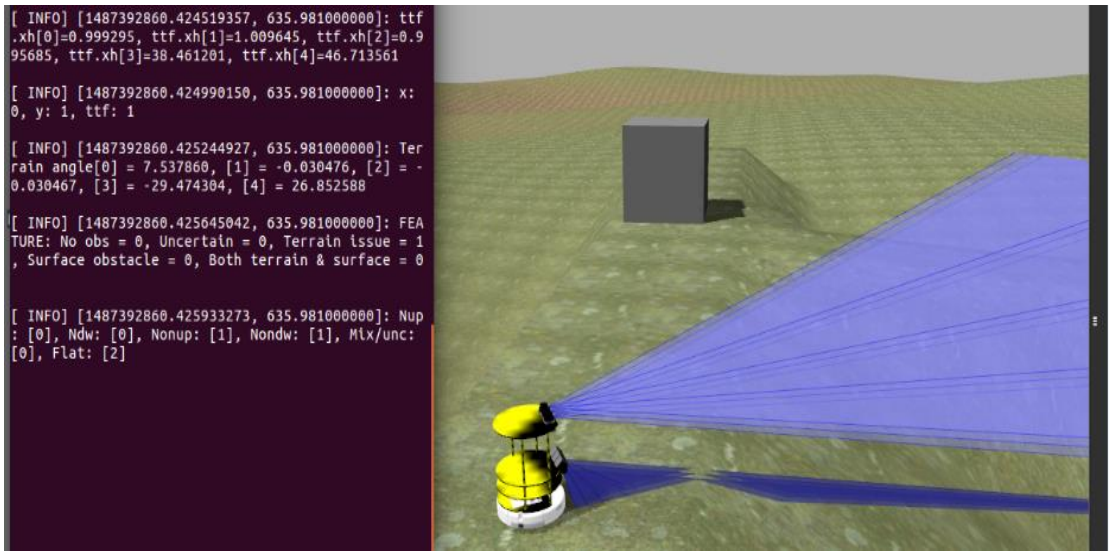

Figure 6.11.9 Navigation on a non-traversable ramps

6.12 Framework Evaluation

As described in [54, 55, 56], there doesn't exist one type of sensor fusion framework for a particular application. So this makes comparison between different fusion frameworks hard since the application for which the fusion framework is designed influences the choice for different stages of the framework and the type of fusion methods i.e., signal, image or decision that can be used.

However, in this section we evaluate the framework based on the basic principles of fusion consistency, data flow between the different stages, degree of autonomy of the vehicle, level of interaction and finally the computation load on the system with and without considering ROS.

6.12.1 Data flow in the framework and consistency

In this framework there are four stages of fusion. First at the track to track fusion, second at the pattern classification, third at the ramp classification and finally

at Dempster-Shafer fusion block that gives the confidence percentage. At each of these stages the data flow and fusion is consistent with respect to Dasarthy's classification of fusion methodologies [51, 52]. Table 6.12.1 shows the type of data is flowing between the different stages in the framework. The data flow between the stages is consistent with the rules outlined in [51, 52] validating the correctness of the framework formulated for the navigation application encapsulating human robot interaction.

Table 6.12.1: Data flow between the stages

Stage	Data Type IN	Data Type OUT
Track to Track Fusion	Sensor data	Feature
Pattern Classification	Feature	Feature/Decision (uncertainty)
Ramp Classification	Feature	Feature/Decision (uncertainty)
Dempster-Shafer	Feature	Decision (uncertainty)

6.12.2 Autonomy level

Another important measure of the system performance is the amount of interaction or autonomy level of the vehicle as described in [88, 89]. In this framework, the interaction time between the user and the vehicle is at minimum possible. The vehicle signals for help from the user only when absolutely necessary and switches back to the autonomous mode on exiting the help phase. This ensures more independent operation of the vehicle and more time for the user to handle his

tasks.

We thus have a vehicle that is not tele operated but an autonomous vehicle with collaborative control [73, 85]. For the field that is shown the vehicle navigates autonomously through all “cells” except at the cells where the help phase is initiated due to uncertainty.

6.12.3 Communication Between User and Vehicle

As outlined in [87], the type of information communicated between the entities and the effective use of this information by the vehicle not just plays a vital role but standardizes our evaluation criteria. We have used communication between the user and the vehicle through command numbers, which serves two important benefits, namely easier for the user to type into his console and faster decoding by the vehicle.

6.12.4 Scalability

The scalability of the framework is another important parameter that is necessary to quantify the system. There are two possible types of scalability namely scalable with respect to the number of sensors that can be added and scalable with respect to the size of the field the vehicle is navigating. The computation of the system using perf can be used to calculate the performance of the centralized processor that is handling the fusion of all the sensors into the system.

The framework is centralized in design and thus requires evaluating the computation load on the central computer that runs the framework. The operating system used is Linux and by using the Linux tool, perf [90, 91], we can measure the computation performance of the system. The computation performance could also

help identify the bounds of the sensor inputs for the framework. Since the most computation intensive part is the track to track fusion for signal level fusion and since the sensor inputs are directly connected to this block measuring the computation load of this stage of the framework will help estimate the maximum number of sensors that can be supported by the system.

The maximum possible area of navigation is also dependent on the power constraints of the central processor. With a long battery capacity the vehicle can navigate the field of any size provided there is user available to give the input when the help phase is sought.

Figures 6.12.4.1 and 6.12.4.2 show the performance for the most computation intensive part of the framework i.e., track to track fusion block with Robot Operating System (ROS) and without ROS. The values for IPC, branch misses and page faults is higher for the block with ROS when compared to the block without ROS. This is because of the several I/O and communication load due to ROS. Hence, the maximum number of inputs that can be supported by the framework is dependent on the type of processor that runs the centralized fusion node and the communication load that arises due to ROS. For this application increasing the number of sensors would not be required since we have the necessary information to decide if the cell in front of the vehicle is navigable or not.

Parameter	With ROS	Without ROS
Instructions	10,848,140,724	9,450,817
Context Switches	970,881	42
Branches	2,218,697,599	862,356
Branch misses	172,617,622	16,644

Figure 6.12.4.1 Performance of the track fusion with and without ROS

CHAPTER 7: COMPLETE COVERAGE PLANNING

In [20], the authors summarize a few planning algorithms for the vehicle. Also, in [20], the vehicle maneuvers over the obstacle by drawing a collision avoidance boundary and navigates along local paths computed while encountering obstacles and resumes navigation along the regular path to its destination once it is dealt with the obstacle. To achieve human interaction it is necessary to have some form of dialogue between the operator and the vehicle. Command numbers are used to achieve bidirectional communication during the assist phase when the vehicle seeks help. This is similar to the collaborative control mentioned in [84].

7.1 Experimental Setup

In this chapter, we consider a grid of size 5 by 9. The field is decomposed into cells forming a 2 dimensional matrix. The robotic vehicle is non-holonomic (i.e., can make only 90 degree turns and cannot move in all directions). The vehicle traverses the field column-by-column covering as many cell blocks as possible. The borders of the field are considered reachable and are assumed to be obstacle free so that the vehicle can navigate the perimeter of the field. The vehicle makes 90 degree turns at the top and bottom cells of the field either turning right at the top of the column or turning left at the bottom of the field, to move to the neighboring column. Two types of obstacles are considered namely, trees and ravines. Trees are assumed to be confined within a single cell boundary. Ravines could spread to more than one cell.

The vehicle is represented as a turtle, one of the possible shapes available in the graphics tool. The method of navigation and the obstacle avoidance are described in the following sections. The programming language used is Python 2.7 [92], IDE used was Spyder [93] and graphics implemented using Turtle Graphics tool [94]. The results of the complete coverage planning are similar to that described in [74].

7.2 Robotic Vehicle Navigation

The vehicle traverses the field, cell-by-cell, from top to bottom or bottom to top along each column. It makes left turns when it reaches the bottom of column (i.e., highest index value for row) and makes a right turn when it reaches the top of the column (i.e., row index 0). The typical layout for the simulation is shown as in Figure 7.2.1.

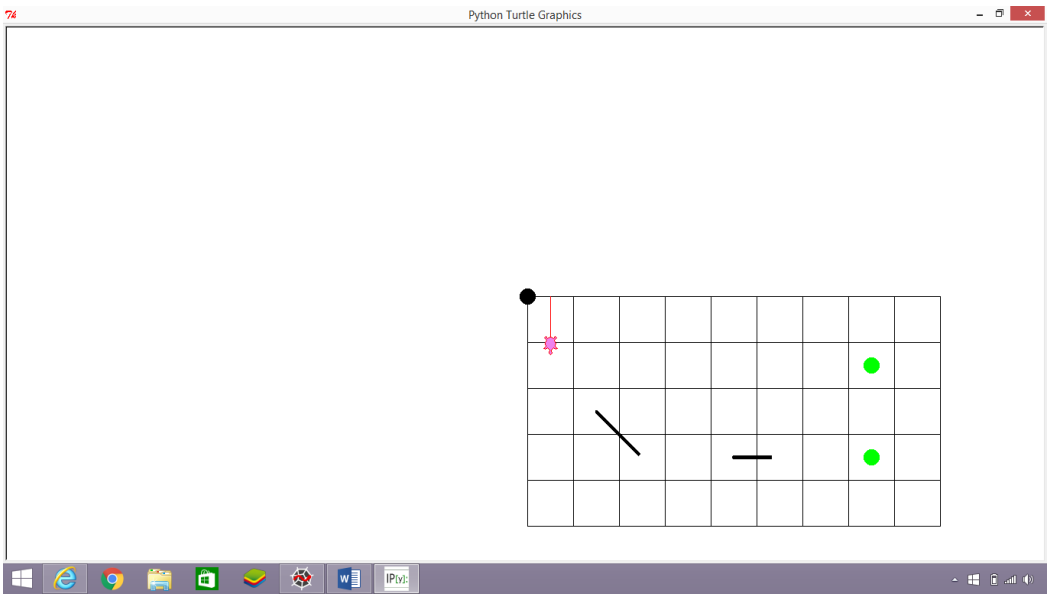

Figure 7.2.1 Simulation Environment.

The vehicle's regular traversal is as shown in Figure 7.2.2. The vehicle's trace is shown in red. We can see from the figure the vehicle covers all the cells while travelling through each column of the grid. Figure 7.2.2 assumes no obstacles in the field.

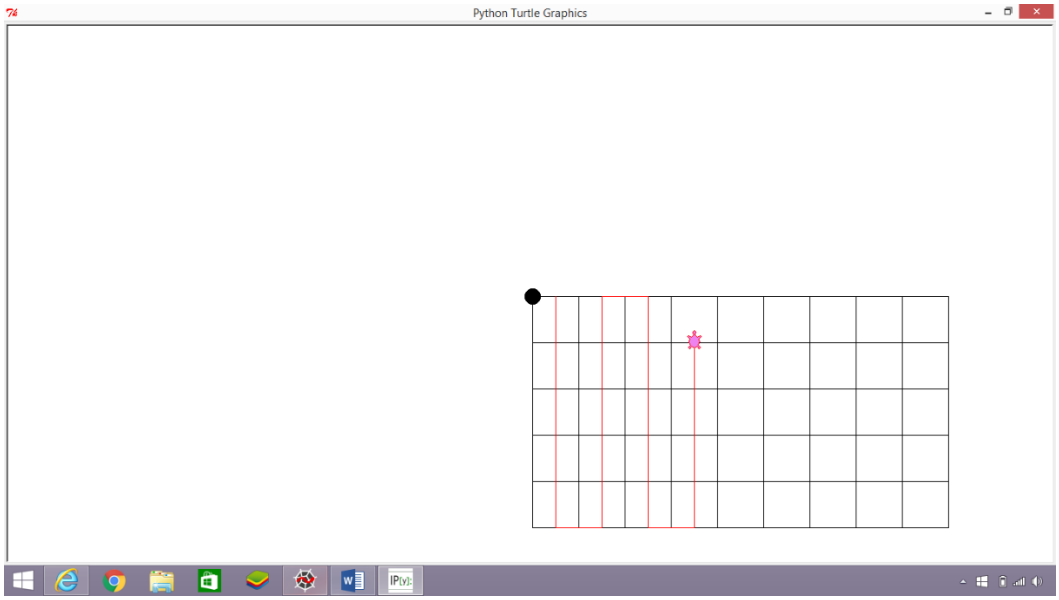

Figure 7.2.2 Regular navigation with vehicle turns at the top and bottom cells of a column.

The simulation environment with obstacles is as shown in Figure 7.2.3.

Trees are drawn as Green circles and Ravines are represented as thick black lines.

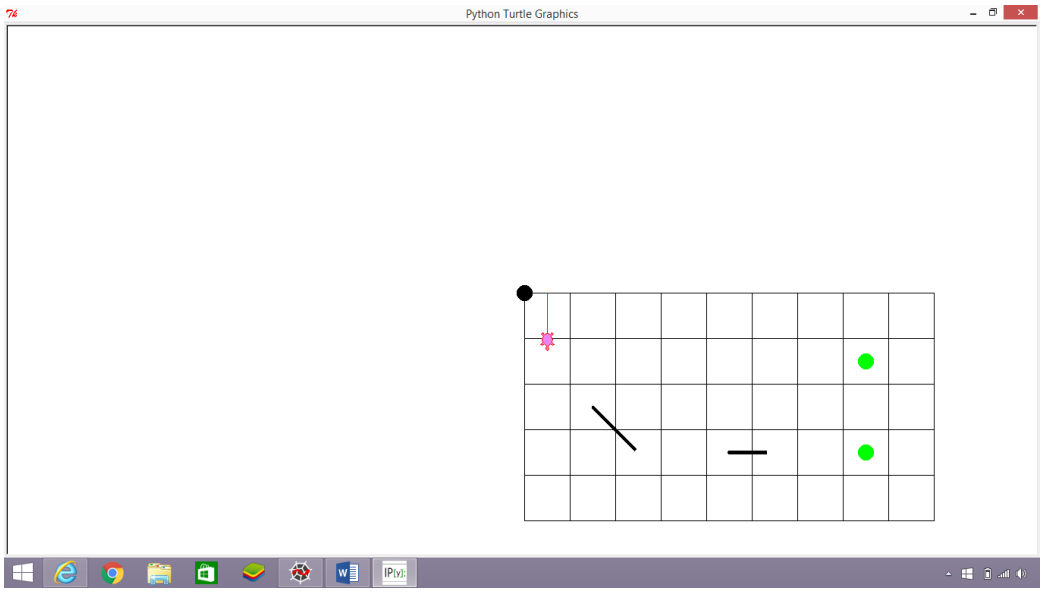

Figure 7.2.3 Field with ravines a thick lines and trees as green circles.

7.3 Obstacle Avoidance

There are two types of obstacles, namely, trees and ravines. They are discussed in the following sub-sections.

7.3.1 Handling Trees

Since the goal of the vehicle is to cover as much area as possible, when faced with a tree or ravine, the vehicle tries to find a path over it using the information from its visited list. As shown in Figure 7.3.1, when a tree is found, the Turtle robot traverses along the path shown in red around the tree through the cells it had previously visited. Since the vehicle has only information about the cells in the left column, it uses that

information to avoid the tree.

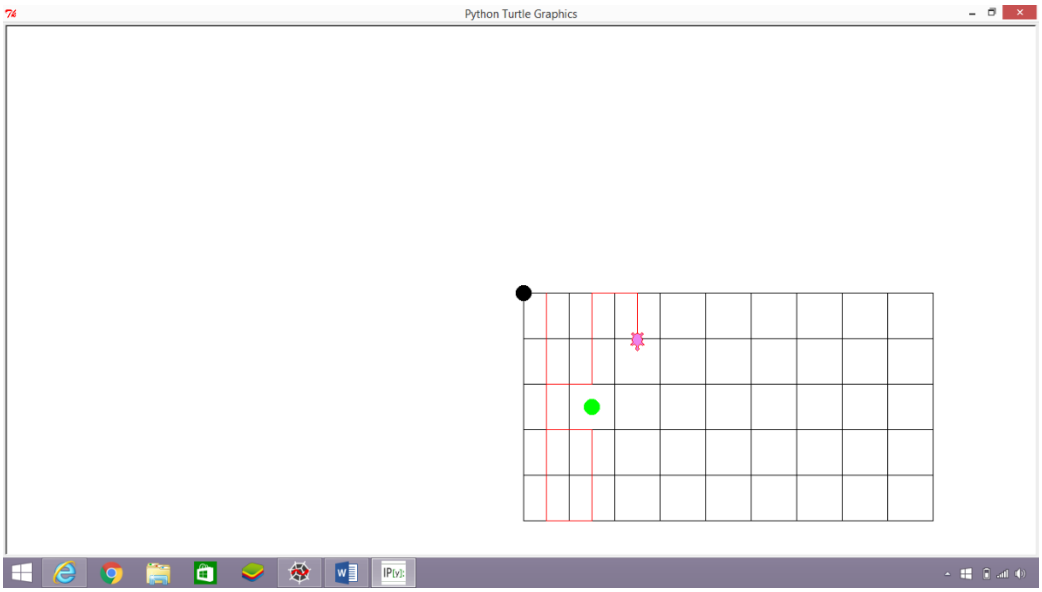

Figure 7.3.1 Navigation around a tree obstacle.

7.3.2 Handling Ravines

Ravine is another obstacle the vehicle encounters during its navigation. Figures 7.3.2.1 – 7.3.2.6 shows the navigation of the Turtle in the presence of a Ravine. When the vehicle encounters the Ravine for the first time it sees it as a tree and avoids it by going through the neighboring column cells as shown in the Figure 7.3.2.1.

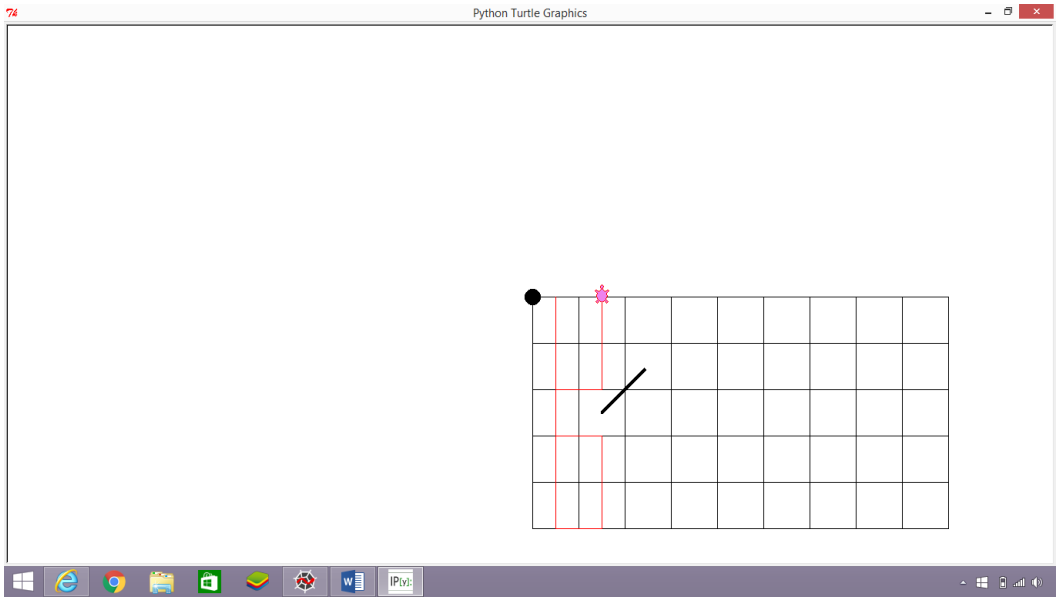

Figure 7.3.2.1 Navigation through a ravine.

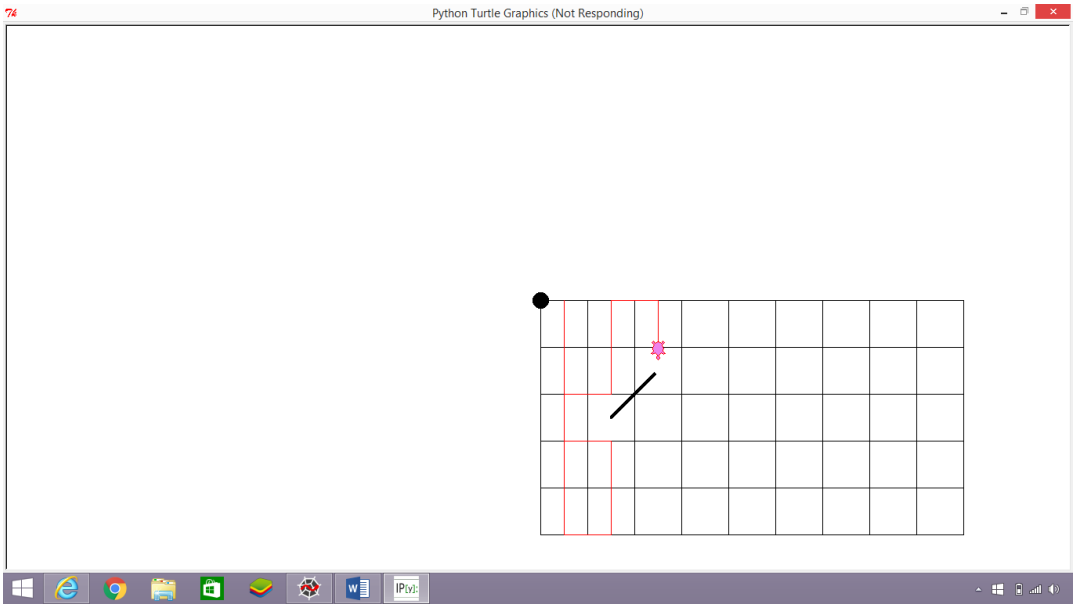

Figure 7.3.2.2 Initiates help phase to interact with the user.

During traversal through the neighboring column when it senses the continuity of the ravine, the vehicle looks up its visited cell history and identifies it cannot follow the same technique of going around the obstacle cell which is shown in Figure 7.3.2.2.

When the turtle realizes this, it queries the user for assistance. In the help phase, the communication between the user and the vehicle is a dialogue of command numbers as illustrated in Figure 7.3.2.3. The options shown are the most fundamental commands required to drive the vehicle.

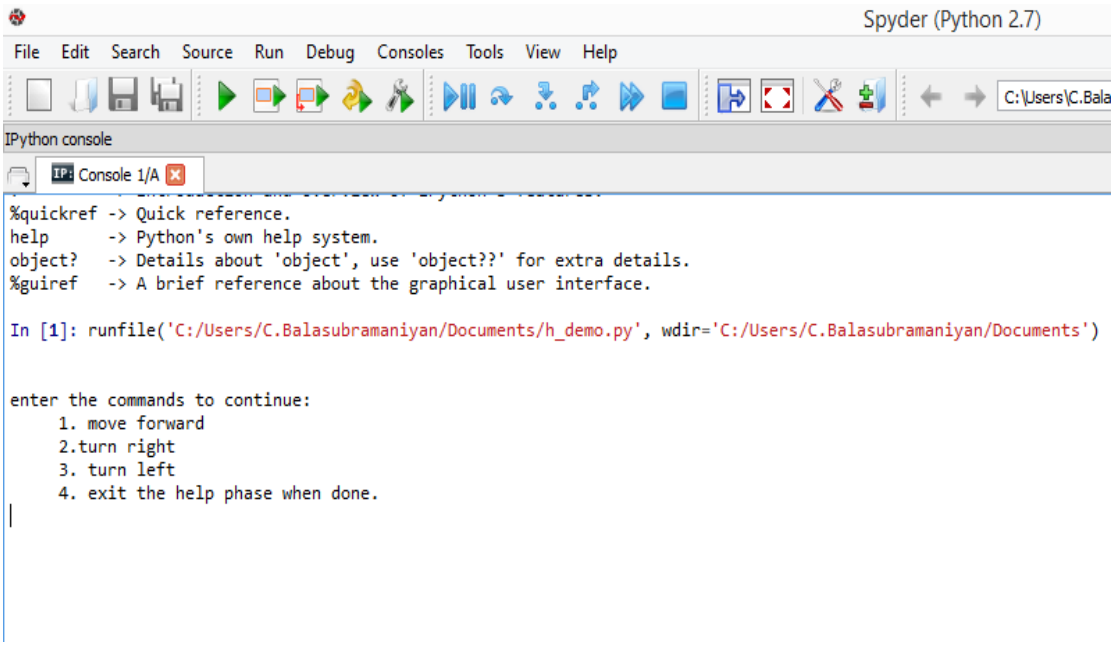

Figure 7.3.2.3 Help phase options presented to the user to steer the vehicle away from the obstacle.

In order to move the vehicle away from the ravine, the user commands the vehicle to turn right twice by entering the command number 2. Alternatively, the user could also enter 3 twice making the vehicle turn left twice. These actions are shown in Figure 7.3.2.4 and 7.3.2.5.

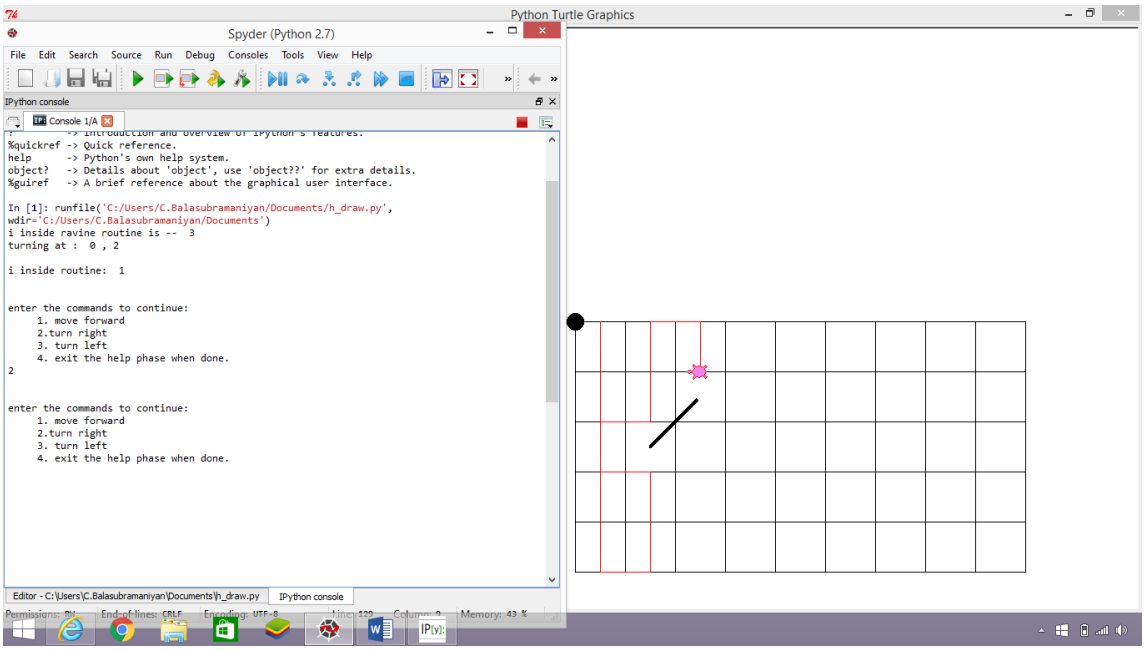

Figure 7.3.2.4 Turn right keyed by the user.

To turn the vehicle backwards instead of using 180 degree turn the vehicle is programmed to turn only by 90 degrees left or right. So to turn backwards the operator needs to enter turn left or turn right twice. Here the operator turns the vehicle right twice to achieve the 180 degree turn.

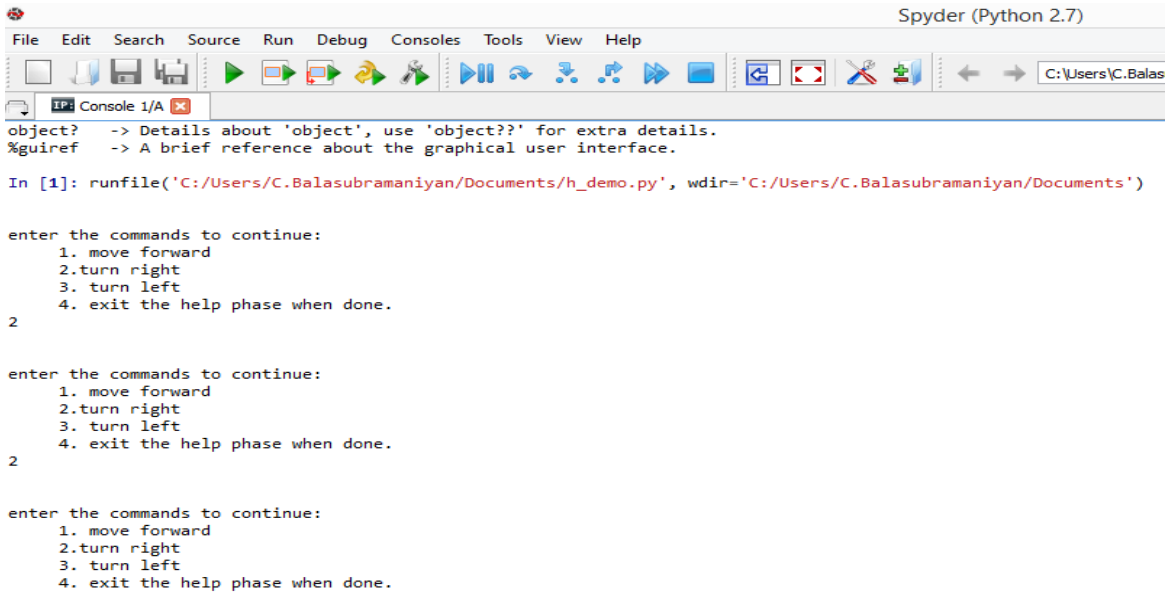

Figure 7.3.2.5 Turn right keyed again.

Since the goal of the help phase is to recover the vehicle from its uncertainty, the user gives only the minimum set of commands to bring the vehicle to a safe position and exits the help phase. The vehicle now resumes its regular traversal moving along the regular path as shown in Figure 7.3.2.6.

Collaborative control is an example of bi-directional communication. The two entities that are involved in the communication are the robot and the operator. When the robotic vehicle interacts with the user through such help numbers the operator acts like a peer to the vehicle and the vehicle engages in coordinated task with the operator instead of following the task of navigation alone.

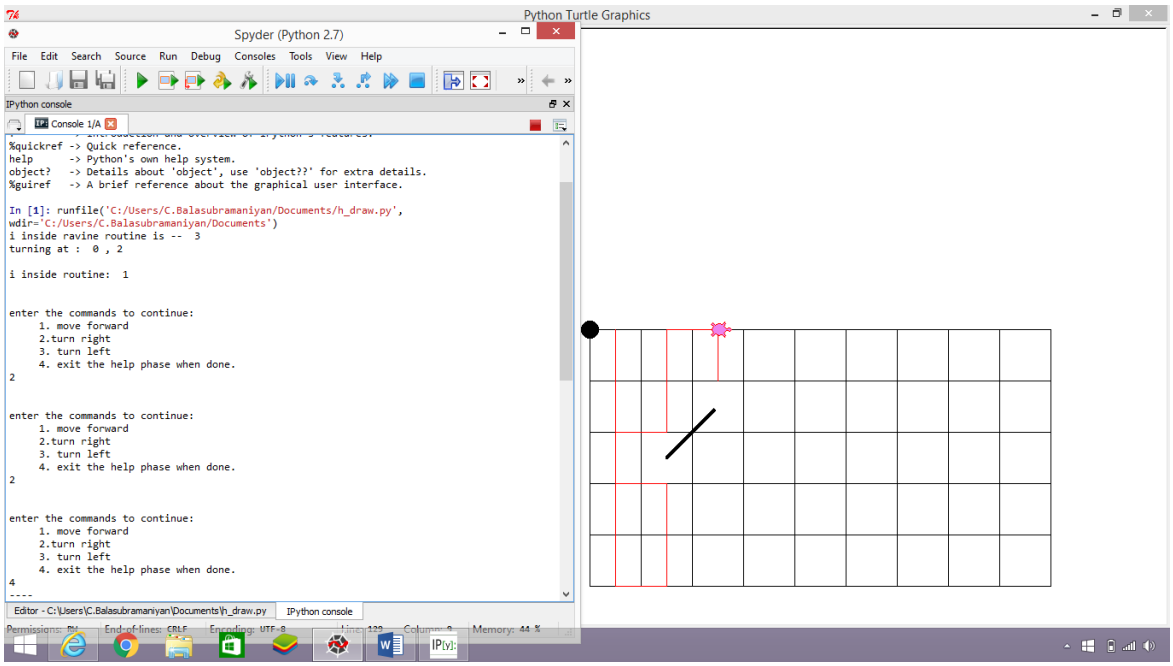

Figure 7.3.2.6 Autonomous mode resumed by the turtle.

From Figures 7.3.2.1-7.3.2.6, we can see that the vehicle could traverse the entire column where user request was initiated. Hence, during its traversal through the next neighboring column it visits those cells from the previous column that are obstacle free as shown in the Figure 7.3.2.7. In the Figure 7.3.2.8, we can see that the vehicle continues its regular path again after visiting the non-visited cells. This ensures maximum area of the field can be covered and the turtle records the positions of the obstacles both trees and ravine as it navigates the field.

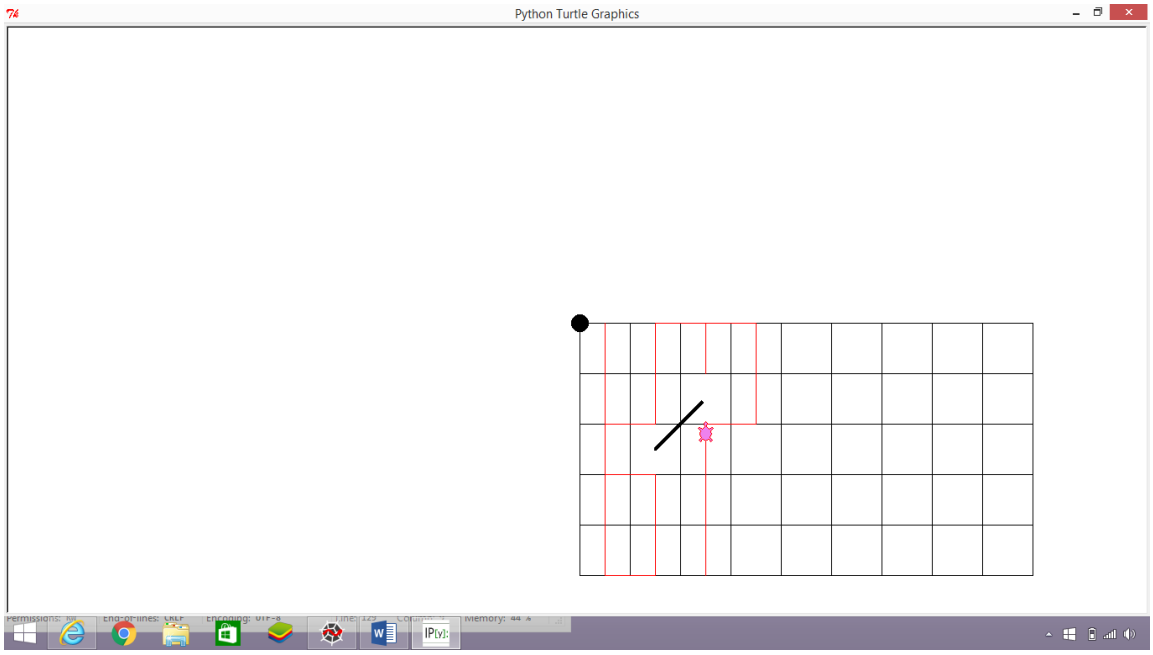

Figure 7.3.2.7 Vehicle visits the cells ignored previously due to the obstacle.

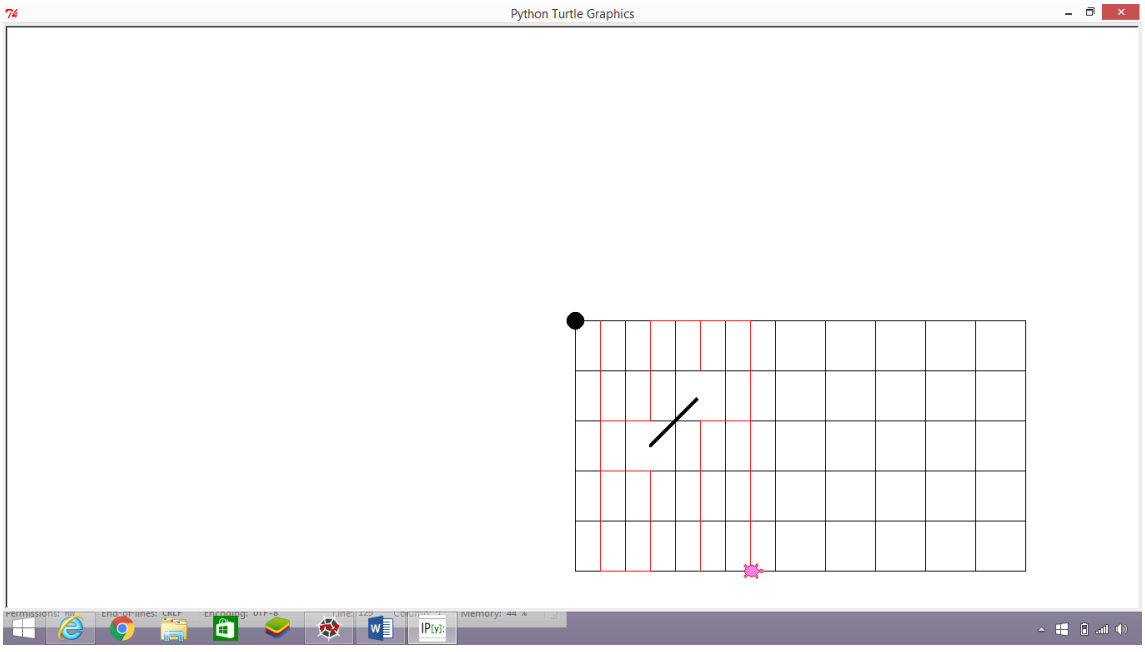

Figure 7.3.2.8 Vehicle resumes its regular navigation until it reaches its destination after covering the obstacle free non-visited cells.

7.3.3 Handling both Trees and Ravines

In this section, we cover an application involving both trees and ravines in the field. The previous two examples described handling the tree obstacle and the ravine individually. The following example shows the presence of both these obstacles in the field but at different positions. The vehicle traversal through this field is shown in Figures 7.3.3.1 – 7.3.3.13. Figures 7.3.3.2 - 7.3.3.10, shows how the vehicle navigates in the presence of ravines in the field. Figures 7.3.3.11 - 7.3.3.12 show the tree obstacle avoidance which is less complex and doesn't required user assistance. The vehicle at the destination is shown in Figure 7.3.3.13.

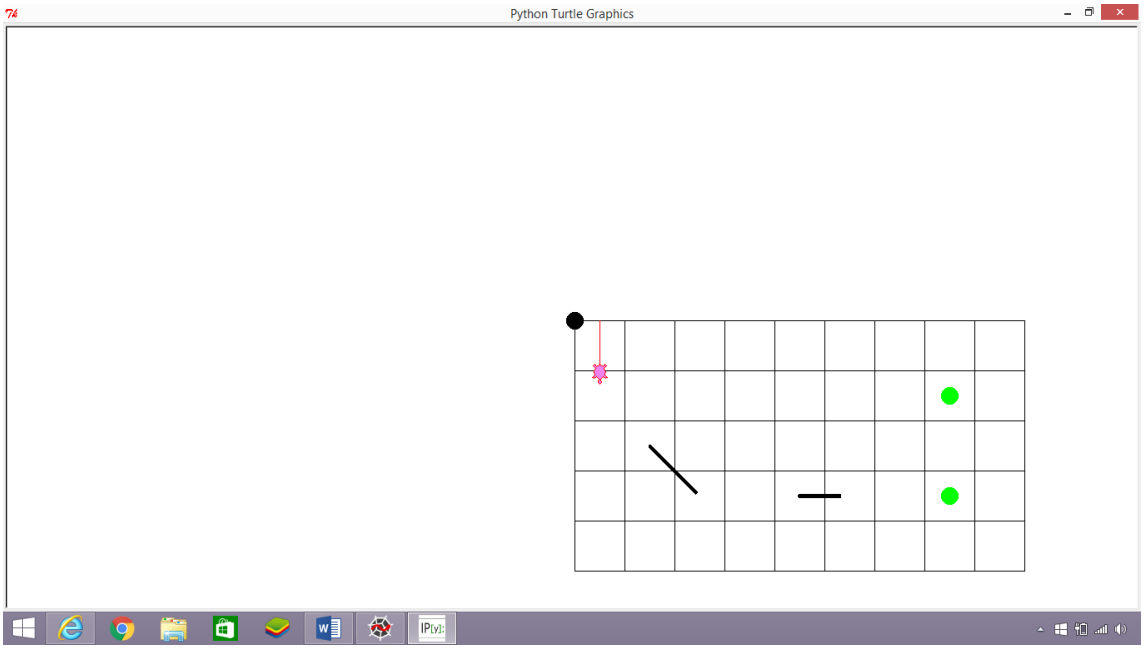

Figure 7.3.3.1. Simulation Environment with ravines a thick lines and trees as green circles.

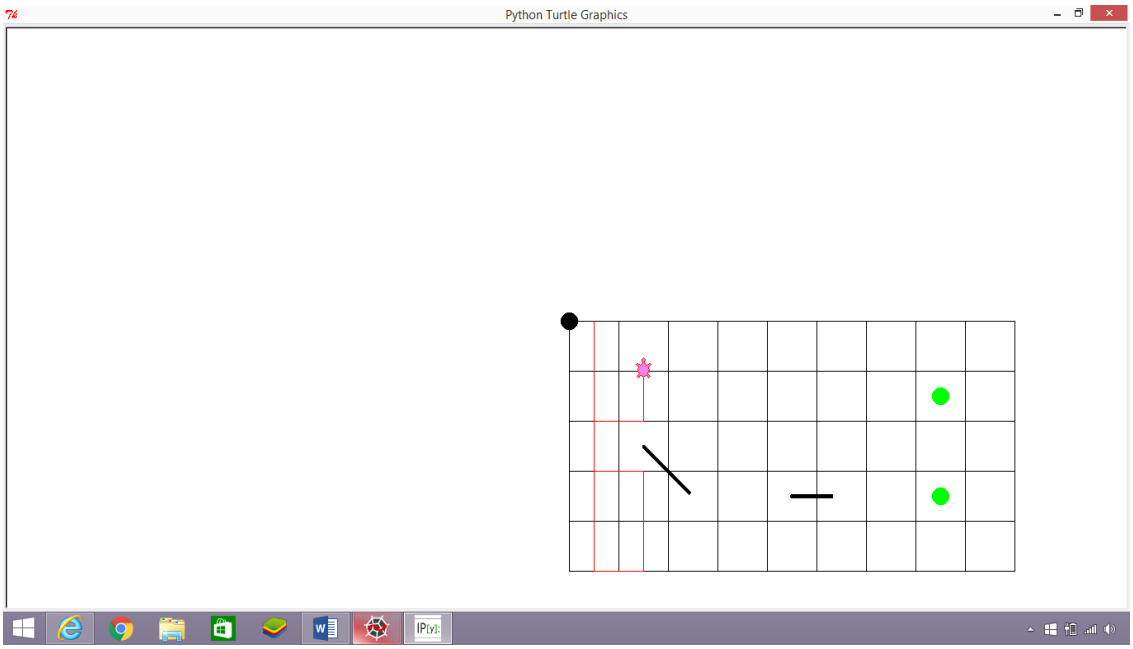

Figure 7.3.3.2. Vehicle navigates through the neighboring column to avoid the obstacle.

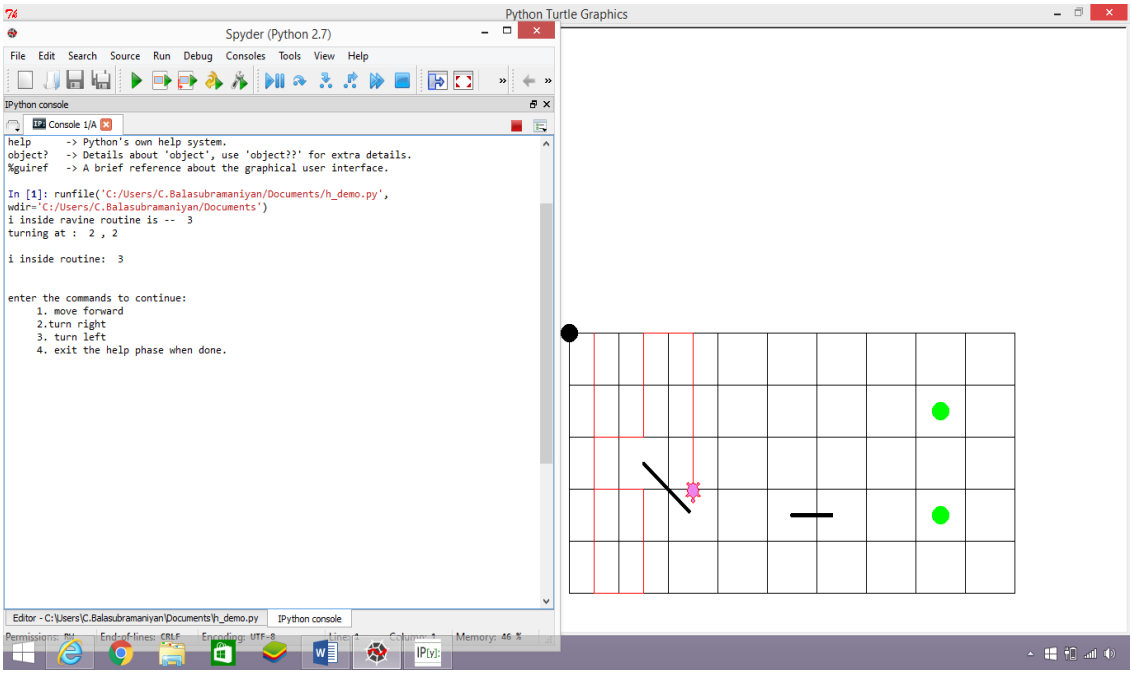

Figure 7.3.3.3. Help phase invoked when the turtle identifies a continuous obstacle across cells.

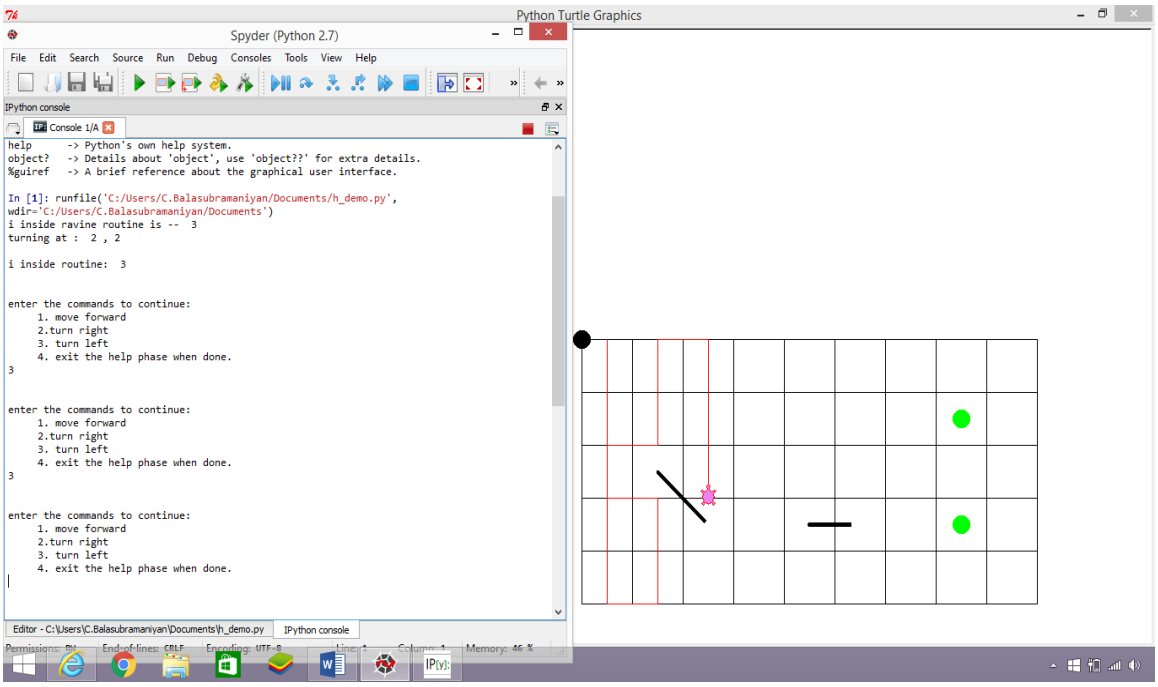

Figure 7.3.3.4. User keys in a sequence of commands to bring the vehicle to safe position.

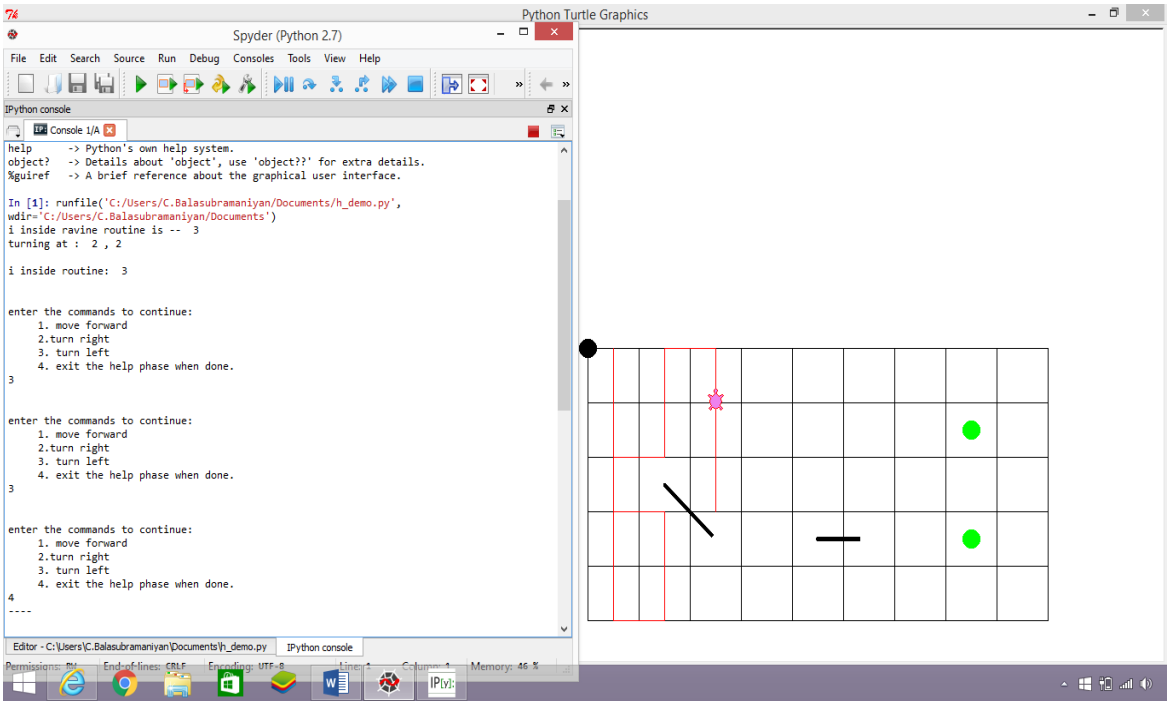

Figure 7.3.3.5. Vehicle switches to autonomous mode.

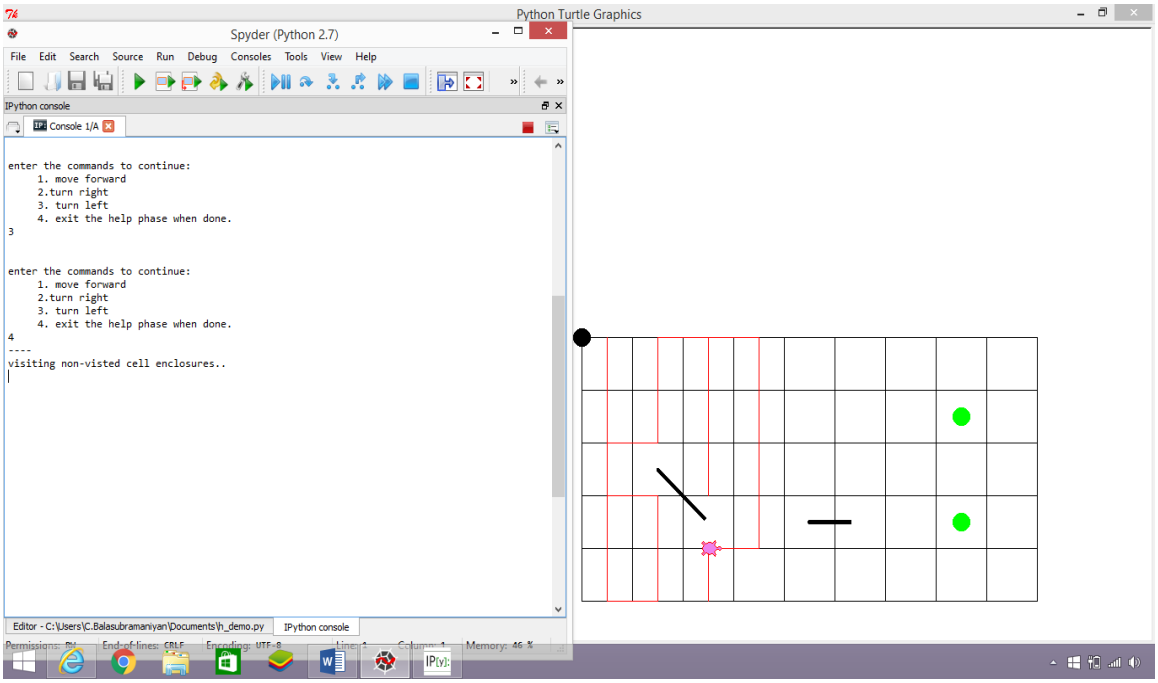

Figure 7.3.3.6. Covering the cells below the obstacle cell.

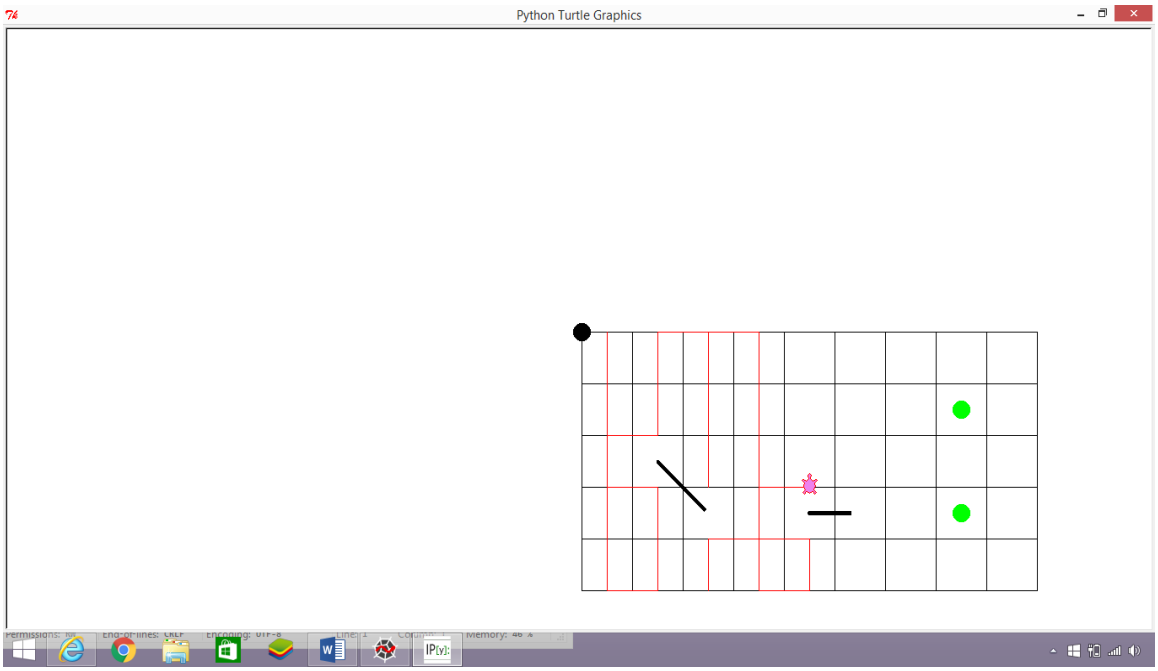

Figure 7.3.3.7. Vehicle goes through the neighboring cell to avoid the obstacle.

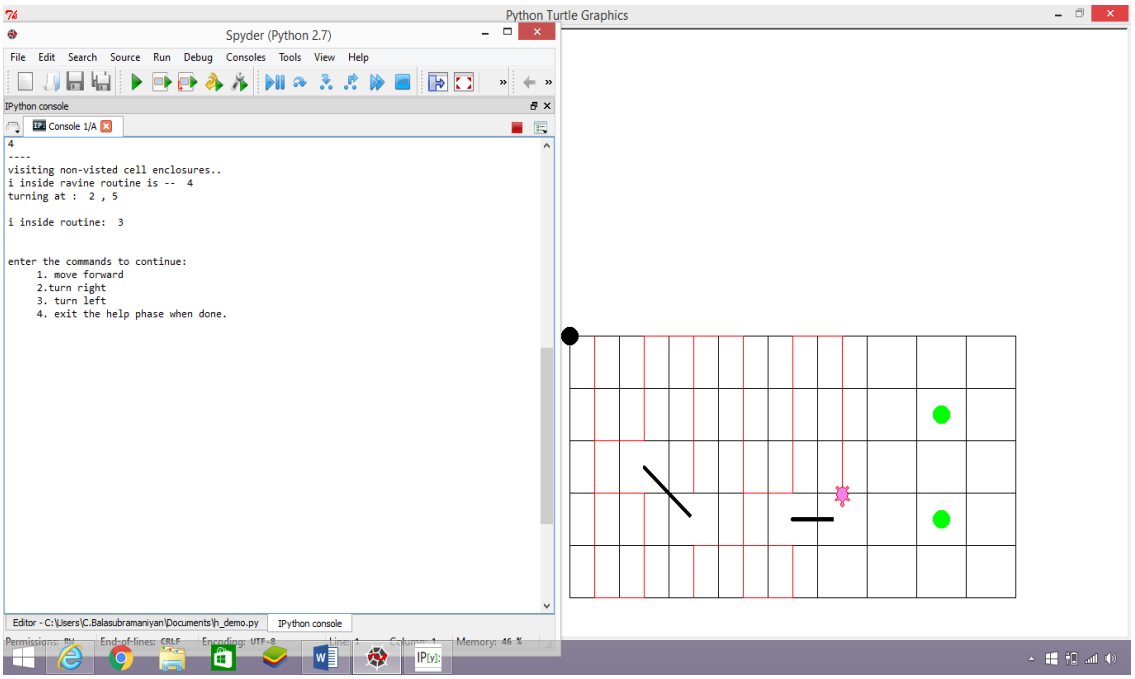

Figure 7.3.3.8. Help phase invoked similar to Figure 4.3.3.3.

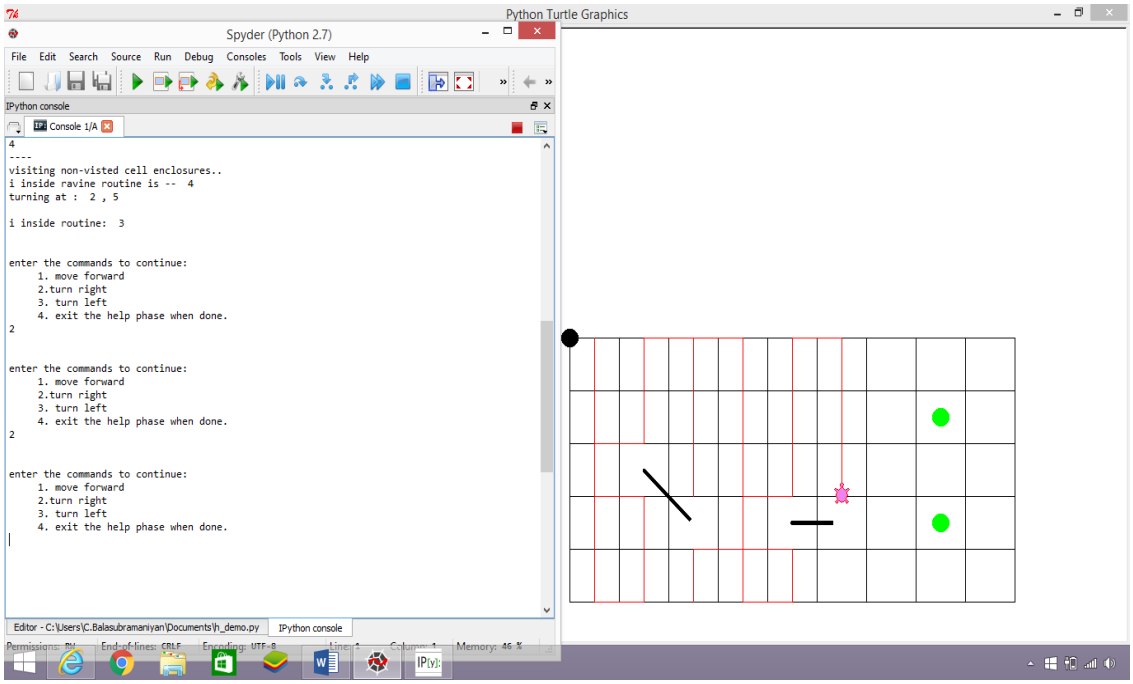

Figure 7.3.3.9. User controls the vehicle through commands

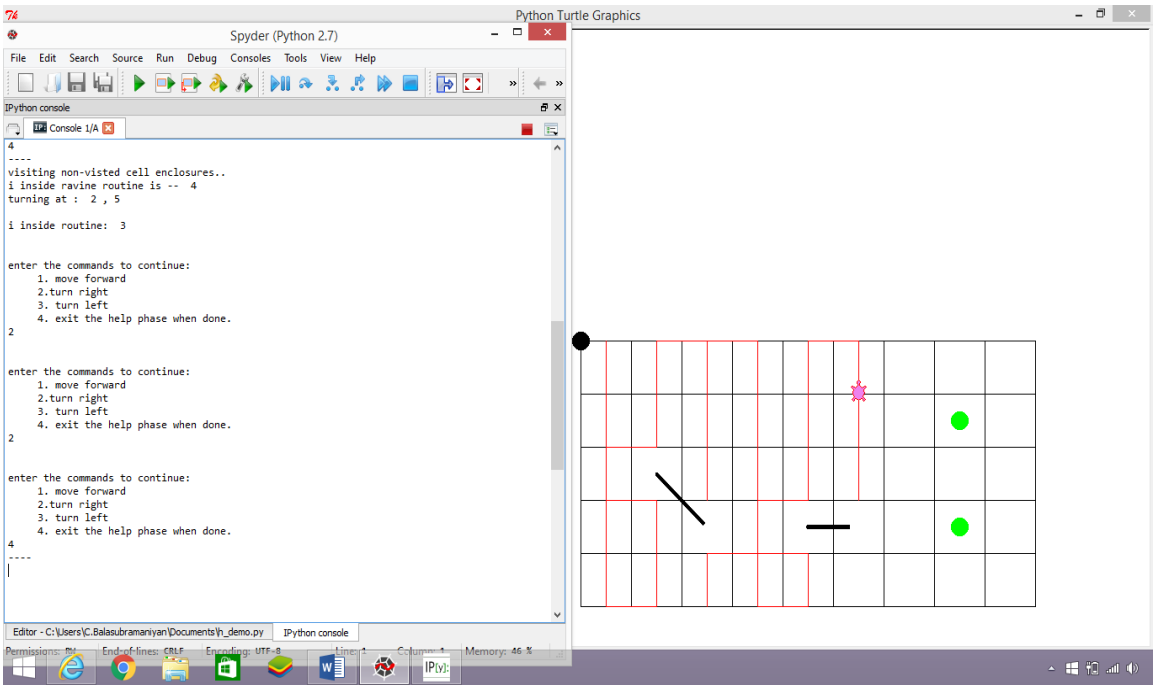

Figure 7.3.3.10. Vehicle swings back to autonomous navigation.

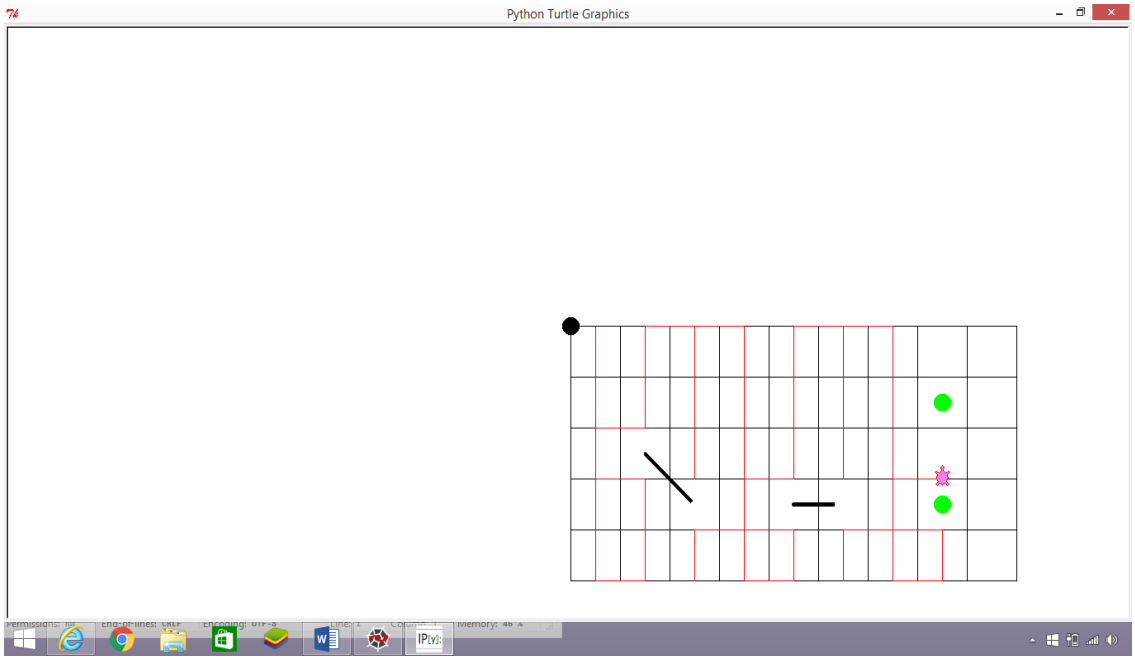

Figure 7.3.3.11. Vehicle avoids the tree by navigating through the neighboring column.

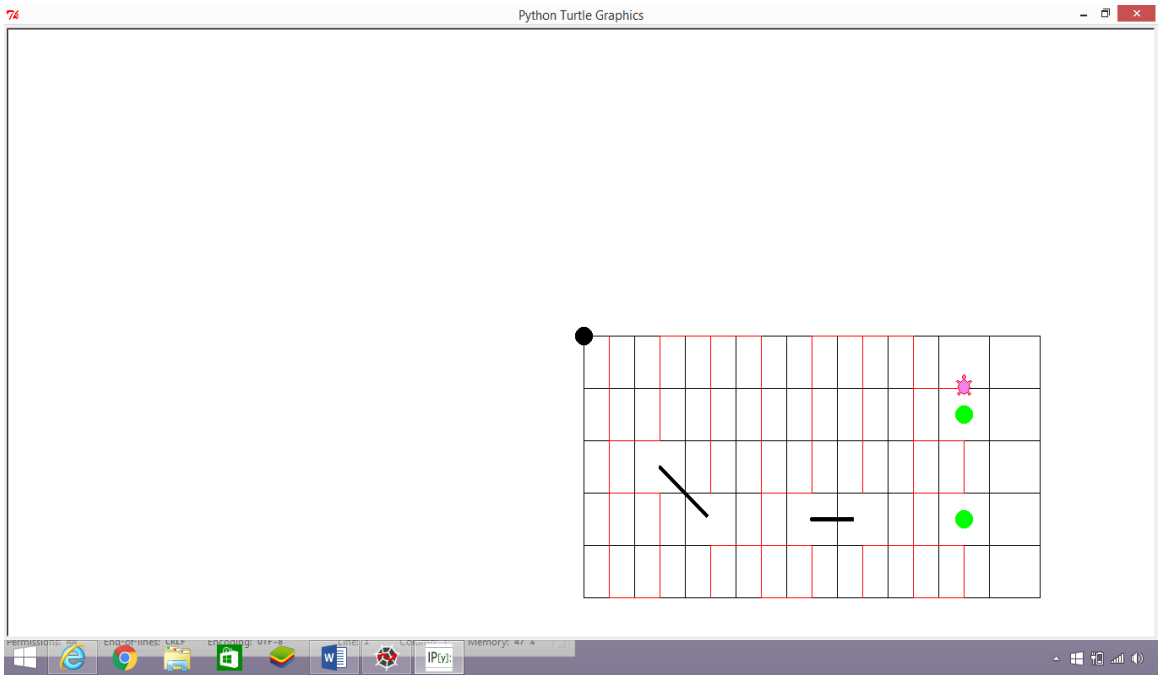

Figure 7.3.3.12. Vehicle avoids the tree by navigating through the neighboring column.

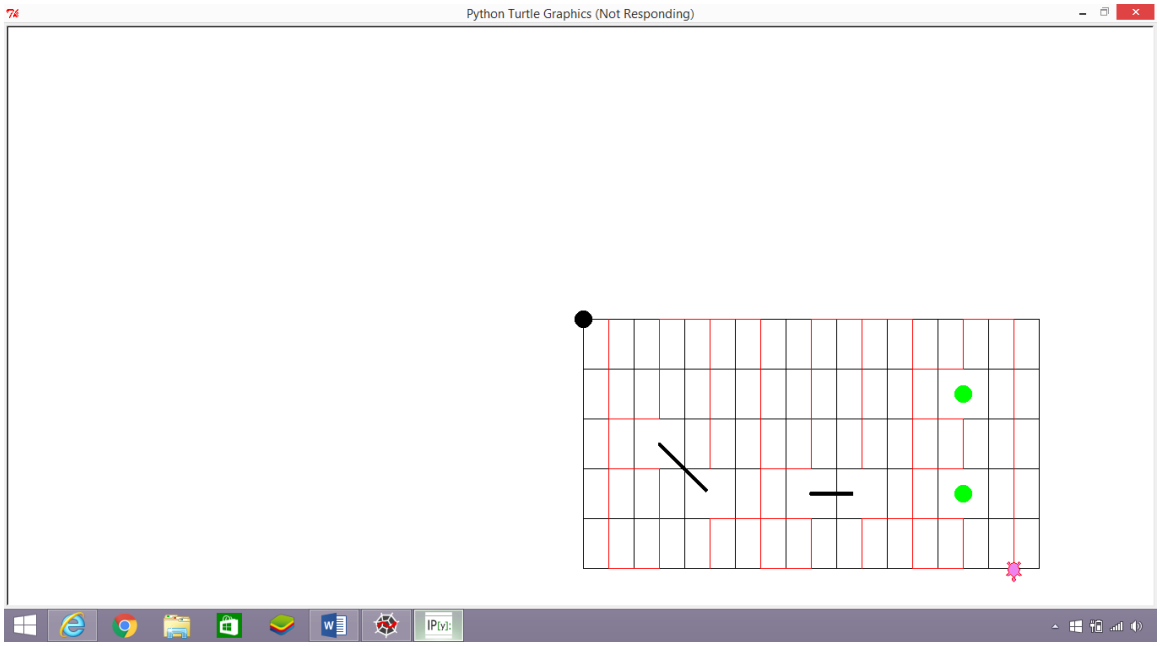

Figure 7.3.3.13. Vehicle at the final goal position.

Let us consider another example with the field shown in Figure 7.3.3.14. The field has more obstacles and the obstacles are placed close to each other. Figures 7.3.3.15 – 7.3.3.28 show the vehicle navigation in the field. The Help phase is invoked when the vehicle becomes indecisive and the vehicle is capable of autonomous navigation even when obstacles are located close to each other (without having to call the user for assistance).

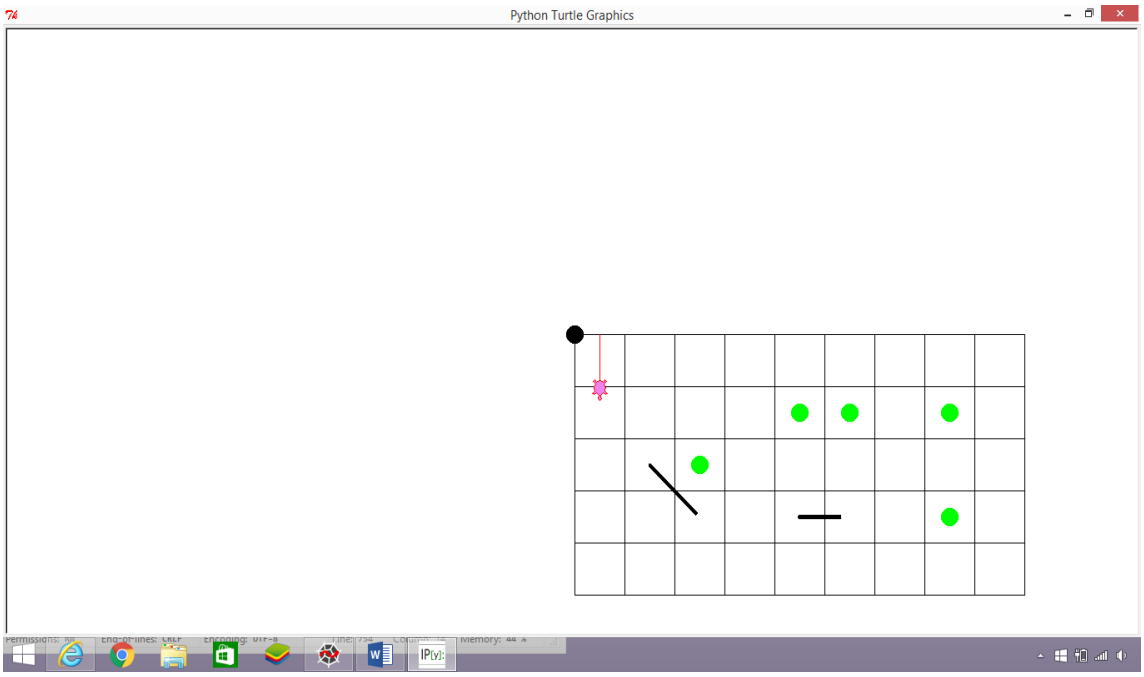

Figure 7.3.3.14. Simulation Environment

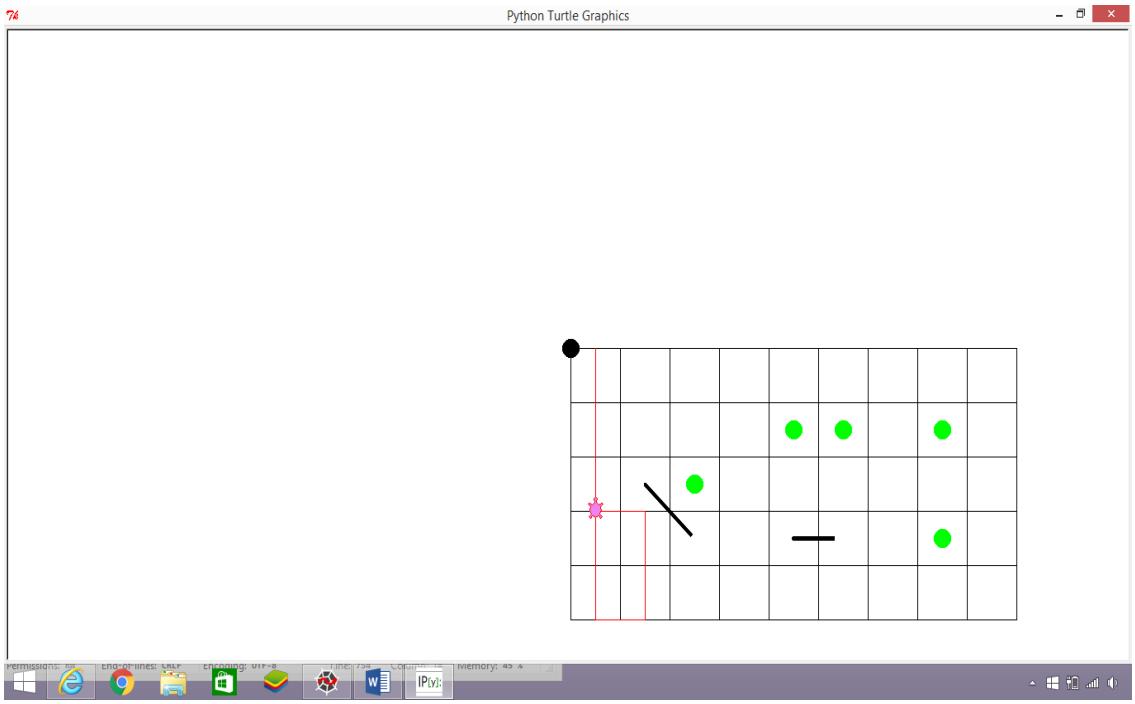

Figure 7.3.3.15. Vehicle navigating around the obstacle.

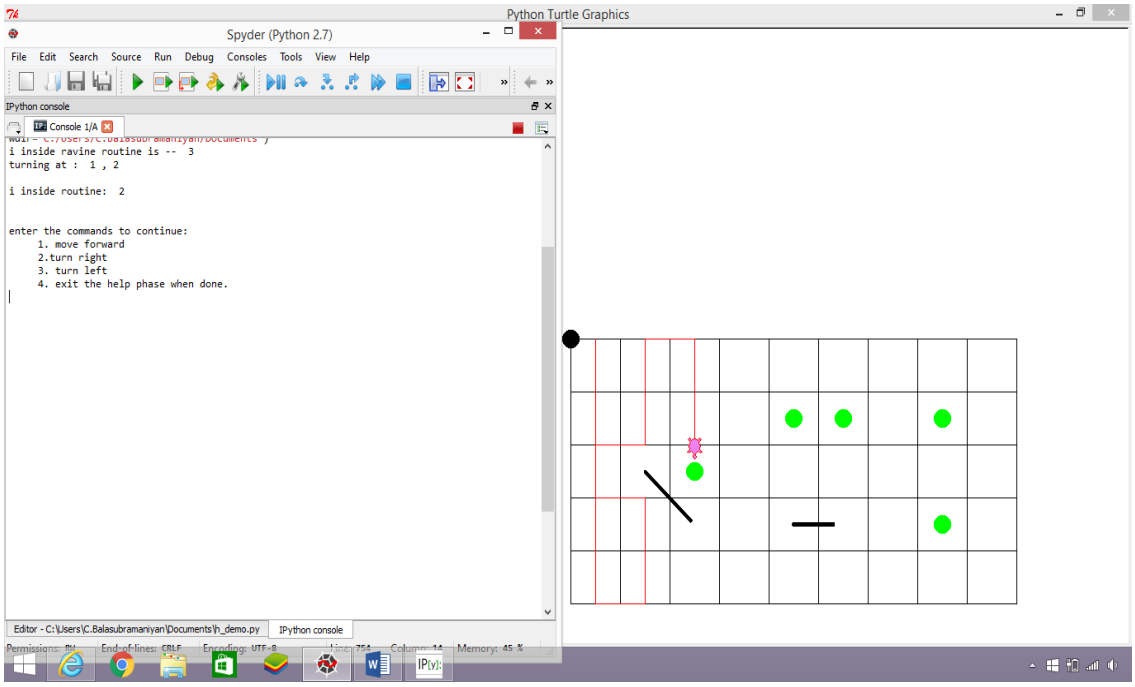

Figure 7.3.3.16. Vehicle seeks user's help.

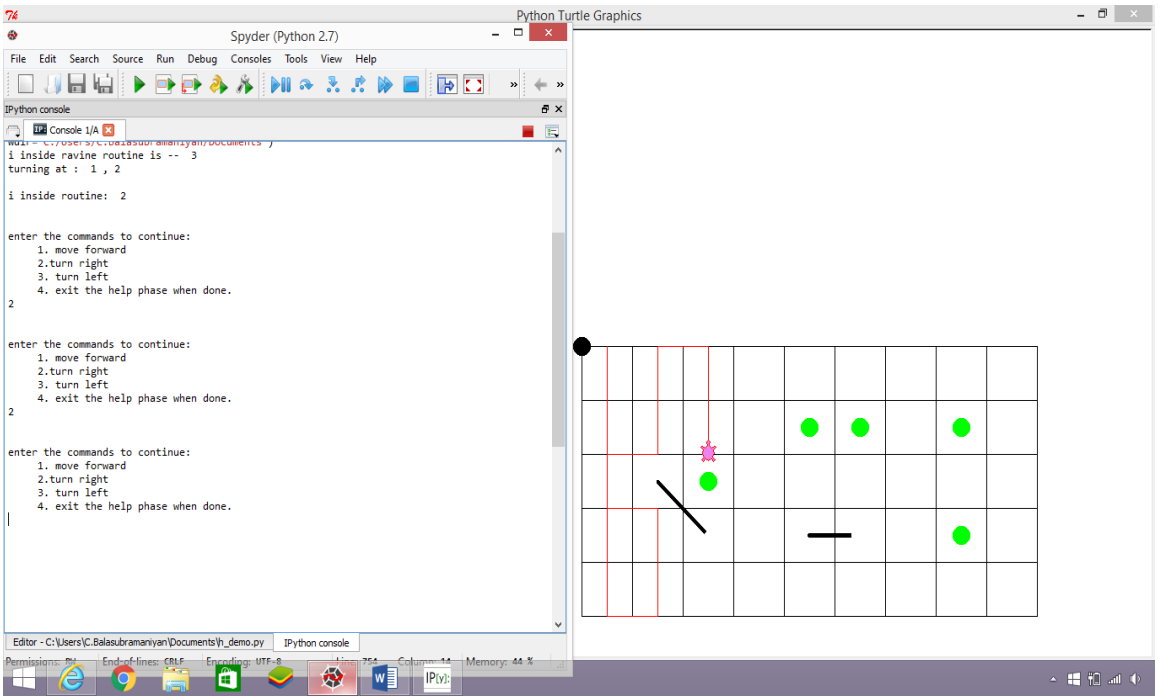

Figure 7.3.3.17. User enters turns the vehicle right twice to avoid the obstacle.

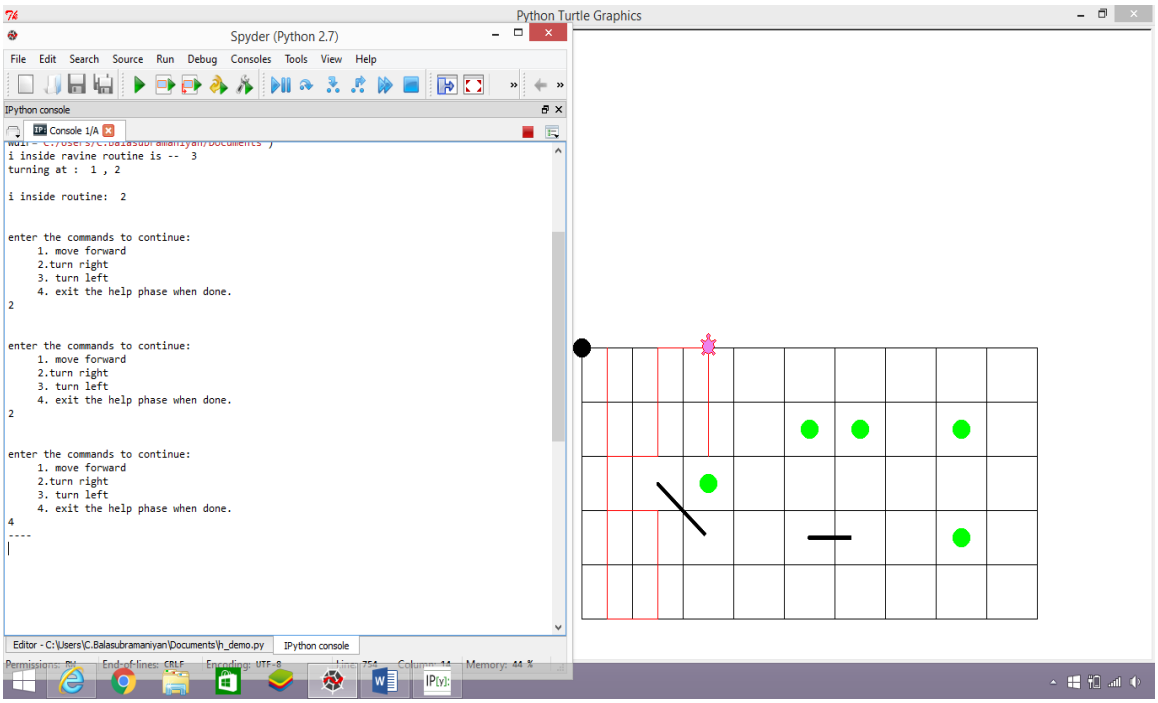

Figure 7.3.3.18. Vehicle switches to the autonomous mode.

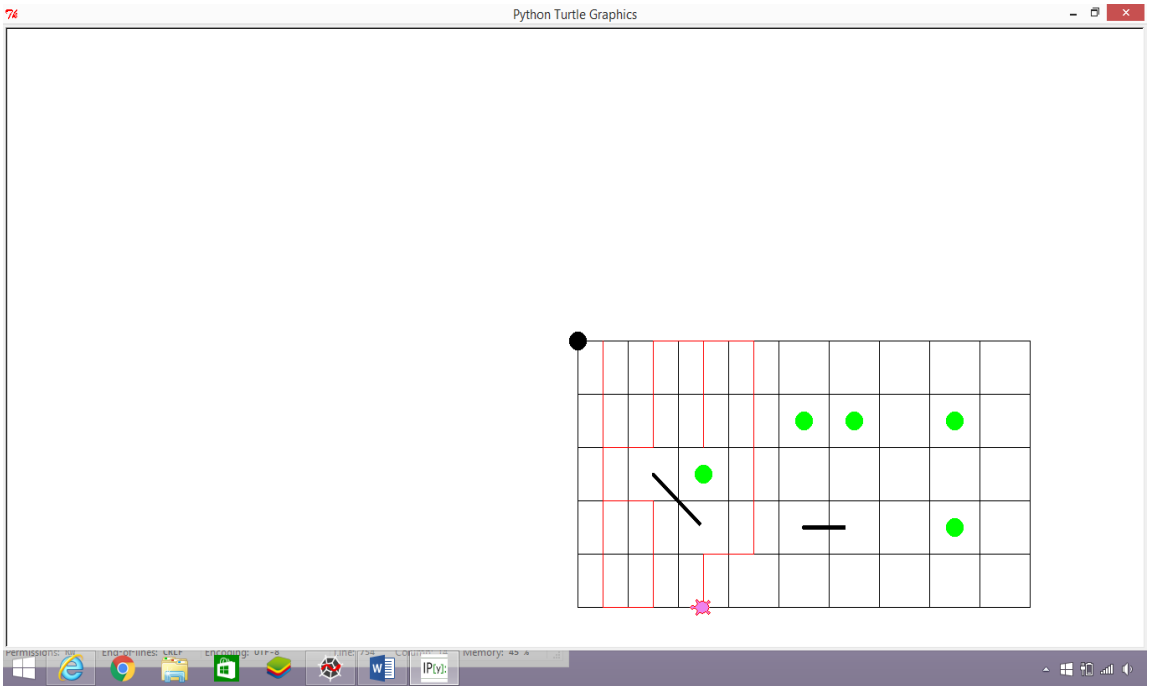

Figure 7.3.3.19. Covering cells that were previously not visited.

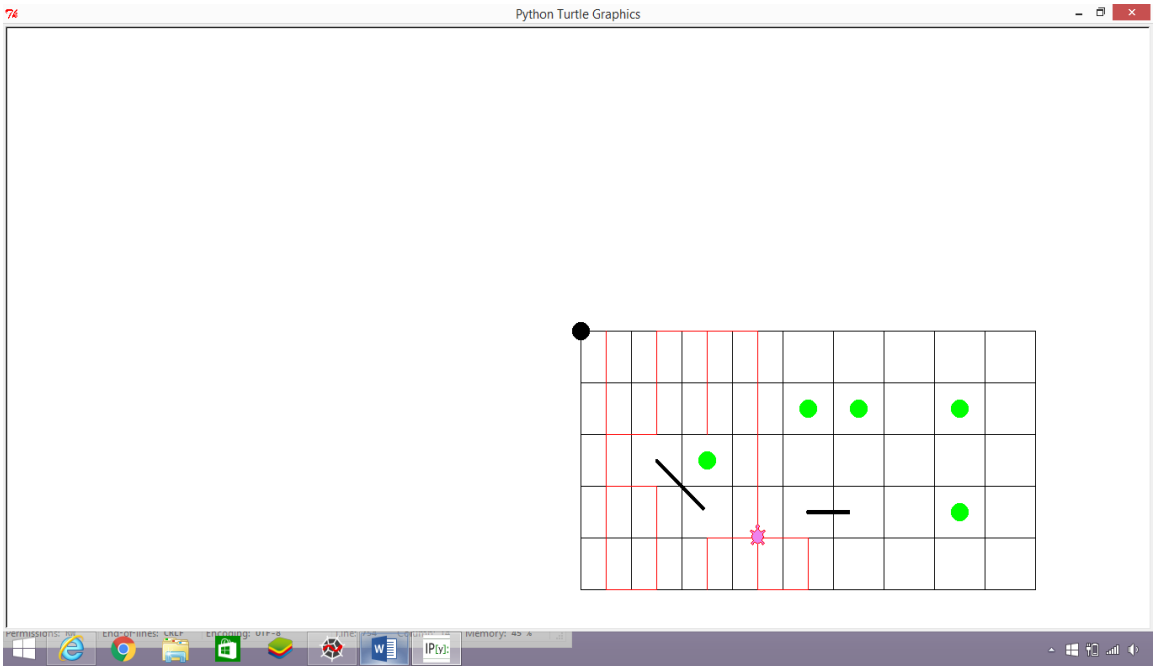

Figure 7.3.3.20. Vehicle navigating around the ravine.

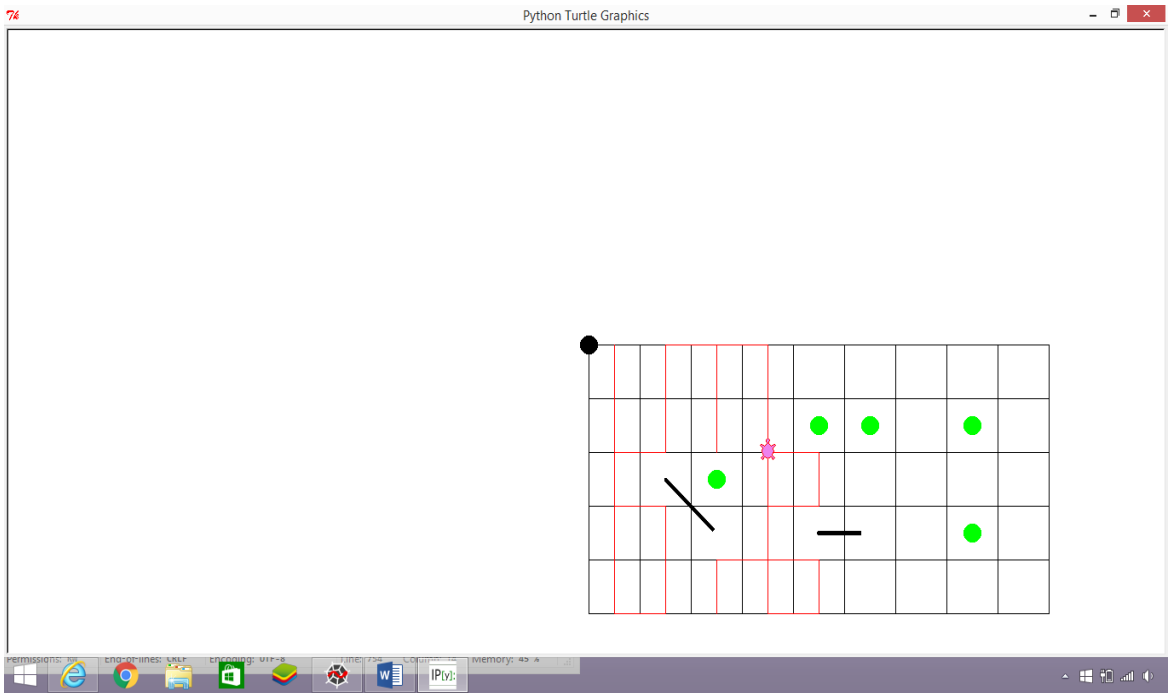

Figure 7.3.3.21. Vehicle navigating around the tree.

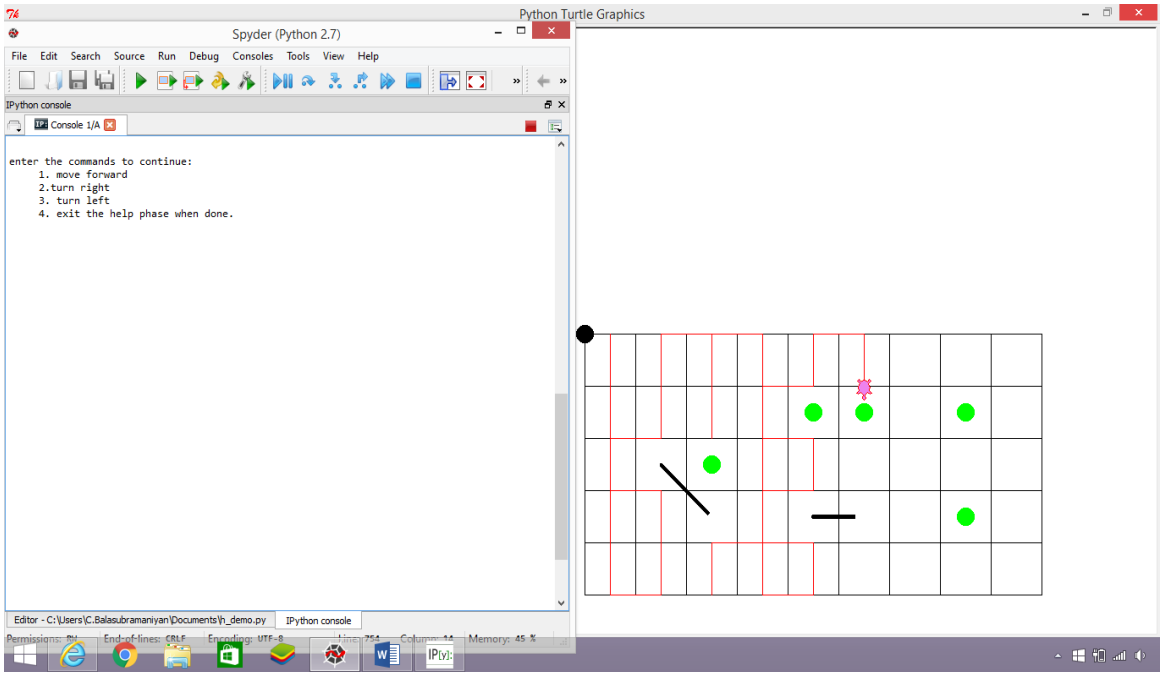

Figure 7.3.3.22. Vehicle enters into the help phase.

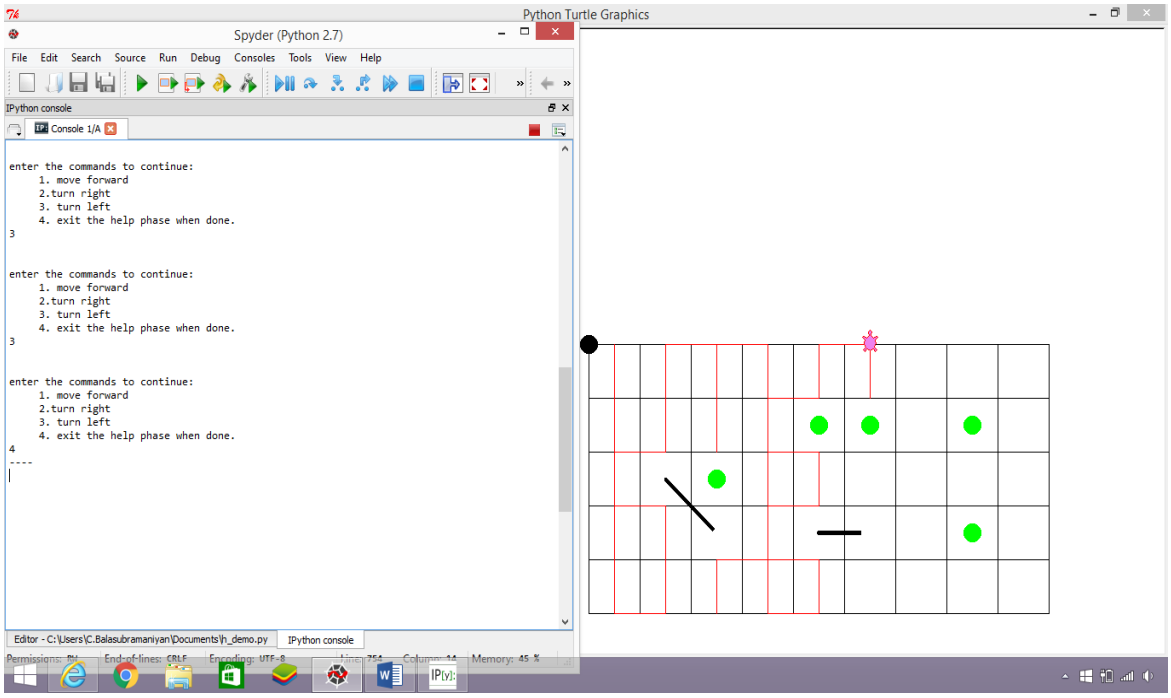

Figure 7.3.3.23. User enters minimum commands to steer the vehicle away from the obstacle

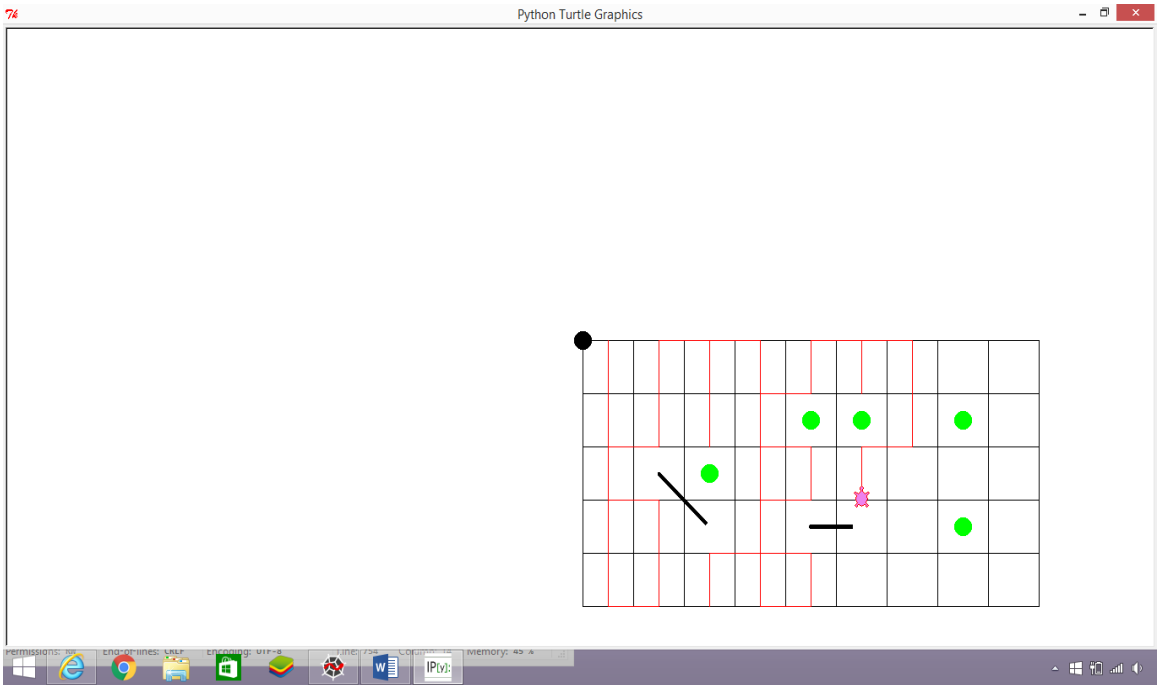

Figure 7.3.3.24. Vehicle handles the closely located tree and ravine without user's help.

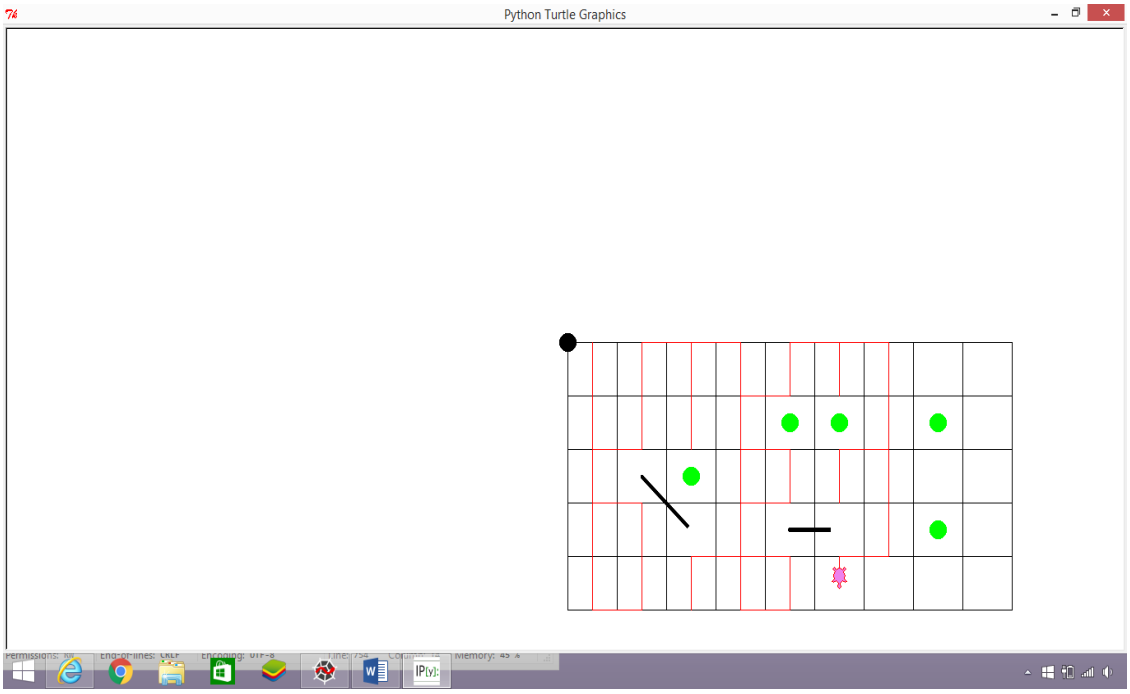

Figure 7.3.3.25. Vehicle covering cells not visited.

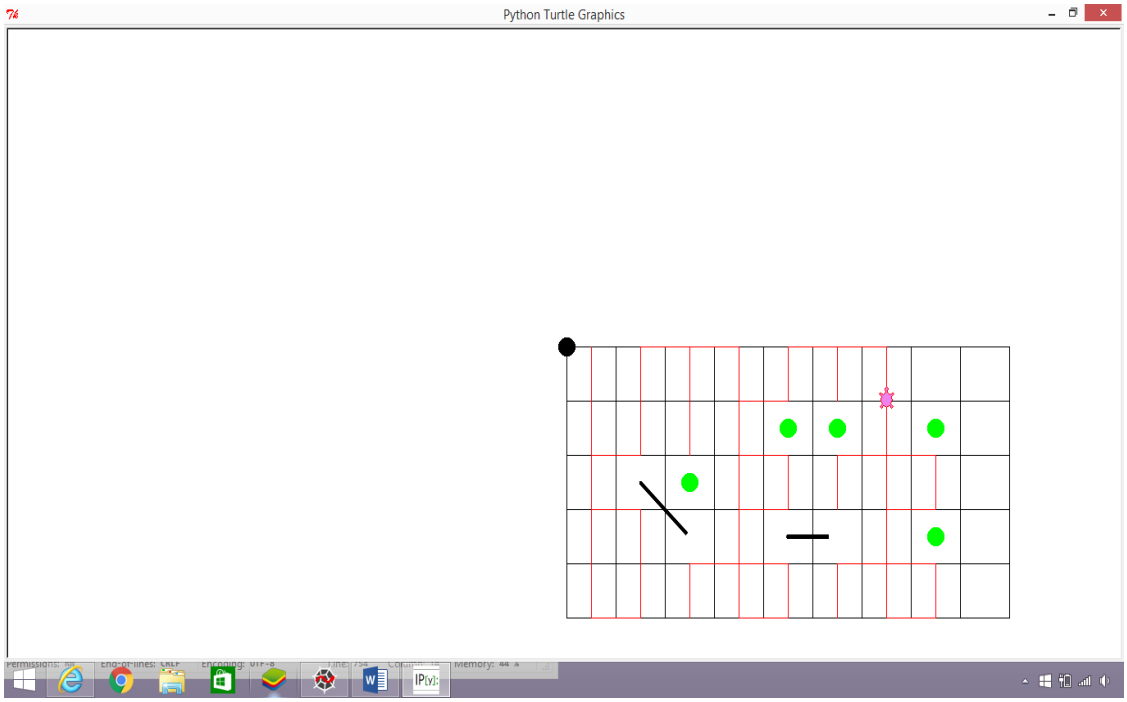

Figure 7.3.3.26. Vehicle navigating around the trees.

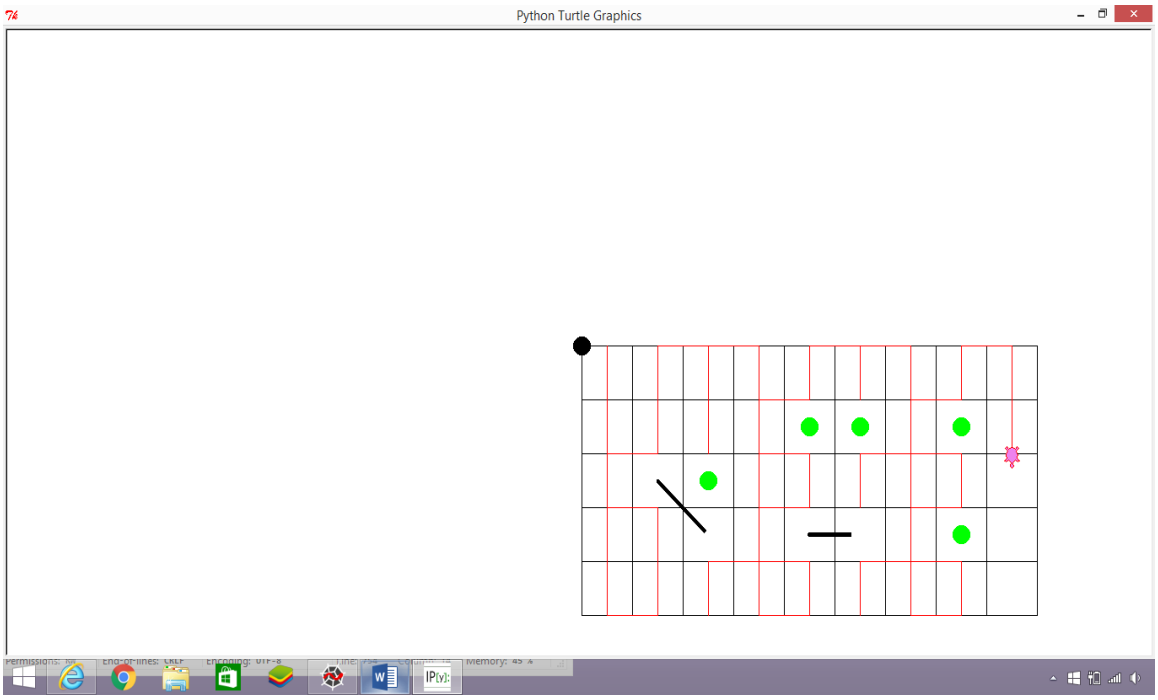

Figure 7.3.3.27. Vehicle follows normal navigation until its destination.

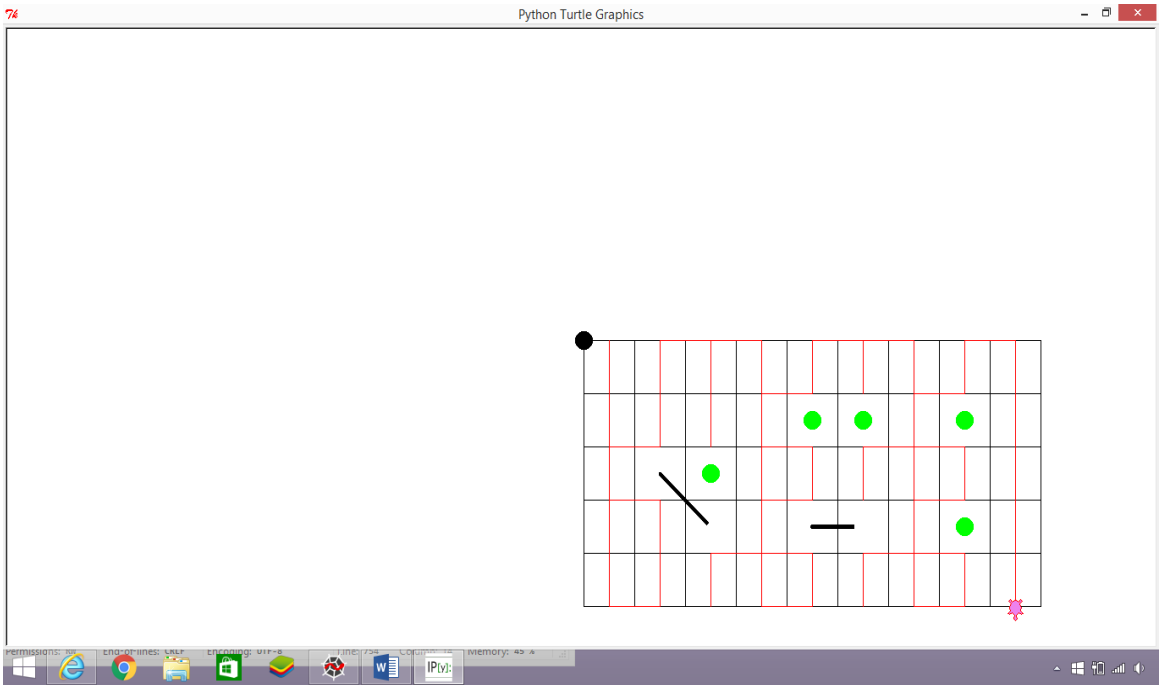

Figure 7.3.3.28. Vehicle at the destination.

CHAPTER 8: ROBOT COGNITIVE MODEL

In this chapter, we demonstrate the use of a robot sensor map similar to the sensory brain map [4] to perform information processing and resource management for sensors on a robot. The sensor map implemented works in a very similar manner as the sensory brain map, performing operations, such as - sensor slot allocation, allocation of un-used sensor space to the other sensors and re-allocation of sensor space back to the sensors which had the original ownership. The sensor manager is the controller of the sensor space and handles these operations through several sub-routines (or functions). There are three functions that do the following:

a) Sensor slot allocation
b) Allocating space from passive sensor slot to active sensor slots
c) Re-allocation of space to dormant sensor slots when they become active again

The sensors are assumed to be laid in a sequential manner in the sensor space. Depending on the initial conditions, the sensors could be originally assigned equally spaced memory slots. The dynamicity of the map is tested with the help of a navigation application. The application includes four tasks and has five sensors on the vehicle. The sensor functions are called depending on the availability of memory for the respective sensor. The sensor manager holds all the relevant information related to each sensor that is present on the vehicle.

The Sensor Manager holds the exclusive ownership of the sensor map space. It keeps track of which sensors are used for the application and periodically (or based on the user given requirements) can run different functions that manage and modify the space for the sensors.

Similar to the use-it-or-lose-it principle, as outlined and described in [4, 5], the sensory brain map is a dynamic structure that keeps changing depending on the usage of our sensory organs. The concept and results are similar to that described in [74].

8.1 Sensor Maps

8.1.1 Introduction and Theory

In [4, 5, 30], the authors describe the cognitive model of the brain through sensory brain maps. When a sensory input is not recognized in the brain map for certain time, the map dynamically alters itself to accommodate more space and resources for those sensor slots that are adjacent to the one that is marked as dormant. This results in a few sensory areas on the map becoming larger, growing into the space that was utilized by the dormant sensor while the other sensor areas show no change in their space (those not adjacent to the dormant sensor). However, when the dormant sensor becomes active again, through the respective task (training), the map again starts to alter, re-assigning its original space in the brain map, showing increase in space at that slot. This is known as differentiation of the brain map [4, 30].

These principles on which the brain operates and micro-manages the sensory map is applied in this work, and is demonstrated through the use of three functions namely, Sensor Space Reduce, Sensor Space Deallocate and Sensor Space Reallocate.

8.1.2 Machine Learning and Sensor Map Dynamicity

In order to illustrate the dynamic nature of the maps, it is necessary to use a learning technique. Since we as the user, can decide the criteria for memory usage and thresholds for the sensors, supervised classifiers are the best fit for this purpose. Each of the sensor space functions actively use a supervised classifier called the Random Forest.

Random Forest [95] belongs to the class of Ensemble learning, and can handle large data sets providing higher accuracy in their predictions. A detailed explanation of the advantages of Random Forests with other classifiers is described in [96, 97].

The user can choose the label for the data sets and define them during the training phase. However, for the present application the data sets are small hence the classifier is very reliable and accurate in its prediction on the test set.

8.1.3. Sensor Map / Sensor Space Management Functions

There are three functions used to manage the sensor space on the vehicle. The description of each of these functions is as below.

a) Sensor Space Reduce:

The purpose of this function is to identify the space that should be retained for each sensory input. This function keeps track of the usage of the sensors on the vehicle and determines how much space should be held in the map for each sensor. When a sensory input keeps reducing as the application progresses, or between different applications, the corresponding space in the map starts to dynamically

reduce as a function of its usage. The classifier uses the frequency of the sensor's usage to determine the percentage by which the space should be reduced. This phenomenon is akin to how the brain reduces the space on its sensory map when a particular sensory input is less and less received on the map during one's lifetime, also known as shrinkage of sensory space on the map.

b) Sensor Space Reallocate:

This function distributes the sensor space of a dormant sensor to its adjacent sensory members on the map. This function checks for sensors that are marked for deletion and identifies its adjacent members on the map. The classifier distributes the space to the dormant sensor's neighbors.

After space distribution, it notifies the sensor manager of these changes. As a result, the re-distribution causes some areas on the map to grow larger, however, when the sensors are at the borders, the entire distribution is for the one sensor that is adjacent to it.

c) Sensor Space Deallocate:

The purpose of this function is to de-allocate the space from those sensors that had previously taken space from the dormant sensor and re-allocate it back to the dormant sensor when it becomes active again. The function uses a classifier that identifies the sensors that become active again and automatically trains those sensors (this is done internally inside the function by running the tasks mapped to it) to increase its usage. Again, the usage metric causes the classifier to dynamically increase the space on the map for the trained sensor while reducing the extra space from its adjacent counterparts.

The use of these functions is illustrated through a navigation application of a vehicle on a field described in Chapter 7. The vehicle has five sensors on its sensory map. We assume that these sensors are laid out adjacently to each other, each have initially full space (denoted as 100%) on the map. The vehicle navigation on the field causes a few sensors to be used and a few to remain un-used. The usage of these sensory inputs determine how the functions will allocate the space on the map for each of them. The sections below describe the use of these learning functions in the robotic navigation application and the corresponding sensor space dynamicity is illustrated through the bar graphs.

8.2 Robotic Vehicle Navigation

The application is implemented in Python 2.7 [92] and uses libraries from sklearn documentation [98, 99] to make use of some machine learning techniques [25, 26] for supervised classification.

8.2.1 Problem Description and Experimental Set up

High level Task: Navigate an unknown terrain consisting of trees and ravines in the field.

Sub Tasks: Move forward, Turn right, Turn left and Check for person.

Sensors: Five sensors, sensors 0, 1 for Move forward, sensor 2 for Turn right, sensor 3 for Turn left and sensor 4 for identifying the person (camera).

Problem statement: To demonstrate the dynamic nature of the robot's sensory map as the application is executed and validate the usage of the three functions defined earlier.

Initial conditions: The vehicle is positioned at the origin of the grid. We assume a rectangular field and the robotic vehicle (represented as a turtle in the Figure 8.2.1) traverses the field (or a subspace in the field), by moving forward, turning right or left and detecting a person (performed internally in the vehicle, but not shown in figure).

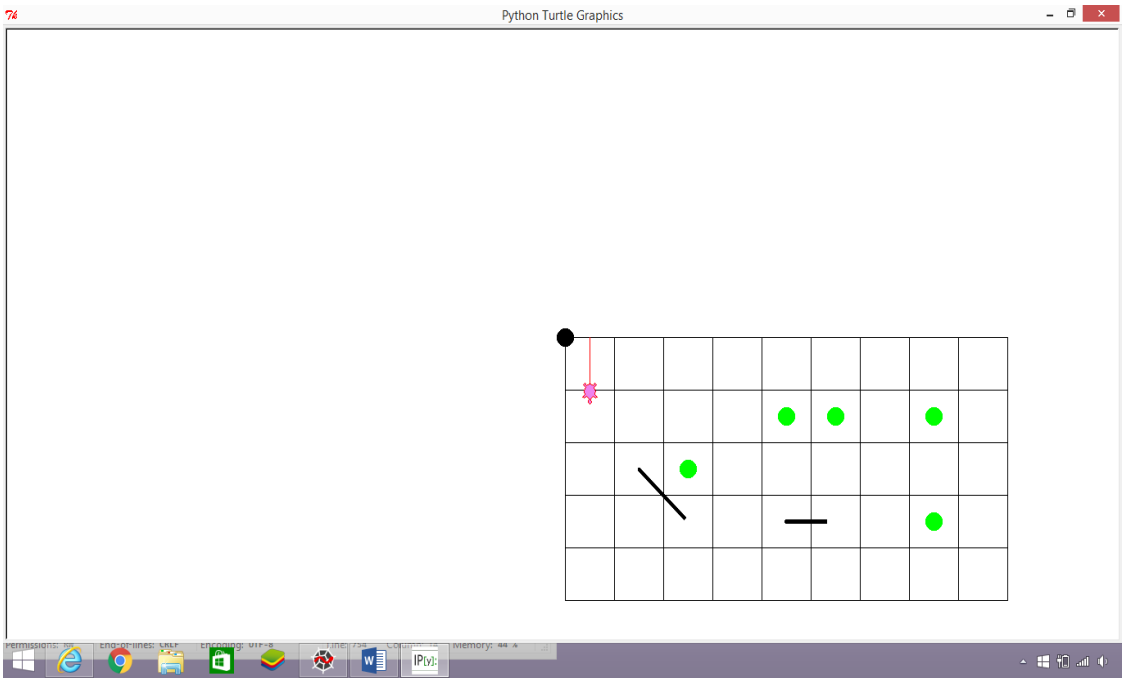

Figure 8.2.1 Robot navigation in a rectangular field.

The robot's sensory map is pre-filled with the memory allocation and sensor usage. We start off the experiment by assigning full space to each sensor slot and randomly assign sensors in the list with some usage value to illustrate how this concept works. Table 8.2.1 shows prefilled variables for the sensor slots.

Table 8.2.1. Initial state of the sensor map.

Sensor ID	Space available (in %)	Usage count
0	100	50
1	100	70
2	100	50
3	100	50
4	100	55

8.2.2 Application Execution and Sensor Map Dynamicity

The navigation application is executed and the vehicle updates the usage of the sub tasks and the sensors during this process. Once the field is navigated, the sensor manager runs the Sensor space dynamicity function. As a result, the space for each sensor starts to reduce depending on how much it was used during its previous application.

The classifier present inside this function does not reduce the sensor space if its usage is above a certain threshold (user-defined), else it reduces the sensor space according to its usage. We present two possible scenarios below on the cases of dynamicity.

Case 1: Dynamic allocation from a border sensor slot

Table 8.2.2.1 indicates the space available for each sensor and the sensor usage after the application was executed. Figure 8.2.2.1, shows the sensor space distribution before and after the application.

Table 8.2.2.1: Sensor map after space reduction.

Sensor ID	Space available (in %)	Usage count
0	100	121
1	100	141
2	80	82
3	65	66
4	55	55

From the table 8.2.2.1 and Figure 8.2.2.1, we can see that as the usage is below a certain threshold the classifier marks the amount of space to be reduced for each sensor. The blue and orange bars indicate the available space before and after the application. The user can specify the criteria for reduction depending on how often the task is being used. When a sensor space is reduced, it refers to the active working area only. For instance in the above table, we can see that space for sensor 2 was reduced by 20%, however this 20% is still under the ownership of sensor 2 (until it reaches 0).

After the space reduction phase, the Sensor manager calls the Sensor Reallocate function to identify the sensors with space reduced to 0. This function then deallocates the space from this sensor (sensor with space equal to 0) to its adjacent sensors. This causes some of the sensor spaces to be larger than others. In this experiment, we execute this function only when the space for a sensor has dropped to 0, indicating with certainty, that the sensor wasn't used in the previous application (its usage count is reduced to 0).

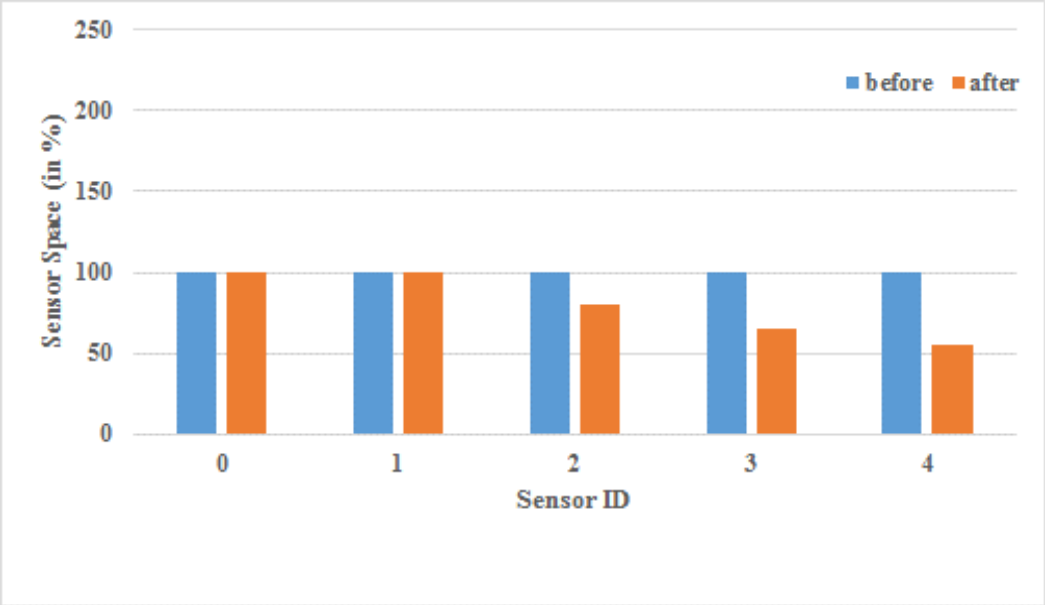

Figure 8.2.2.1 Sensor space distribution before and after the navigation application.

Table 8.2.2.2 shows the map distribution after this space reallocation. Figure 8.2.2.2, shows the space distribution from the passive sensor to its adjacent member sensors. Since there is only sensor 3 as a neighbor to sensor 4, all of sensor 4 space is allocated to sensor 3 causing it to grow larger which is shown in the Figure 8.2.2.2.

Table 8.2.2.2: Sensor map after sensor re-allocation.

Sensor ID	Space available (in %)	Usage count
0	100	121
1	100	141
2	100	82
3	200	66
4	0	0

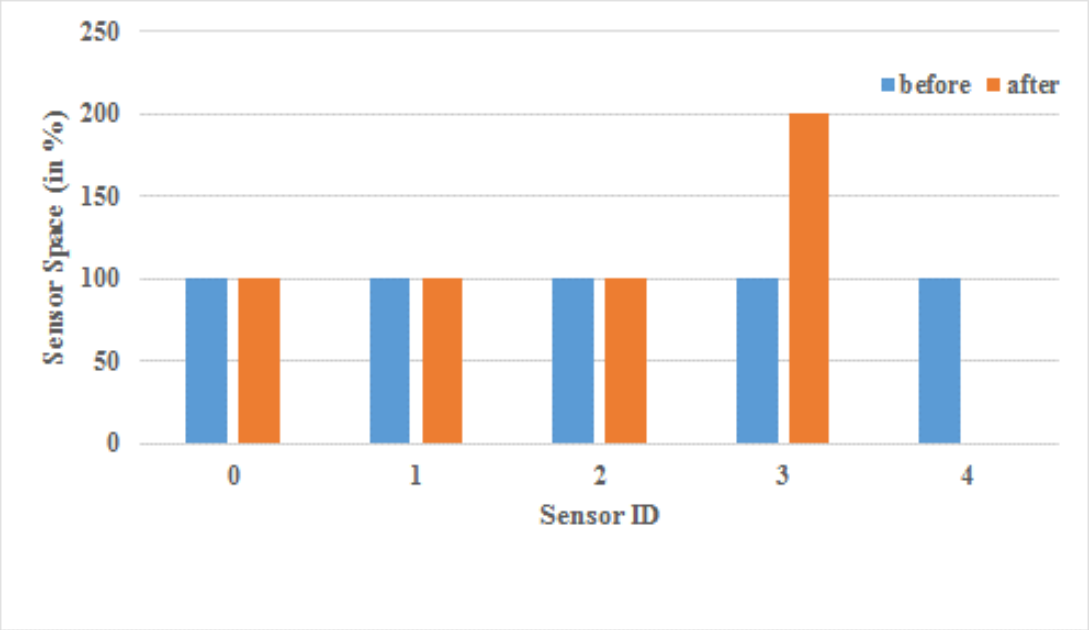

Figure 8.2.2.2 Space distribution after sensor de-allocation. Only sensor 3 gets the space from sensor 4 in this example.

We then execute the application again and re-enable the task mapped to sensor 4 (check for person). When this test application is called, it wants to use the sensor 4 for its task (detect person). Since the map had de-allocated sensor 4 space (since it wasn't used in the previous application) to sensor 3, it needs to re-assign that space back to sensor 4.

When the test application identifies that this sensor has no space, it calls another function, Handle Feasibility. This function identifies the sensor with the ID specified in the argument list and starts training the vehicle internally to increase its usage thereby allocating space back into its area and simultaneously de-allocating

the space from its adjacencies. The feasibility function uses the Sensor De-allocate function internally to achieve this task. As a result, the sensor space is reduced for a few sensors as shown in the Table 8.2.2.3 and Figure 8.2.2.3.

Sensor 3, which showed an increase in its space has now shrunk and sensor 4 is re-allocated its space to be used in the application. The sensor manger, if necessary can now execute the sensor space reduction as described previously, to reduce the space depending on the usage in the application.

Table 8.2.2.3: Sensor map after test application.

Sensor ID	Space available (in %)	Usage count
0	100	184
1	100	204
2	100	100
3	108	84
4	92	63

Case 2: Dynamic allocation from a sensor slot in between

Table 8.2.2.4 (similar to Table 8.2.2.2) indicates the space available for each sensor and the sensor usage after the application was executed. Figure 8.2.2.4, shows the Sensor space distribution before and after the application.

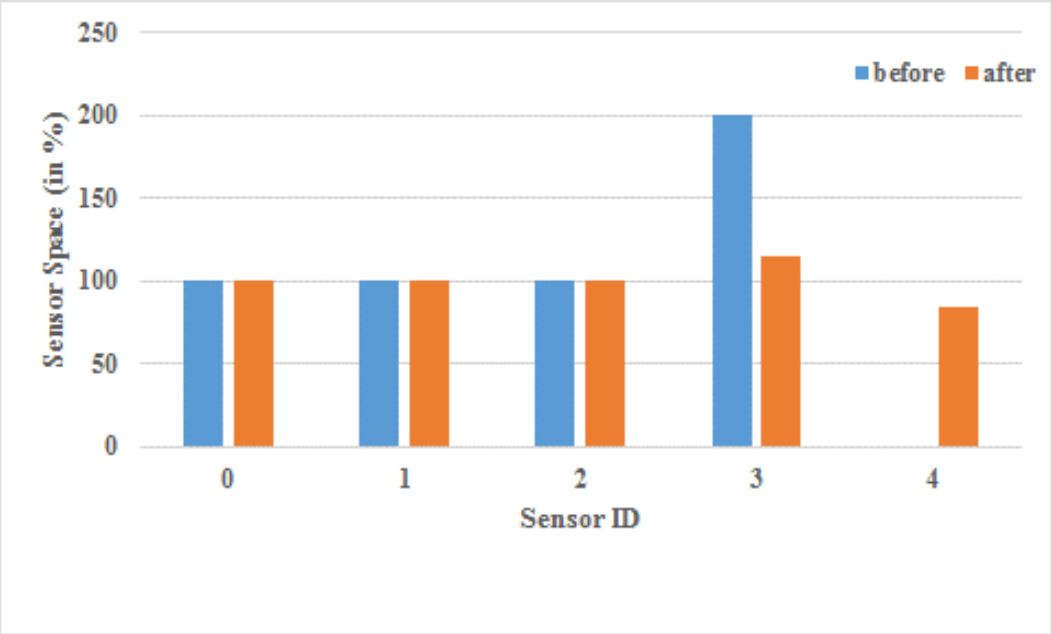

Figure 8.2.2.3 Space distribution after test application. Sensor 4 becomes active and regains the space it had sacrificed to sensor 3 previously due its dormancy.

Table 8.2.2.4: Sensor map after space reduction.

Sensor ID	Space available (in %)	Usage count
0	100	121
1	100	141
2	80	82
3	65	66
4	55	55

From the table and graph 8.2.2.4, we can see that as the usage is below a certain threshold the classifier marks the amount of space to be reduced for each

sensor. The blue and orange bars indicate the space before and after the application. The user can specify the criteria for reduction depending on how often the application was being executed.

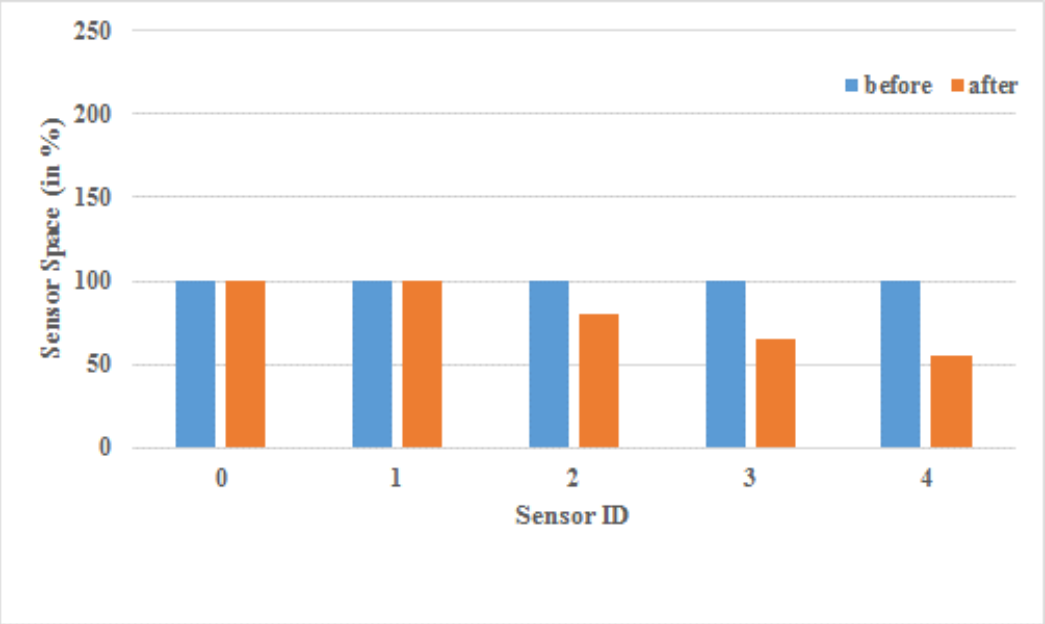

Figure 8.2.2.4 Sensor space distribution before and after the navigation.

After the space reduction phase, the sensor manager calls the Sensor Re-allocate function to identify the sensors with space reduced to 0. This function then de-allocates the space from this sensor (sensor with space equal to 0) to its adjacent sensors. This causes some of the sensor spaces to be larger than others.

In this experiment, we execute this function only when the space for a sensor

has dropped to 0, indicating with certainty that the sensor wasn't used in the previous application (its usage count is reduced to 0). Table 8.2.2.5 shows the map distribution after this space re-allocation.

Figure 8.2.2.5 shows the space distribution from the passive sensor to its adjacent member sensors. For the purpose of demonstration we have disabled task 3 that results in sensor 3 becoming unused and the usage dropping to 0 and space reduced to 0. On executing the Sensor space re-allocate function, we get the Table 8.2.2.5. Since sensors 2 and 4 are adjacent to sensor 3 its space is distributed to 2 and 4 causing their areas to grow larger which is evident from the Figure 8.2.2.5.

Table 8.2.2.5: Sensor map after sensor re-allocation.

Sensor ID	Space available (in %)	Usage count
0	100	126
1	100	146
2	135	82
3	0	5
4	155	59

We then execute the application again and re-enable the task mapped to sensor 3. Since the map had de-allocated sensor 3 space (it was forcefully reset for testing) to sensors 2 and 4, it needs to re-allocate that space back to sensor 3. When the test application identifies that this sensor has no space it calls another function, Handle Feasibility. This function identifies the sensor with the ID specified in its argument list and starts training the vehicle internally, to increase its usage thereby allocating space back into its area, simultaneously de-allocating the space from its

adjacencies. The feasibility function uses the Sensor De-allocate function to achieve this task. As a result, the sensor space is reduced for a few sensors (2 and 4) as shown in the Table 8.2.2.6 and Figure 8.2.2.6.

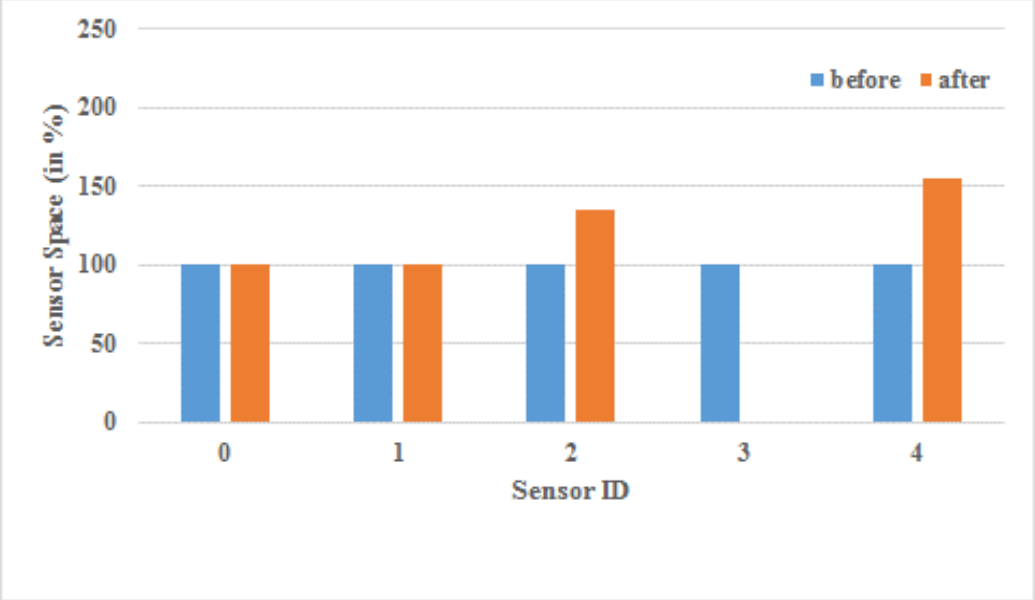

Figure 8.2.2.5 Space distribution showing Sensors 2 and 4 acquiring sensor 3 space.

Sensors 2 and 4, which showed an increase in its space has now shrunk and sensor 3 is re-allocated its space to be used in the application. This process is also called Differentiation of the brain map [4, 5]. The sensor manger, if necessary can now execute the sensor space reduction as described previously, to reduce the space depending on the usage in the application.

Table 8.2.2.6: Sensor map after test application (de-allocate function).

Sensor ID	Space available (in %)	Usage count
0	100	184
1	100	204
2	106	104
3	58	83
4	126	83

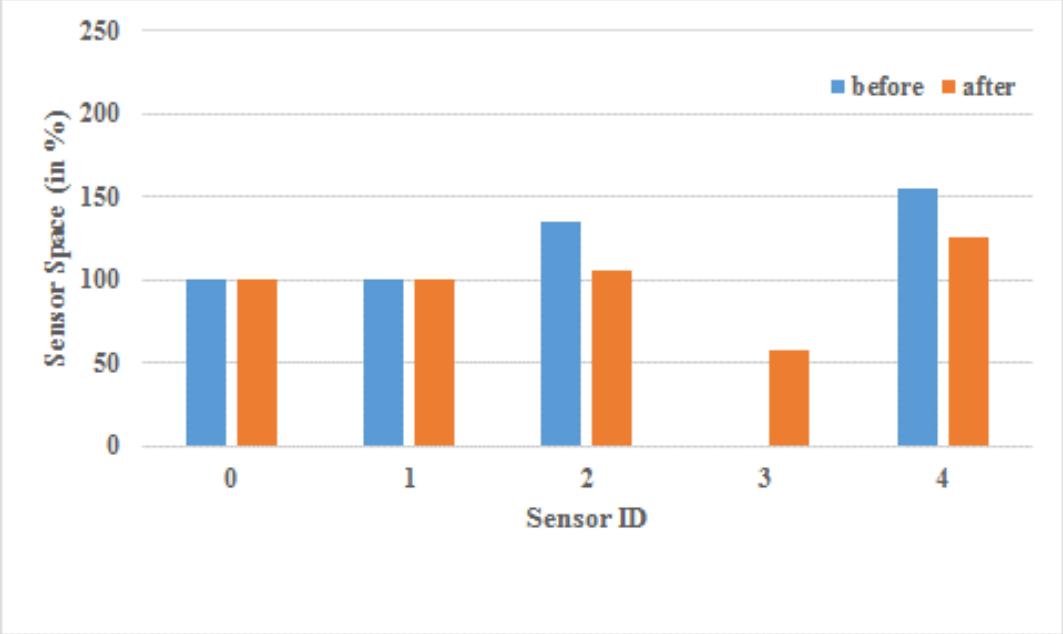

Figure 8.2.2.6 Space distribution after test application. Sensor 3 becomes active and regains the space it had sacrificed to sensors 2 and 4.

CHAPTER 9: C O N T R I B U T I O N , CONCLUSION AND FUTURE WORK

9.1 Contribution

In this research work, we have discussed the background research on Human-robot interaction (HRI), the various application of HRI in our society, multi sensor fusion types and architectures, machine learning, path planning, sensor fusion framework for human robot interaction, sensor management architectures and brain sensor maps. The research proposes a sensor fusion framework that encapsulates each of the topics through different entities to achieve decision fusion for a high level task while increasing confidence through the stages and in the process use a cognitive model based on the brain map platform to achieve human-like sensor data processing.

The sensor framework applies heterogeneous sensor fusion to achieve feature extraction and uses the features thus realized to formulate a decision. The idea that only the present evidence is necessary to decide the cell reachability and the inclusion of uncertainty makes Dempster-Shafer a good choice for our decision fusion. By measuring attributes of different characteristics and types, i.e., trees/blocks and ramps/ravines, we incorporate sensor fusion across heterogeneous sensors. Achieving high level task execution through heterogeneous sensor fusion and using a decision fusion approach close to human reasoning.

The main goal of this research was to formulate a sensor framework that uses sensor fusion and sensor maps to achieve robotic vehicle navigation application. The navigation application implemented covers maximum area of the field and invokes human involvement through a help phase. This increases the reliability of the system by converting the indecisive situations into an opportunity for collaborative control, thus reducing the possible collision of the vehicle with the obstacle. The cognition model for the robot is a sensory map that handle dynamic sensor management and resource management. This is a novel approach to handle sensor information on the robotic vehicle through concepts of neuro science to utilize sensor space effectively and simulate human-brain like model for robot cognition. The sensor maps is a concept introduced to define robot cognition that is similar to the brain maps. The research work does not focus on comparing the different cognition models that are used in AI or related fields. The dynamicity of the maps refers to both the sensor maps used in cognition and the environment map where the vehicle navigates. The number of sensors on the vehicle can change and hence the space for the sensors in the map also changes between applications thus making the sensor area dynamic. This is useful in resource management of sensors (and sensor space).

We were able to see from the navigation application that the vehicle covers more area on the field and invokes user's assistance only when absolutely necessary thus increasing autonomy with human interaction as described in Chapter 6 and 7. The sensors demonstrated increase and decrease in their space and their behavior is dictated by their location on the map. The experiments presented through Case 1 and Case 2 under Section 8.2.2 describe the dynamicity of the maps that is parallel to our

brain sensory maps.

The sensor fusion part of the framework is being implemented in C++ and uses ROS to simulate the LIDAR sensors. There are two main types of features that will be extracted by the LIDARs, namely, trees (by the horizontal LIDAR) and ravines (by the 45 degree tilted LIDAR). These are two sensors that measure the condition of the environment. There is no path planning algorithm that the vehicle executes but a column-by-column navigation covering as many cells as possible is the main goal of the robotic navigation.

The sensor fusion works at three levels, signal fusion using Kalman filter, feature extraction from the signal fusion result and finally using features to identify the nature of the obstacle and the reachability of the cell. Each fusion cell will carry information about the obstacle (or feature) through two main variables, an indicator flag and a confidence value that is dependent on the indicator. Human involvement is mandatory to provide more information i.e., soft data to the fusion algorithm. The framework incorporates several internal layers of sub-decisions before executing the final decision. Using the ROS helps to simulate the sensors and handle the physical layer of the sensor fusion framework and the robotic vehicle testing platform is on the turtle bot vehicle which runs on the Linux operating system.

The framework is centralized in design and thus requires evaluating the computation load on the central computer that runs the framework. The most computation intensive part of the framework was the track to track fusion block and the performance was calculated bot with ROS and without ROS. There was significant increase in the performance bottleneck while using ROS because of the

several dependencies and processing due to ROS. The operating system used is Linux and by using the Linux tool, perf, we can measure the computation performance of the system. The computation performance could also help identify the bounds of the sensor inputs for the framework. Since ROS doesn't allow dynamic allocation of sensor memory the concept of sensor maps was executed on a simple robotic navigation application created using python and run offline instead of real time inside the framework. A detailed video showing the framework working with all test cases can be found in [100] that captures all the contributions of this research work. Identifying the scalability is dependent on the application and the computation load on the central node. Since sensor fusion frameworks are hard to compare [54, 55, 56] and is highly dependent upon the application and the system configuration we can conclude that the maximum number of sensors can be the upper limit of performance of the node and the fusion efficiency.

9.2 Application of the framework

The framework combines the data from sensors to achieve heterogeneous fusion. Zapatabot, an autonomous robotic All Terrain Vehicle (ATV) that can navigate unknown terrains has several sensors such as LIDARs, camera etc. A detailed explanation of the Zapatabot and its features is described in [101, 102]. Figure 9.2 [101] shows the Zapatabot. The sensor fusion framework that was described will find its application on this vehicle that hosts many sensors and navigates rough terrains.

Figure 9.2 Zapatabot - An Autonomous Robotic All Terrain Vehicle (ATV)

9.3 Future Work

The framework can be extended to implement other robotic applications. The significant changes would require changing the sensors and their positions on the vehicle. Also, using Gazebo [82], different vehicle models exist that can be used in place of the vehicle that was customized for this research work. Since the system is scalable, more features can be added to the patter identifier and correspondingly including those features at all stages of the framework.

REFERENCES

[1] Chandrasekaran, B., & Conrad, J. M. Human-robot collaboration: A survey. In SoutheastCon 2015 (pp. 1-8). IEEE, 2015

[2] Pravia, M. A., Prasanth, R. K., Arambel, P. O., Sidner, C., & Chong, C. Y. Generation of a fundamental data set for hard/soft information fusion. In Information Fusion, 2008 11th International Conference on (pp. 1-8). IEEE, June 2008

[3] Hall, D. L., & Llinas, J., An introduction to multisensor data fusion. Proceedings of the IEEE, 85(1), 6-23, 1997

[4] Ramachandran, V.S., The Tell-Tale Brain: A Neuroscientist's Quest for What Makes Us Human, New York, NY:W. W. Norton & Company, 2011

[5] Doidge, N., The Brain's Way of Healing: Remarkable Discoveries and Recoveries from the Frontiers of Neuroplasticity, New York, NY: Penguin Publishing Group, 2015

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

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

[8] Vincze, M., Zagler, W., Lammer, L., Weiss, A., Huber, A., Fischinger, D., Gisinger and C. Towards a Robot for Supporting Older People to Stay Longer Independent at Home. ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of, pp. 1-7, 2014

[9] Keren, G., Ben-David, A., & Fridin, M. Kindergarten assistive robotics (KAR) as a tool for spatial cognition development in pre-school education. In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on (pp. 1084-1089). IEEE, October 2012

[10] Rani, P., Sarkar, N., Smith, C. A., & Kirby, L. D. Anxiety detecting robotic system–towards implicit human-robot collaboration. Robotica, 22(01), 85-95, 2004

[11] Jacob, M. G., Li, Y. T., & Wachs, J. P. Gestonurse: a multimodal robotic scrub nurse. In Proceedings of the seventh annual ACM/IEEE international conference on Human-

Robot Interaction (pp. 153-154). ACM, March 2012

[12] Ricks, D. J., & Colton, M. B., Trends and considerations in robot-assisted autism therapy. In Robotics and Automation (ICRA), 2010 IEEE International Conference on (pp. 4354-4359). IEEE, May 2010

[13] Wögerer, C., Bauer, H., Rooker, M., Ebenhofer, G., Rovetta, A., Robertson, N., & Pichler, A. LOCOBOT-low cost toolkit for building robot co-workers in assembly lines. In Intelligent Robotics and Applications (pp. 449-459). Springer Berlin Heidelberg, 2012

[14] Szafir, D., Mutlu, B., & Fong, T. Communication of intent in assistive free flyers. In Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction (pp. 358-365). ACM, March 2014

[15] Levinger, J., Hofmann, A., & Theobald, D., Semi-autonomous Control of an Emergency Response Robot. In AUVSI Unmanned Systems North America Conference, San Diego, CA (pp. 914-927), 2008

[16] Dasarathy, B. V., Sensor fusion potential exploitation-innovative architectures and illustrative applications. Proceedings of the IEEE, 85(1), 24-38, 1997

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

[18] Buede, D. M., & Girardi, P., A target identification comparison of Bayesian and Dempster-Shafer multisensor fusion. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 27(5), 569-577, 1997

[19] Koks, D., & Challa, S., An introduction to Bayesian and Dempster-Shafer data fusion, 2003

[20] Ghangrekar, S., & Conrad, J. M., Modeling and simulating a path planning and obstacle avoidance algorithm for an autonomous robotic vehicle. In 2009 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (pp. 1-3). IEEE, 2009

[21] Mitchell, H. B., Sensor Management. In Data Fusion: Concepts and Ideas (pp. 323-331). Springer Berlin Heidelberg, 2012

[22] Liggins II, M., Hall, D., & Llinas, J. (Eds.). (2017). Handbook of multisensor data fusion: theory and practice. CRC press

[23] Mitchell, H. B., Data fusion: concepts and ideas. Springer Science & Business Media, 2012

[24] Russell, S. J., Norvig, P., Canny, J. F., Malik, J. M., & Edwards, D. D., Artificial intelligence: a modern approach (Vol. 2). Upper Saddle River: Prentice hall, 2003

[25] Patnaik, S., Robot Cognition and Navigation: An Experiment with Mobile Robots. Springer Science & Business Media, 2007

[26] Mitchell, T. M., Machine learning. *Burr Ridge, IL: McGraw Hill*, 1995

[27] Carbonell, J. G., Michalski, R. S., & Mitchell, T. M., An overview of machine learning. In *Machine learning* (pp. 3-23). Springer Berlin Heidelberg, 1983

[28] Michalski, R.S., A theory and methodology of inductive learning, Artificial Intelligence, vol. 20, no. 2, pp. 111–161, 1983

[29] Watkins, C. J., & Dayan, P., Q-learning. Machine learning, 8(3-4), 279-292, 1992

[30] Doidge, N., The Brain That Changes Itself: Stories of Personal Triumph from the Frontiers of Brain Science (James H. Silberman Books), 2007

[31] Iyengar, S. S., & Brooks, R. R. (Eds.)., Distributed sensor networks: sensor networking and applications. CRC press, 2016

[32] Joshi, R., & Sanderson, A. C. Multisensor fusion: A minimal representation framework (Vol. 11). World Scientific, 1999

[33] Luo, R. C., & Lai, C. C. Multisensor fusion-based concurrent environment mapping and moving object detection for intelligent service robotics. *IEEE Transactions on Industrial Electronics*, *61*(8), 4043-4051, 2014

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

[35] Matía, F., & Jiménez, A. Multisensor fusion: an autonomous mobile robot. Journal of Intelligent and robotic systems, 22(2), 129-141, 1998

[36] Khaleghi, B., Khamis, A., Karray, F. O., & Razavi, S. N. Multisensor data fusion: A review of the state-of-the-art. Information Fusion, 14(1), 28-44, 2013

[37] Dallil, A., Oussalah, M., & Ouldali, A. Sensor fusion and target tracking using evidential data association. IEEE sensors journal, 13(1), 285-293, 2013

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

[39] Banerjee, T. P., & Das, S. Multi-sensor data fusion using support vector machine for motor fault detection. Information Sciences, 217, 96-107, 2012

[40] James, A. P., & Dasarathy, B. V. Medical image fusion: A survey of the state of the art. Information Fusion, 19, 4-19, 2014

[41] El Faouzi, N. E., Leung, H., & Kurian, A. Data fusion in intelligent transportation systems: Progress and challenges–A survey. Information Fusion, 12(1), 4-10, 2011

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

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

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

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

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

[47] Xiong, N., & Svensson, P. Multi-sensor management for information fusion: issues and approaches. Information fusion, 3(2), 163-186, 2002

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

[49] Durrant-Whyte, H. F. Sensor models and multisensor integration. The international journal of robotics research, 7(6), 97-113, 1988

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

[51] Dasarathy, B. V. Decision fusion (Vol. 1994). Los Alamitos, CA: IEEE Computer Society Press, 1994

[52] Dasarathy, B. V. Decision fusion strategies in multi sensor environments. IEEE transactions on systems, man, and cybernetics, 21(5), 1140-1154, 1991

[53] Wald, L. Data fusion: definitions and architectures: fusion of images of different spatial resolutions. Presses des MINES, 2002

[54] Hall, D. L., & Llinas, J. Multisensor data fusion. In Multisensor Data Fusion. CRC press, 2001

[55] Elmenreich, W. A review on system architectures for sensor fusion applications. In IFIP International Workshop on Software Technologies for Embedded and Ubiquitous Systems (pp. 547-559). Springer Berlin Heidelberg, 2007

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

[57] Kam, M, Zhu, X. and Kalata, P. Sensor fusion for mobile robot navigation. Proceedings of the IEEE, 85(1):108-119, Jan. 1997

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

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

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

[61] Alami, R., Chatila, R., Fleury, S., Ghallab, M. and Ingrand, F. An architecture for autonomy. International Journal of Robotics Research, 17(4):315-337, April 1998

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

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

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

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

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

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

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

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

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

[71] Munz, M., Dietmayer, K., & Mählisch, M. Generalized fusion of heterogeneous sensor measurements for multi target tracking. In Information Fusion (FUSION), 2010 13th Conference on (pp. 1-8). IEEE, 2010

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

[73] Chandrasekaran, B., & Conrad, J. M. Sensor fusion using a selective sensor framework to achieve decision and task execution. In SoutheastCon 2016 (pp. 1-7). IEEE, 2016

[74] Chandrasekaran, B., & Conrad, J. M. Complete Coverage Planning: Achieving Human Interaction and Maximum Coverage During an Autonomous Robotic Vehicle Navigation of an Unknown Terrain. In Workshops at the Thirty First AAAI Conference on Artificial Intelligence, 2017

[75] Wu, H., Siegel, M., Stiefelhagen, R., & Yang, J. Sensor fusion using Dempster-Shafer theory [for context-aware HCI]. In Instrumentation and Measurement Technology Conference, 2002. IMTC/2002. Proceedings of the 19th IEEE (Vol. 1, pp. 7-12). IEEE, 2002

[76] Challa, S., & Koks, D. Bayesian and dempster-shafer fusion. Sadhana, 29(2), 145-174, 2004

[77] Chandrasekaran, B., Gangadhar, S., & Conrad, J. M. A Survey of Multisensor Fusion Techniques, Architectures and Methodologies. In SoutheastCon, 2017 (pp. 1-8). IEEE, 2017

[78] O'Kane, J. M., A gentle introduction to ROS., 2014

[79] http://www.ros.org/

[80] http://wiki.ros.org/

[81] http://answers.ros.org/questions/

[82] http://gazebosim.org/

[83] http://www.turtlebot.com/

[84] http://oceanservice.noaa.gov/facts/lidar.html

[85] Fong, T., & Thorpe, C. Vehicle teleoperation with collaborative control. In Multi-Robot Systems: From Swarms to Intelligent Automata: Proceedings from the NRL Workshop on Multi-Robot Systems (p. 195). Springer Science & Business Media, November 2013

[86] Rubo, Z., Guochang, G., & Guoyin, Z. AUV obstacle-avoidance based on information fusion of multi-sensors. In Intelligent Processing Systems, 1997. ICIPS'97. 1997 IEEE International Conference on (Vol. 2, pp. 1381-1384). IEEE, 1997

[87] Scholtz, J. Evaluation methods for human-system performance of intelligent systems. NATIONAL INST OF STANDARDS AND TECHNOLOGY GAITHERSBURG MD MANUFACTURING ENGINEERING LAB, 2002

[88] Feil-Seifer, D., & Matarić, M. J. Human-robot interaction (hri) interaction human robot. In Encyclopedia of complexity and systems science (pp. 4643-4659). Springer New York, 2009

[89] Yanco, H. A., & Drury, J. L. Classifying human-robot interaction: an updated taxonomy. In *SMC (3)* (pp. 2841-2846), 2004

[90] https://perf.wiki.kernel.org/index.php/Tutorial

[91] http://www.brendangregg.com/perf.html

[92] https://www.python.org/download/releases/2.7/

[93] https://pythonhosted.org/spyder/

[94] https://docs.python.org/2/library/turtle.html

[95] Breiman, L. Random forests. Machine learning, 45(1), 5-32, 2001

[96] http://www.datasciencecentral.com/profiles/blogs/random-forests-algorithm

[97] http://www.dabi.temple.edu/~hbling/8590.002/Montillo_RandomForests_4-2-2009.pdf

[98] http://scikit-learn.org/stable/documentation.html

[99] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., & Vanderplas, J. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12(Oct), 2825-2830, 2011

[100] https://youtu.be/S9fn2irSP-4

[101] McKinney, R. A., Zapata, M. J., Conrad, J. M., Meiswinkel, T. W., & Ahuja, S. (2010, March). Components of an autonomous all-terrain vehicle. In IEEE SoutheastCon 2010 (SoutheastCon), Proceedings of the (pp. 416-419). IEEE.

[102] Cortner, A., Conrad, J. M., & BouSaba, N. A. (2012, March). Autonomous all-terrain vehicle steering. In Southeastcon, 2012 Proceedings of IEEE (pp. 1-5). IEEE.