NAVIGATION OF MOBILE ROBOTS IN THE PRESENCE OF OBSTACLES (A COMPARATIVE STUDY)

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

by

BESİME ERİN

In Partial Fulfilment of the Requirements for the Degree of Doctor of Philosophy in Computer Engineering

NICOSIA 2011

ABSTRACT

One of the key challenges of mobile robot navigation is the navigation of robots in environments that are densely cluttered with obstacles. The environment which a robot moves in may vary from static areas with fixed obstacles, to fast changing dynamic areas with moving dynamic obstacles. Many researchers have studies and reported various types of control methods and these methods can be classified as global and local navigation methods. The control of robots using the traditional control algorithms is not satisfactory as far as the navigational accuracy and the time to reach the goal are concerned, when the robot is in a complicated surrounding. A literature search has shown that the most popular control methods for such systems are based on potential field method, vector field histogram method, and local adaptive navigation schemes that tightly couple the robot actions to the sensor information. The development of robot navigation algorithms in uncertain environments densely cluttered with obstacles is one of the most challenging tasks. One of alternative and powerful ways of constructing a control system is the use of fuzzy systems approach, and because of the environmental uncertainties, fuzzy navigation algorithm has been proposed by the author.

The navigation of a mobile robot using potential field method, vector field histogram method, local adaptive navigation schemes and fuzzy navigation have been developed in this thesis. The structure and various control algorithms of mobile robot navigation have been presented. Using fuzzy logic, a powerful navigation algorithm has been developed by the author which will guide the robot from a starting point to a goal point in the shortest possible time. For navigation of the robot, the new knowledge base that includes fuzzy terms have been created. This fuzzy knowledge base describes the relation between the angles from the left and right corners of obstacles and determines the robot turn angle by considering the distances from the obstacles. The control rules of navigation and inference engine operations have been described. These control rules allow the robot to thoroughly use the available sensor information when choosing the control action to be taken. The development of navigation systems based on potential field method, vector field histogram method, local adaptive navigation scheme, and fuzzy navigation are carried out by developing a Visual C++ (2008) simulation

program. The comparative simulation results of robot navigation system demonstrate the advantage of the fuzzy navigation algorithm.

ÖZET

Engellerle donanmış ortamlarda, hareketli robot uygulamalarındaki başlıca zorluklardan biri robotun böyle bir ortamdaki navigasyonudur. Robotun hareket ettigi ortam sabit engelli ve sabit bölgelerden, hızla değişen dinamik engelli dinamik bölgelere kadar değişebilir. Birçok araştırmacı navigasyon probleminin çözümünü evrensel ve lokal olarak tanımlamış ve değişik çesit kontrol metodlarıyla çözmüştür. Karmaşık durumlarda robot konrolünün mevcut klasik algorıtmalarla doşruluk, ve çabukluk (mesafede ve zamanda) açısından amaca ulaşmak için yeterli olduğu söylenemez. Hareketli robot navigasyon sisteminin karmaşıklığı sistemi yüksek ücretli ve riskli yapabilir. Bu tür sistemler için en popüler kontrol metodları potansiyel alan metodu, vektör alan histogram metodu ve de lokal uyarlanabilir navigasyon şeması olup bütün bu metodlar robotun haraketlerini algılayıcı sensör bilgisine sıkıca bağlar. Bilinmeyen ortamlarda engellerle donatilmis robot navigasyon sistemi tasarımı önem kazanır. Bu tasarımlardan bir tanesi "bulanık mantık" kontrol sistemi olup çevresel bilinmeyenler sebebiyle bu sistem önerilmiştir.

Bu tezde hareketli robot sistemlerini potansiyel alan metodu, vektör alan histogram metodu, lokal uyarlanabilir navigasyon şemasi ile bulanık mantık navigasyon sistemi geliştirilmiştir. Hareketli robotların yapısal ve kontrol edilebilir algoritmaları takdim edilmiştir. Ayni zamanda, bulanık mantık prensiplerini kullanarak, bu hareketli robotların kontrol sistemi de geliştirilmiştir. Robot navigasyonu için yeni bilgi veritabanı bulanık mantık terimleri kullanarak yaratılmıştır. Bu bulanık mantık veri tabanı her engelin sol ve sag köşesinden ve robotun dönüş açısından, engelden ve aksiyondan olan uzaklık bağıntılarını tanımlamaktadır. Navigasyon, "inference engine operation" olarak tanımlanıyor. Bu kontrol kuralları robotun algılanan bilgileri verimli bir şekilde kullanmasını sağlar. Navigasyon sistemlerinin geliştirilmesi, potansiyel alan metodu, vektör alan histogram metodu, lokal uyarlanabilir navigasyon şemasi, ve bulanık mantık navigasyonu Visual C++(2008) simülasyon programı ortamında yapılmıştır. Karşılaştırılan simulasyon sonuçları neticesinde robot navigasyon sisteminde bulanık mantık navigasyon algoritmasının avantajı görülmektedir.

ACKNOWLEDGMENT

Studying at the Department of Computer Engineering, working with a highly devoted teaching community, remain one of the most memorable experiences of my life. This acknowledgement is an attempt to earnestly thank my teachers who have directly helped me during preparation of my thesis.

I would like to take special privilege to thank my supervisors Prof.Dr. Rahib Abiyev and Prof.Dr. Dogan Ibrhim who allocated me a thesis in the area of my interest. It was because of their invaluable suggestions, motivation, cooperation and timely help in overcoming problems that the work is successful.

Last but not the least, I would thank my mother and my brother who have been inspirational and very helpful in the development of this thesis.

TABLE OF CONTENTS

1. INTRODUCTION

Nowadays robots are used in different areas, these are automation of different industrial sectors, replacement of human works, game plying etc. Basically robots are some automated platform, equipped with actuators and sensors that are operating under the control of a computing system [1]. The robot performs tasks by executing motions in workspace within the real world [2].

The problem of mobile robot navigation is a wide and complex one. The environments which a robot can encounter vary from static indoor areas with a few fixed objects, to fast-changing dynamic areas with many moving obstacles. The goal of the robot may be reaching a prescribed destination or following as closely as possible a pre-defined trajectory or exploring and mapping an area for a later use [4]. In most cases, this task is broken down into several subtasks. These are;

1. Identifying the current location of the robot, and the current location of things in its environment,

2. Avoiding any immediate collisions facing the robot,

3. Determining a path to the objective, and

4. Resolving any conflicts between the previous two subtasks, taking into account the kinematics of the robot.

For maximum usefulness, the techniques to achieve these tasks should be generalized. They should be usable in a wide variety of situations, work in both static and dynamic environments, and be applicable to a large class of robots.

Design of a fast and efficient procedure for navigation of mobile robots in the presence of obstacles is one of important problems in robotics. Given the initial and final configurations of a mobile robot, the navigation algorithm must be able to determine whether there is a continuous motion from one configuration to the other, and find such a motion if one exists. The task of the navigation system is to guide the robot towards the goal without collision with obstacles.

Obstacle avoidance is the primary requirement for any autonomous robot, and designing a robot requires the integration and coordination of many sensors and actuators. In general, the robot acquires information about its surrounding through various sensors mounted on the robot. Usually, multiple sensors or camera can be used to detect the presence of obstacles.

1.1. Navigation Methodologies

Many efforts have been paid in the past to develop various robot navigation algorithms. Depending on the environment surrounding the robot the research in this field can be classified into two major areas: the *global navigation* and the *local navigation* [1,2]. In global navigation, the environment surrounding the robot is known and a path which avoids the obstacles is selected. In one example of the global navigation techniques, graphical maps which contain information about the obstacles are used to determine a desirable path. In local navigation, the environment surrounding the robot is unknown, or only partially known, and sensors have to be used to detect the obstacles and a collision avoidance system must be incorporated into the robot to avoid the obstacles. For a mobile robot, it is likely to be able to sense only the nearby obstacles, which constraints the applications of global navigation methods. On the other hand, the local motion planning methods dynamically guide the robot according to the locally sensed obstacles, which requires less prior knowledge about the environment.

There have been developed numbers of methodologies for Local and Global robot navigation. Local navigation systems are many and varied. There are many approaches in each type of system. The most commonly used are the methods based on the use of Artificial Potential Fields (APF), Vector Field Histogram (VFH) Technique, local navigation, fuzzy navigation techniques. Others are Vector Field Histogram (VFH) Technique, Dynamic Window Approach, Agoraphilic Algorithm, Rule Based Methods and Rules Learning Techniques.

The objective of the “Global Navigation” is to find an optimal path from the robot’s current location to the goal position. In Goal Seeking the goal position could be fixed or moving (as in it docking of two mobile robots) [6]. This can be observed as the problem of path-planning in a partly observed universe. In this well-searched area there are a number of algorithms have been developed that can give the optimal path to the goal. Global navigation could be realized using search trees. Global Planning; finding the optimal path in a dynamic environment is

hard. To solve this and find near optimal path the "Time-minimal Path Planner" was proposed.

The other different approaches, Predictive Modeling, Probabilistic techniques, Command Arbitration, Distributed Architecture, and Hybrids-Integrated Approaches are proposed for robot navigation problems.

Many times the robots are moved in unknown environments. For these objects the use of local navigation methods for avoiding obstacles and seeking goal acquire importance. This thesis is devoted to the local robot navigation methods.

1.2. Thesis Objectives

In most cases the environment is characterized with uncertainty, fast-changing dynamic areas with many moving obstacles. The aim of this thesis is the development of robot navigation systems avoiding from obstacles in unknown environment. The design of intelligent robots needs the considering of uncertainties inherent to unstructured environments, unreliable and incomplete perceptual information. The objective of this thesis is to show, explain and justify the development of a computer simulation program that implements known mobile robot navigation algorithms in the Windows XP/Visual Studio 2008 environment using Visual C#. This software allows the comparison of these traditional and fuzzy algorithms on the same canvas using the same obstacle configuration and also measures their differences in time and number of points generated. This thesis also provides in-depth knowledge about path planning and robot navigation in general and explains in detail the background behind the three traditional navigation algorithms and the fuzzy navigation algorithm which has been developed. The traditional navigation algorithms used on the educational platform have been analyzed using a statistical package but the fuzzy navigation algorithm has been compared on the same canvas as the others to see the difference.

This thesis has been organized as a description of the development of an robotics software including a fuzzy navigation algorithm for the purpose of simulation of mobile robot navigation. Thesis includes four chapters, conclusion, references and appendices.

Chapter one includes literature analysis on mobile robot navigation. The description of more used Local and Global Navigation Methods are described. Description and theoretical background on the traditional navigation algorithms implemented and state of application fuzzy technology in robot navigation are given.

Chapter two describes the description of robot navigation algorithms including Potential Field (PF) method, Vector Field Histogram (VFH) method and local adaptive navigation method. The mathematical background of navigation methods are described.

Chapter three describes newly developed fuzzy navigation algorithm. The structure of fuzzy system for robot navigation is described. The knowledge base of robot navigation is formed and inference mechanisms of fuzzy navigation are given.

Chapter four describe the computer modelling of robot navigation algorithms including Potential Field, Vector Field Histogram, local adaptive navigation and fuzzy navigation methods. The algorithms and flowcharts of navigation methods are presented. The simulator is develop and applied for educational purposes. It also includes a full summary of the simulation results of the aforementioned simulator and a full description of its use.

Conclusion includes important results obtained from the thesis.

Appendix includes fragments of robot navigation programs.

2. REVIEW ON MOBILE ROBOT NAVIGATION

2.1. Overview

In this chapter review of robot navigation problems using different techniques are described.. The description of different Local Navigation Methods such as Artificial Potential Field, Vector Field Histogram, Rule Based and Machine Learning Methods and also fuzzy navigation methods are presented.

Most mobile robot navigation techniques are developed at two levels: Local level and Global level. Recently local navigation by reactive means is employed by techniques such as fuzzy systems. In this a robot relies only on current or recent sensor data. This is useful for rapid responses to avoid collisions. The state of application of fuzzy technology for robot navigation is described.

2.2 Robot Navigation Methods and Obstacle Avoidance

2.2.1 Local Navigations

There are many kinds of Local navigation systems. The most commonly used are the methods based on the use of Artificial Potential Fields (APF), first proposed by Khatib, and popularized by Borenstein and Koren [9]. Others are Vector Field Histogram (VFH)[6] Technique, Local Navigation and Fuzzy Navigation Techniques [8].

Artificial Potential Fields: The "Artificial Potential Fields" (APF) method involves modeling tile robot as a particle in space, acted on by some combination of attractive and repulsive fields [9] [10]. In this technique obstacles and goods are modeled as charged surfaces, and the net potential generates a force on the robot. These forces push the robot away from the obstacles, while pulling it towards the goal. The robot moves in the direction of greatest negative gradient in the potential. Despite its simplicity this method can be effectively used in many simple environments [4].

Vector Field Histogram (VFH) Technique: Borenstein and Koren [6, 7] devised the "Vector Field Histogram'' (VFH) technique. The VFH uses local maps consisting of occupancy grids. These grids are transformed into maps where the occupied cells contain peaks, and open spaces contain troughs. The robot is then drawn towards the troughs. Ulrich and Borenstein extended this into VFH+ method. This method includes the robots kinematic

constraints when generating the troughs and peaks. Only open spaces which can be entered by the robot while travelling at its current speed are made into troughs. If no open spaces are available, the robot's speed is reduced, until an opening shows up. If no opening is found, the robot stops.

Dynamic Window Approach: The method is applied to move the robot using a model of the positions attainable by the robot. Fox [11] developed a "Dynamic Window Approach" (DWA) as a function of the robot's velocity space instead. By considering all the possible velocities attainable by the robot and applying those that lead away from any impending collisions, the robot can navigate at fast speeds through its environment. [11] extends this idea to "μDWA" such that a robot uses not only the map generated by what the sensors see, but also overlays what the robot expects to see. This method enables the root to the repulsion of obstacles not in the robots direction of avoid obstacles it has not yet seen, but is aware of it. This is useful when maps of some fixed obstacles (e.g. walls) are known, but other dynamic obstacles (e.g. people) are unknown.

Agoraphilic Algorithm Another mobile robot navigation technique is the "Agoraphilic Algorithm" developed by Ibrahim and McFetridge [12]. In this method, the force is generated not by the obstacles, but the free space. This involves only an attractive force, and no repulsive forces. This free space attraction is moderated by a fuzzy controller which keeps the robot moving towards the goal. An advantage of this algorithm is that the robot is unconcerned with edges of objects. Hence, the robot can freely pass through doorways, and near corners, if the goal is beyond such obstacles

Rule Based Methods: Rule-based methods can be employed as an alternative to field-based methods. These can be simple rules of the form developed by almost every hobby robotics [13] - "if left sensor active, turn right", or they can be generalized by fuzzy logic and machine learning techniques. Simple rules are easy to use, but are often very limited to the environments in which they are built. Fuzzy control systems, employing fuzzy set theory, have been proven to produce better performance than simple rules. However, these too are limited to environments highly similar to the one in which they were constructed.

Rules Learning Techniques: Aoki [14] used a robot programmed only with its basic set of performable actions, and allowed the robot to learn its fuzzy rules by exploring its environment, using a reinforcement learning algorithm. Therefore, the robot develops the rule set that is suited to its environment in a methodical way. However, these rules still need to be

automatically generated for each new environment. For generalized me, a great deal of time needs to be spent in learning every possible environment the robot may encounter.

Other rule learning methods have included neural networks, as used by Tsoukalas and evolutionary/genetic techniques, as used by Ram. Those techniques have similar strengths to Aoki's approach, and have a faster relearning time. They are able to retrain online, rather than offline.

These learning systems often do not explicitly include the robots kinematic constraints into the algorithm. Instead, the kinematic constraints are usually contained within the automatically generated rules.

2.2.2 Global Navigation

The objective of the "Global Navigation" is to find an optimal path from the robot's current location to the goal position. The goal position could be fixed or moving (as in it docking of two mobile robots). This can be observed as the problem of path-planning in a partly observed universe. In this well-searched area there are a number of algorithms have been developed that can give the optimal path to the goal.

Search Trees: : In Computer Science search trees include brute-force search of the known map. This involves constructing all the possible paths, and discarding those which are no optimal.

A dynamic search method was developed by Stentz and Hebert [15], is an improvement of the brute force algorithm. This is functionally similar, but claimed to be over 200 times more efficient.

Kuffner developed a simple algorithm based on Rapidly-exploring Random Trees (RRTs). RRT Connect works by creating trees starting at the start and the goal configurations [16]. The trees each explore space around them and also advance towards each other through the use of a greedy heuristic. This efficiently solves the path planning problem, even in high dimensions.

Global Planning in a Dynamic Environment: It is well known that global search for the optimal path in a dynamic environment is NP-hard. This means that any algorithm to determine this will not complete in reasonable time for real-time operation. Therefore, various

techniques have been developed to address the finding of 5 “near-optimal” robust path in a dynamic environment.

Fujirriura developed a “Time-minimal Path Planner”. In this method the robot first computes the motion of every obstacle in its environment, then plans the longest distance it can cover in the shortest amount of time. The robot moves along this “time-minimal path, and then re-plans the next path. Thus a sequence of motions leading to reaching the goal is achieved. The drawback of this method is that the motion of all obstacles must be known in advance, making the system impractical for general use.

Predictive Modeling: Different researchers including Tsubouchi and Gutsche [17] have worked on the predictive modelling of dynamic obstacles in order to develop continuous collision-free paths. Tsubouchi and Foka both used a four-step cycle of observing the current motion of the obstacles, generating forecasts of future motion, devising short-term collision-free paths, and devising motions along one of those paths. Tsubouchi used machine-learning techniques to develop the models of object motion, while Foka used Markov Decision Processes; Gutsche used implicit rules of statistically significant behavior. This is used to build a global estimate of the flow of objects.

Probabilistic techniques The methods mentioned above assume the availability of unambiguous localization information, i.e. the position of the robot is known, at least approximately. However, they all break down if the location is known only probabilistically. In the latter case, there are fewer well-researched techniques available. Cassandra Simmons and Howard all work with policy-based techniques, adapted from Markov Decision Processes using certain policy for combining the desired action at all possible locations. By acting, iteratively, on the combined action, the robot progresses forward. After each iteration, the set of possible locations decreases, until the robot reaches the goal.

Takeda developed a planning process which takes into account the likelihood of localization errors in the development, of paths. This means that the robot follows a path which leads t o the fewest potential localization errors [18,19].

Command Arbitration: The above mentioned systems for obstacles avoidance and goal reaching are independent of each other. They often produce conflicting commands. Some program logic must go into deciding between various instructions computed by the various

navigators. It is also possible at this level to arbitrate between multiple local or global planners. This can produce more complex behaviors than a single planner can provide.

One of the simplest architectures for a command arbiter is subsumption architecture. In that architecture multiple planners are layered, so that higher levels subsume lower ones. The robot needs acquire "competencies" at each level. Lower level competencies include local responses "avoid objects", while higher levels can perform global navigation by selectively suppressing the commands of the lower levels. This is a strict hierarchical system, and while it is robust, there is little flexibility within the system [19].

A more flexible system for the selection between various commands is the voting system. An example of this technique is the [20], [21]. The systems acts on the string actions with the highest votes within the search space.

Hybrid-Integrated Approaches: In this approaches the whole motion problem can be considered as a single task. This requires an integrated model of both goal-acquisitions as well as collision avoidance. This is challenging because the task of global planning is far more computationally intensive than that of local navigation, and collision avoidance requires much quicker response times.

Low [23] used Cooperative Extended Kohonen Maps to develop an integrated global and local planning system. This reduces the duplication of effort in planning the motions on for both global and local navigation. Therefore, a performance, comparable t o a local reactive system is achieved without the need for an extra global planning level. This enables the robot to escape concave and corridor-like obstacles.

Xiao [24] describe an Evolutionary Planner/Navigators (EP/N) which in parallel recompute the optimal path using an evolutionary (genetic) algorithm, while feeding in new obstacle information directly into the search algorithm. While this work does not directly relate to the dynamic problem, it showed considerable promise [24].

2.3 Review on Navigation Methods

The most common used techniques in navigation are Artificial Potential Fields Method [25], Vector Field Histograms[6,26], Local Adaptive Navigation[27] and Fuzzy Logic based Navigation Techniques, Hybrid techniques.

As mentioned above the application of artificial potential fields to obstacle avoidance was first developed by Khatib. This approach uses repulsive potential fields around the obstacles (and forbidden regions) to force the robot away and an attractive potential field around goal to attract the robot. Consequently, the robot experiences a generalized force equal to the negative of the total potential gradient. This force drives the robot downhill towards its goal configuration until it reaches a minimum and it stops. The artificial potential field approach can be applied to both global and local methods [25].

The simple APF method described has several key problems.

- The field may contain local minima, especially when there are many obstacles in the environment. This can trap the robot, it's a gradient-descent algorithm cannot escape from it local minimum.

- The robot may find itself unable to pass through small openings such as through doors.

- The robot may exhibit oscillations in its motions.

- The robot may be guided away from the goal by a moving obstacle which creates a moving local minimum. Being stuck in this "shadow" means that the robot cannot move around the obstacle.

There have been various attempts to address these issues in the APF. These have included virtual obstacles created to offset minima, alternate field functions including harmonic functions and distance transform with no local minima. Borenstein developed a modified response method, by decreasing the repulsion of obstacles not in the robots direction of motion. This reduces the amount of oscillation, while still allowing the robot to avoid obstacles in its path. These are successful in the static environment, but may not be as suitable for the dynamic environment as the computational complexity is very high.

The global methods assume a priori knowledge of the workspace and are usually based on the construction of the robot's configuration space which shrinks the robot to a point. However, the global methods require that two main problems be addressed first, the obstacles must be mapped into the robot's configuration space. Second, a path through the configuration space must be found for the point representing the robot. To generate these paths, the artificial potential methods' surrounds the configuration space obstacles with repulsive potential energy functions, and places the goal point at a global energy minimum. The point in configuration

space representing the robot is acted upon by force equal to the negative gradient of this potential field, and driven away from obstacles and to the mini.

In [25], it is stated that the potential field methods were rapidly gaining popularity in obstacle avoidance applications and potential field principle is particularly attractive because of its elegance and simplicity. Substantial shortcomings have been identified as problems that are inherent to this principle. Based upon mathematical analysis, [25] presents a systematic critism of the inherent problems. The heart of this analysis is a differential equation that combines the robot and the environment in a unified system. A potential field method (PFM) especially designed for real-time obstacle avoidance with fast mobile robots is designed called Virtual Force Fıelds(VFF) method. It is stated that there is a major drawback of this PFM, that is, the configuration space might include regions in which a certain condition which controls the stability of the robot is not satisfied and the robot will begin to oscillate. Obviously, unstable conditionsexist also for the general environment model. The parameter that can be tuned to guarantee stability is f which is defined as a ratio between the repulsive force constant and the target force constant and should be determined experimentallyç

Enxiu SHI and Junjie GUO [28] analyzing the disadvantage of Artificial Potential Field (APF) for mobile robot obstacle avoidance, local trap and vibrating, developed a new method for improving Artificial Potential Field. Authors approved that local trap was eliminated effectively and vibrating was mitigated but not eliminated through adjusting the parameters of the functions. The potential force area of mobile robot obstacle avoidance was limited and adjusted at any moment. The problem of mobile robot obstacle avoidance based-on APF was eliminated and the rapidness of arriving at the target was improved by this method.

Warren [29] considered a technique for coordinating multiple robots and finding the paths in the presence of obstacles. To accomplish this, the robots are prioritized. A path that avoids only the stationary obstacles is planned for the highest priority robot. Then, a trajectory for the next lower priority robot is planned so that it avoids both the stationary obstacles and the higher priority robot which is treated as a moving obstacle. This process is continued until trajectories for all of the robots have been planned. Potential fields are applied and are used to modify the path of the robot in order to avoid from the obstacles. The advantage of using artificial potential fields is that they offer a relatively fast and efficient way to solve for safe trajectories around both stationary and moving obstacles.

.

Marta C. Mora and Josep Tornero [30] developed a multi-rate predictive artificial potential field method for dynamic and an uncertain environment. It is based on the combination of classical Artificial Potential Field methods (APF) with Multi-rate Kalman Filter estimations (MKF), which takes into account present and future obstacle locations within a temporal horizon. By doing that, position uncertainty of obstacles is considered in the avoidance algorithm. This implies anticipation to the movement of the obstacles and its consideration in the path planning strategy. Forces derived from the potential field are taken as control inputs for the system model as well as considered in the Kalman Filter estimation. This leads to the generation of a local trajectory that fully meets the restrictions imposed by the kinematic model of the robot.

In [31] Evolutionary Artificial Potential Field (EAPF) for real-time robot path planning is considered The artificial potential field method is combined with genetic algorithms, to derive optimal potential field functions. The proposed Evolutionary Artificial Potential Field approach is capable of navigating robot is situated among moving obstacles [29,30]. Potential Potential field functions for obstacles and goal points are also defined. Multi-objective evolutionary algorithm (MOEA) is utilized to identify the optimal potential field functions. Fitness functions like, goal-factor, obstacle-factor, smoothness-factor and minimum-path length- factor are developed for the MOEA selection criteria. An algorithm named escape-force is introduced to avoid the local minima associated with EAPF. Moving obstacles and moving goal positions were considered to test the robust performance of the proposed methodology.

[32] describes Exact Robot Navigation Using Artificial Potential Functions. In a dated paper the preliminary implementation of mobile robot navigation using Artificial Potential Functions is described. The methodology is presented for exact robot motion planning and control that unifies the purely kinematic path planning problem with the lower level feedback controller design. Complete information about the freespace and control is encoded in the form of a special artificial potential function (a navigation function). This function connects the kinematic planning problem with the dynamic execution problem in a provably correct function. The navigation function automatically gives rise to a bounded-torque feedback controller for the robot's actuator that guarantees collision-free motion and convergence to the destination from all initial free configurations. The formula that is presented admits sphere-worlds of arbitrary dimension and is directly applicable to configuration spaces whose forbidden regions can be modelled by such generalized discs. As said at the beginning, one of

the algorithms that I have used in this thesis for my simulation software, the basis for this thesis, uses potential field algorithm and the paper referenced above is the background for the development of this algorithm.

One of popular obstacle avoidance method is based on edge detection which is called Edge Detection Method. In this method, an algorithm tries to determine the position of the vertical edges of the obstacle and then steer the robot around either one of the "visible" edges. The line connecting two visible edges is considered to represent one of the boundaries of the obstacle. A disadvantage with current implementations of this method is that the robot stops in front of obstacles to gather sensor information. However, this is not an inherent limitation of edge-detection methods; it may be possible to overcome this problem with faster computers in future implementations.

In some edge-detection approach (using ultrasonic sensors), the robot remains stationary while taking a panoramic scan of its environment. After the application of certain line-fitting algorithms, an edge-based global path planner is instituted to plan the robot's subsequent path. A common drawback of both edge-detection approaches is their sensitivity to sensor accuracy. Ultrasonic sensors present many shortcomings in this respect poor directionality secular's reflections. Any one of these errors can cause the algorithm to determine the existence of an edge at a completely wrong location, often resulting in highly unlikely

A method for probabilistic representation of obstacles in a grid-type world model has been developed at Carnegie-Mellon University (CMU). This world model, called a certainty grid, is especially suited to the accommodation of inaccurate sensor data such as range measurements from ultrasonic sensors.

In the certainty grid, the robot's work area is represented by a two-dimensional array of square elements, denoted as cells. Each cell contains a certainty value (CV) that indicates the measure of confidence that an obstacle exists within the cell area. With the CMU method, CV's are updated by a probability function that takes into account the characteristics of a given sensor. If an object is detected by an ultrasonic sensor, it is more likely that this object is closer to the acoustic axis of the sensor than to the periphery of the conical field of view. For this reason, CMU's probabilistic function increases the CV's in cells close to the acoustic axis more than the CV's in cells at the periphery.

In CMU's applications of this method, the mobile robot remains stationary while it takes a panoramic scan with its 24 ultrasonic sensors. Next, the probabilistic function is applied to each of the 24 range readings, updating the certainty grid. Finally, the robot moves to a new location, stops, and repeats the procedure. After the robot traverses a room in this manner, the resulting certainty grid represents a fairly accurate map of the room. A global path-planning method is then employed for off-line calculations of subsequent robot paths.

Vector Force Field Method is widely used method for robot navigation [6]. VFF method allows for fast, continuous, and smooth motion of the controlled vehicle among unexpected obstacles and does not require the vehicle to stop in front of obstacles. The VFF method uses a two-dimensional Cartesian histogram grid for obstacle representation. As in CMU's certainty grid concept, each cell in the histogram grid holds a certainty value that represents the confidence of the algorithm in the existence of an obstacle at that location.

The histogram grid differs from the certainty grid in the way it is built and updated. CMU's method projects a probability profile onto those cells that are affected by a range reading; this procedure is computationally intensive and would impose a heavy time penalty if real-time execution on an on-board computer was attempted. Increments only one cell in the histogram grid for each range reading, creating a "probability distribution" with only small computation. While this approach may seem to be an oversimplification, a probabilistic distribution is actually obtained by continuously and rapidly sampling each sensor while the vehicle is moving. Thus, the same cell and its neighbouring cells are repeatedly incremented. This results in a histogram probability distribution in which high certainty values are obtained in cells close to the actual location of the obstacle.

The Vector Field Histogram Method (VFH)[6] method uses a two-dimensional Cartesian histogram grid as a world model. This world model is updated continuously with range data sampled by on-board range sensors. The VFH method subsequently employs a two-stage data reduction process in order to compute the desired control commands for the vehicle. In the first stage the histogram grid is reduced to a one- dimensional polar histogram that is constructed around the robot's momentary location. Each sector in the polar histogram contains a value representing the polar obstacle density in that direction. In the second stage, the algorithm selects the most suitable sector from among all polar histogram sectors with a low polar obstacle density, and the steering of the robot is aligned with that direction.

The VFH method employs a two-stage data-reduction technique, rather than the single-step technique used by the VFF method. Thus, three levels of data representation exist:

1. The highest level holds the detailed description of the robot's environment. In this level, the two-dimensional Cartesian histogram grid C is continuously updated in real time with range data sampled by the on-board range sensors.

2. At the intermediate level, a one-dimensional polar histogram H is constructed around the robot's momentary location. H comprises n angular sectors of width CY. A transformation map the active region C* onto H, resulting in each sector k holding a value h, that represents the polar obstacle density in the direction that corresponds to sector k.

3. The lowest level of data representation is the output of the VFH algorithm: the reference values for the drive and steer controllers of the vehicle.

The concept of the VFH+ Obstacle avoidance algorithm is similar to the original VFH algorithm [7]. The input to this algorithm is a map grid of the local environment, called histogram grid, which is based on the earlier certainty grid and occupancy grid methods. The VFH+ method employs a four-stage data reduction process in order to compute the new direction of motion. In the first three stages, the two-dimensional map grid is reduced to one-dimensional polar histograms that are constructed around the robot's momentary location. In the fourth stage, the algorithm selects the most suitable direction based on the masked polar histogram and a cost function.

The VFH+ method builds a polar histogram around the robot's current position, looks for openings in the histogram, and then determines between one and three suitable directions for each opening [7]. VFH+ also assigns a cost value to each of these primary candidate directions. VFH+ then selects the primary candidate direction with the lowest cost as its new direction of motion.

The Agoraphilic algorithm which is a reactive local navigation technique based on the potential fields methodology is developed in [34]. The algorithm employs attractive, virtual forces generated by the surrounding free space. These attractive forces effective drive the robot toward the areas of greatest free space, while a fuzzy weighting function is applied to add bias to the free-space toward the goal location. The ability to focus the forces in such a way is utilized for behavior based control. The force shaping property can be exploited to

create a number of primitive behaviors and provide an alternative behavior fusion technique. The Free Space Force is an attractive force that effectively 'pulls' the robot toward the surrounding areas of free space. To stop the robot from wandering aimlessly trying to find an open space, a limiting function is applied to essentially focus the force toward the goal.

Atsushi Fujimori, Peter N. Nikiforuk, and Madan M. Gupta [27] proposed local navigation technique with obstacle avoidance for mobile robots in which the dynamics of the robot are taken into consideration. The only information needed about the local environment is the distance between the robot and the obstacles in three specified directions. The navigation law is a first-order differential equation and navigation to the goal and obstacle avoidance are achieved by switching the direction angle of the robot. Simulation examples are given in order demonstrate the effectiveness of the described technique. In this thesis this method is modelled for comparative purpose.

[35] describes a dynamic artificial neural network based mobile robot motion and path planning system. The robot car is able to navigate on flat surface among static and moving obstacles, from any starting point to any endpoint. The motion controlling ANN is trained online with an extended backpragation through time algorithm, which uses potential fields for obstacle avoidance. The paths of the moving obstacles are predicted with other ANNs for better obstacle avoidance.

[36] discusses genetic algorithms used to design a path for robot navigation where a specific one is explained thoroughly. Basically, the x and y coordinates of the grid containing obstacle and non-obstacle cells are contained in the chromosome structure. Each subsequent pair of genes contains the path point(x,y) for each path point. The path fitness is based on both path length and feasibility with a significant penalty for paths that collide with obstacles.

Cellular neural networks are used for mobile robot navigation in [37]. Cellular neural networks is a good method for mobile robot navigation because the method makes it possible to find an optimum free path in an unknown environment. When the robot is at a dead end, the path is replanned with no further information needed. The situation when there is no free path between the robot and the goal is always recognized by the algorithm. This method does not suffer from local minima problems.

2.4. State of application of fuzzy technology for robot navigation

When dealing with the navigation problem of a real-world mobile robot, it is usually difficult or impossible to obtain an accurate model of a dynamic environment. During navigation in unknown environments, the mobile robot generates control commands based on sensory data. The robot can navigate safely in a dynamic environment by reacting to obstacles detected by sensors in real time. A major drawback is that due to the limitation of sensors the robot may get lost even if a path to the goal exists.

To develop algorithms for mobile robot navigation in an unknown environment, the following points should be considered.
1) The mathematical model of the environment is generally unavailable.
2) Sensory data are uncertain and imprecise due to noise.
3) Real-time operation is essential.

In this regard, fuzzy-logic based algorithms have been proposed for designing robust controllers that are able to deliver satisfactory performance in face of large amounts of parameter variations and noise. In addition, due to its simplicity of implementation, fuzzy logic control is well suited for autonomous robotics. Recently fuzzy logic has been utilized in navigation systems for mobile robots. The one of them that uses fuzzy control in mobile robotics belong to reactive approaches. In 1985, Sugeno and Nishida developed a fuzzy controller to drive a model car along a track delimited by two walls. Shortly after, Takeuchi *et al.* used fuzzy logic control in the obstacle avoidance behavior of mobile robots. Later in 1991, Yen and Pfluger proposed a method of path planning and execution using fuzzy logic for mobile robot control. From then on, the efficiency of using fuzzy logic in mobile robot navigation systems has been well demonstrated. A comprehensive study of fuzzy logic-based autonomous robot navigation systems is given in [8]. Recently, several new and improved solutions to the mobile robot navigation problem in unknown environments based on fuzzy logic have been proposed [8], [38], extensively demonstrating that the interpolative nature of fuzzy control results in smooth movement of the robot and graceful degradation in face of errors and fluctuations in sensory data. The efficiency of using fuzzy logic in mobile robot navigation systems has been well demonstrated in these researches.

Nowadays fuzzy logic has become a means of collecting human knowledge and experience and dealing with uncertainties in the control process. Fuzzy logic is becoming a very popular topic in control engineering [38,39]. Considerable research and applications of this new area for control systems have taken place. Control is by far the most useful application of fuzzy logic theory, but its successful applications to a variety of consumer products and industrial systems have helped to attract growing attention and interest.

The basic idea of fuzzy control in mobile robot navigation may be classified into the categories described below according to the form of the fuzzy rule. The direction-based fuzzy rule takes the following form.

IF disallowed-direction is A and desired-direction is B, THEN steering-direction is C. Where A, B and C are all represented by fuzzy sets. This form of fuzzy rule combines information about obstacles and goal position together and gives the final steering direction which is safe, in the sense of avoiding collisions, and desired, in the sense of seeking the goal. The fuzzy rule bases designed in and typically consist of direction-based rules.

The speed-based fuzzy rule takes into account obstacle repulsion and goal attraction to set the speeds for the motors. Most conventional motion planning algorithms that are based on the model of the environment cannot perform well when dealing with the navigation problem for real world mobile robots where the environment is unknown and can change dynamically.

Xiaoyu Yang and his co-authors [40] developed a layered goal-oriented motion planning strategy using fuzzy logic. The paper considers navigating of a mobile robot in an unknown environment. The information about the global goal and the long-range sensory data are used by the first layer of the planner to produce an intermediate goal, referred to as the way-point, which gives a favourable direction in terms of seeking the goal within the detected area. The second layer of the planner takes this way-point as a sub goal and, using short-range sensory data, guides the robot to reach the sub goal while avoiding collisions. The resulting path, connecting an initial point to a goal position, is similar to the path produced by the visibility graph motion planning method, but in this approach there is no assumption about the environment. Due to its simplicity and capability for real-time implementation, fuzzy logic has been used for the proposed motion planning strategy. The resulting navigation system is implemented on a real mobile robot, Koala, and tested in various environments.

Hongche Guo, Cheng Cao, Junyou Yang and Qiuhao Zhang [41] developed an obstacle-avoidance control algorithm based on the fuzzy matching of obstacle environment. The algorithm is mainly used in the obstacle avoidance control of omni-directional Lower Limbs Rehabilitation Robot. The method generates the Eigen value of obstacle environment using detected angle information of obstacle boundary, and fuzzy matches with the known environment information to realize the obstacle-avoidance control of robot. The design of obstacle-avoidance control enhances the patient safety and decreases the environment using demand.

S.Parasuraman V.Ganapathy and Bijan Shirinzadeh [42] developed a method to encode the fuzzy sets, fuzzy rules and procedure to perform fuzzy inference into expert system for behaviour based robot navigation. The design of the behaviour is based on regulatory control using fuzzy logic and the coordination and integration is defined by fuzzy rules. Fuzzy rules define the context of applicability for each behaviour. The complexity of robot behaviour is reduced by breaking down robot behaviours into simple behaviours or units, and then combined with others to produce more complex actions. Fuzzy logic decision mechanism simplifies the design of the robotic controller and reduces the number of rules to be determined. In addition, the new behaviour can be added or modified easily. Some of the experimental results are given for the Obstacle avoidance, Wall following and Goal-Seek behaviours.

The next paper analyzes Fuzzy Logic Path Planner and Motion Controller by Evolutionary Programming for Mobile Robots [43]. In this paper, a fuzzy logic controller (FLC) consists of two levels: the planner level and the motion control level. The planner level generates a path to the destination by avoiding obstacles. The singleton outputs of the planner are obtained by using lines and arc methods obtained by heuristics and tuned by evolutionary programming. Evolutionary fuzzy system, real variables optimization method based on evolutionary algorithm, is used for tuning of fuzzy system in the path motion controller. The condition for optimization is the minimum time to reach to the target point.

Wheel speed and turn angle are the main parameters in the self navigation of a robot [44].. The parameters of self-navigation significantly influence robot-walking time and robot walking quality. In [44] a generic and intelligent approach of robot path determination is proposed. First a simplified robot navigation model is introduced. Second, a fuzzy rule-based system is established to control robot wheel speed based on the distance from the obstacles,

and the turn angle. Then the geometric features of the environment are identified, the wheel speed is determined and the optimal path found. The approach is applied for robot navigation, and the results are simulated with computer system to show the advantages of this approach.

Next, the discussion continues with a variation. In [45] mobile robot navigation is implemented using Alpha Level Fuzzy Logic System. This paper presents the issues associated with the Mobile Robot Navigation and the establishment of a new technique to navigate the mobile robot in a real world environment. The issues discussed are: (i) If multiple obstacles are appeared in the environment with equal distances as perceived from multiple sensors of robot, then the corresponding multiple obstacles are treated as a whole and the robot deviate from obstacles widely and avoid then reaching to the target. As a result of the wide deviation, the robot takes long time and long path to reach the target position, (ii) Behavior rule selection when multiple obstacles are appeared in the environment with equal distances from robot. The robot navigation with optimal path, time and rule selection are more important and critical task, whenever the mobile robots are engaged to search the lives in the event of natural disaster like earthquake etc. A methodology is proposed and used for resolving the above issues and discussed in this paper. The conclusion is the experimental study conducted to investigate the proposed Alpha-Level Fuzzy Logic System(ALFLS) methodology for mobile robot navigation using Khepera II mobile robot show that the proposed formulation reduces the complexity of building the navigation system in a complex environment. The proposed methodology demonstrated improved performance of mobile robot navigation in terms of the (i) smaller time taken of the robot to reach the target and (ii) the distance travelled to reach the target position, which is shorter compared to the other accepted methods.

Continuing on the literature that we have found on Fuzzy Navigation, there exists an article on A Fuzzy Controller With Supervised Learning Assisted Reinforcement Learning Algorithm for Obstacle Avoidance [46]. [46] show that fuzzy logic system promises an efficient way for obstacle avoidance. However, it is difficult to maintain the correctness, consistency, and completeness of a fuzzy rule base constructed and tuned by a human expert. Authors use a neural fuzzy system with mixed coarse learning and fine learning phases. In the first phase, supervised learning method is used to determine the membership functions for the input and output variables simultaneously. After sufficient training, fine learning which employs reinforcement learning algorithm to fine-tune the membership functions for the

output variables is applied. For sufficient learning, a new learning method using modified Sutton and Barto's model is proposed to strengthen the exploration. Through this two-step tuning approach, the mobile robot is able to perform collision-free navigation. To deal with the difficulty in acquiring large amount of training data with high consistency for the supervised learning, authors in [46] develop a virtual environment simulator, which is able to provide desktop virtual environment (DVE) and immersive virtual environment (IVE) visualization. Through operating a mobile robot in the virtual environment (DVE/IVE) by a skilled human operator, the training data are readily obtained and used to train the neural fuzzy system.

A mobile robot navigation method using fuzzy logic and a modified Q-learning algorithm is presented in [47]. This paper proposes a new fuzzy logic-based navigation method for a mobile robot in an unknown environment. This method endows the robot with the capabilities of obstacles constructed based on the human sense and a reinforcement learning algorithm is used to fine tune the fuzzy rule base parameters. [47] show that the advantages of the proposed method are its simplicity, its easy implementation for industrial applications, and the robot joins its objective despite the environment complexity.

In [48] a Comparison Between a Fuzzy Behavioural Algorithm and a Vector Polar Histogram Algorithm for Mobile Robot Algorithm is described. It is stated that fuzzy behavioural and vector field histogram(VFH) are two primary reactive approaches used for navigating autonomous ground vehicles(AGVs) through uncertain environments. In the paper the particular fuzzy behavioural approach is considered. The behaviours are considered using preference based voting and command fusion. Accordingly, the algorithm preference-based fuzzy behavioural algorithm (PBFB) will be called. The original VFH approach utilized sonar's as range finders. Due to the inaccuracy of sonar measurements, the algorithm compensates by considering the certainty of existence of the obstacles. The approach was modified to use a laser range finder and resulted in Vector Polar Histogram(VPH) algorithm. In the given method sensor readings are used to build a polar histogram which is further filtered and decoded to obtain the steering direction. The robustness of this algorithm primarily comes from the high accuracy of the laser sensors measurements. In contrast, in the PBFB algorithm the range data is classified into three fuzzy sets: short, medium and long. Hence, the PBFB algorithm is relatively insensitive to errors in the sensor readings. When high accuracy sensors are not available, the more complex VFH algorithm must be used

instead of the VPH algorithm, while the PBFB should experience minimal performance degradation. From the viewpoints of structural complexity, and ease of programming and tuning, the VPH approach takes less programming and implementation time. Hence, the VPH is the better choice for tasks where implementation speed is highly important. The PBFB algorithm generates much smoother paths, indicating that the PFPB algorithm would be the better choice for tasks where energy consumption and motion comfort are important, for example, in the design of semi-autonomous wheelchairs for the disabled.

As shown there are clearly a number of robust techniques for various key sub-problems in robot navigation. There are also wide varieties of techniques which are well developed while not completely robust. However, there is still no known technique or combination of techniques which will result in a robust, generalized performance. At the same time in more cases the environment is characterized with fast-changing dynamic areas with many moving obstacles. For these reasons the development of navigation algorithms avoiding as statistical as dynamical obstacles is becoming is very important problem. In this thesis a mobile robot navigation approach equipped with fuzzy rule-based navigation algorithm which uses goal oriented approach to navigation and obstacle avoidance system is proposed. The designed mobile robot should avoid from as static as dynamic obstacles.

Owing to its simplicity and hence its short response time, the fuzzy navigation is especially suitable for on-line applications with strong real-time requirements. On-line planning is an on-going activity. The planner receives a continuous flow of information about occurring events and generates new commands in response to the incoming events, while previously planned motions are being executed. The fuzzy-rule-base of the proposed system combines the repelling influence, which is related to the distance and the angle between the robot and nearby obstacles, with the attracting influence produced by the distance and the angular difference between the actual direction and position of the robot and the final configuration to generate actuating commands for the mobile platform.

3. THEORETICAL BACKGROUND OF ALGORITHMS IMPLEMENTED

3.1. **Overview**

The most common used techniques in navigation are Artificial Potential Fields Method, Vector Field Histograms, Local Adaptive Navigation and Fuzzy Logic based Navigation Techniques. In this section the description of these robot navigation methods- Potential Field Method, Vector Field histogram, local adaptive navigation method are given. Mathematical backgrounds on PFM, VFH, local adaptive navigation are described.

3.2 **Potential Field Method**

Potential field methods are rapidly gaining popularity in obstacle avoidance applications for mobile robots and manipulators. While the potential field principle is particularly attractive because of its elegance and simplicity, substantial shortcomings have been identified as problems that are inherent to this principle.

The philosophy of the potential field approach is that the mobile robot moves in a field of forces. The goal position to be reached is an attractive potential while each obstacle generates a repulsive potential. A potential field can be viewed as an energy field and so its gradient at each position is a force. Potential fields can be applied locally while path determination, trajectory planning and control steps are achieved in one step in real time. The idea of obstacles exerting virtual repelling forces towards a robot, while the target generates a virtual attractive force uses a similar concept that takes into consideration the robot's velocity in the vicinity of obstacles. In one example called the Brooks implementation [33], if the magnitude of the sum of repulsive forces exceeds a certain threshold, the robot stops, turns into the direction of the resultant force vector, and moves on. This method also requires the robot to stop and thus is not suited for teleautonomous operation. The theory of the potential field method is given below briefly.

In the Artificial Potential Field approach, Potential Field was originally developed as on-line collision avoidance approach, applicable when the robot does not have a prior model of the obstacles, but senses them during motion execution. Using prior model of the workspace, it can be turned into a systematic motion planning approach. Potential field methods are often referred to as "local methods". This comes from the fact that most potential functions are

derived in such a way that their calques at any configuration do not depend on the distribution and shapes of the obstacles beyond some limited neighborhood around the configuration.

In order to avoid the difficulties associated with the dynamical model, the control law is based only on the gravitational potential and a new artificial potential. It is shown that to drive the mobile robot to a desired point in an unconstrained movement is necessary the artifical potential to be a potential functional whose point of minimum is attractor for the system. Also this method is used for a constrained movement in the environment with obstacles. The target position is represented by an artificial attractive potentail field and obstacles by corresponding repulsive fields, so that the trajectory to the target can be associated with a unique flow-line of the gradient field through the initial position and can be generated through the initial position and can be generated via a flow-line tracking process. This approach is suitable for real-time motion planning of robots since the algorithm is simple and computationally much less expensive than other methods based on global information about the task space. It is difficult in the artificial potential field framework to regulate the trasient behavior of the generated trajectories such as the movement time to the target and the shape of the velocity profile [26].

3.3. Mathematical Background on Potential Field Method continues with the below given simplified model of the mobile robot:

A potential, $\phi(r)$ is defined by the Laplace equation $\nabla^2\phi = 0$ in a closed region, Ω, of continuous, equal connectivity. The boundary of Ω, $\partial\Omega$ does not have to be connected. It includes the surfaces of all obstacles and the goal point. $\phi(r) = \phi_1$ at the surfaces of obstacles and $\phi(r) = \phi_0$ at the goal point. There are no local minima on $\phi(r)$. Nevertheless, the exponential decay of the field from any point leads to areas where the magnitude of the gradient on $\phi, |\nabla\phi|$, is very small while the range of $|\nabla\phi|$ may be very large. The field decays rapidly near the goal, and far from the goal there is only a slight change in the field.

In this thesis, the potential value is calculated for each point on the grid by using the Laplaces's equation where the value of the field at the goal point is set to the value of -2^{124} and boundary points are to zero.

The Laplace equation in two dimensions is represented on equally spaced and connected grid by the following partial differential equation:

$$\phi_{(i,j)} = \frac{\phi_{(i+1,j)} + \phi_{(i-1,j)} + \phi_{(i,j+1)} + \phi_{(i,j+1)}}{4}$$

where i is position on the grid in the x direction, j is position on the grid in the y direction
The potential value given above for each grid point is referred to as the gridValue, a two-dimensional array. gridSize is the parameter that determines the length and width of the grid cell. Field values are calculated for any point in the workspace by using linear interpolation. Passing two parameters a and b into the function field, we use the following sets of equations to determine the field value at any given point that is anywhere on the canvas (i.e. can be anywhere in-between the grid). In this algorithm, *gridSize* is the length and width of each grid cell. *numberofGridLinesX* and *numberofGridLinesY* are the number of grid cells along the X and the Y axes. *dx* and *dy* are the differential values for x and y values. *sx* and *sy* are simply approximations. *bot1* and *bot2* are calculated values which eventually are used to calculate the *potential field* value at any given point i.e. this is linear interpolation. Computing of potential field is given in the following fragment:

x = a / gridSize
y = b / gridSize;
gridx = numberOfGridLinesX + 1
gridy = numberOfGridLinesY + 1

dx = x - Floor(x)
dy = y - Floor(y)
sx = Floor(x)
sy = Floor(y)
bot1 = (1.0 - dy) * gridValue[sx, sy] + dy * gridValue[sx, sy + 1]
bot2 = (1.0 - dy) * gridValue[sx + 1, sy] + dy * gridValue[sx + 1, sy + 1]

field(a,b) = dx * bot2 + (1.0 - dx) * bot1

The mathematical function *Floor(x)* returns the largest integer less than or equal to the specified double-precision floating-point number.

One problem with potential fields is that they may have local minima where the robot gets trapped before reaching the goal. Another situation which may cause local minima is closely spaced obstacles. Because of this the potential field approach can also be applied globally by means of numerical potential fields or navigation functions that can be defined on a grid without local minima. Another problem with potential fields is unstable oscillation where the dynamics of the robot and/or the environment gets unstable. This may be caused by high speeds, narrow corridors, or sudden changes in movement. Since a point robot can be defined as a mobile robot with no dimension, it can simply reach the goal by following the gradient descent. The mobile robot is moved towards the goal in such a way that the center of the robot is moved from the current point to the next point on the path while the orientation of the robot is taken as the tangent angle of the current point. Finding the tangent angle at each point of a given path and then calculating the updated robot position is a simple process as specified below:

$$top = field(robx, roby - 1)$$

$$bottom = field(robx, roby + 1)$$

$$left = field(robx - 1, roby)$$

$$right = field(robx + 1, roby)$$

robx and *roby* are the current x and y positions of the mobile robot respectively.

$$\alpha = \tan^{-1} \frac{top - bottom}{left - right}$$

$$x_{\min} = lineLength * \cos(\alpha)$$

$$y_{\min} = lineLength * \sin(\alpha)$$

$$robx = robx + x_{\min}$$

$$roby = roby + y_{\min}$$

where, *lineLength* is a parameter which sets the distance the mobile robot travels at each simulation tick and the routine *field(x,y)* calculates the potential field value given by the gridpoint x and y.

A few general approaches such as *roadmaps*, *cell decomposition*, and *potential field* are widely used in motion planning algorithms. Although *the potential field* technique has some disadvantages such as local minima, it has applied to a variety of applications. Especially, numerical potential fields, *distance transform*s and *harmonic potential fields* are attractive since they do not contain local minima and can be defined in a W-space.

3.4. Vector Field Histogram Plus

The Vector Field Histogram Plus (VFH+) method includes four stages for computing direction of robot motion. In first three stages the two-dimensional map grid is transformed into one-dimensional polar histograms. These are implemented using primary polar histogram, binary polar histogram and masked polar histogram. In last stage, using masked histogram and cost function the algorithm selects the suitable direction for the robot. The brief description of these stages is given below.

In VFH+ method, there exists a circular window with diameter w_s where the robot scans it environment. This forms our histogram grid which has dimension w_s x w_s. In this histogram grid, each cell has a certainity value $c_{i,j}$ which has value 1 where we are confident there exists part of an obstacle and has value 0 where there is no obstacle. In developed algorithm where obstacles are represented as rectangles and ellipses in a static environment, the circular window mentioned above is obtained from a two-dimensional array called savepoint which holds the information whether each cell is part of an obstacle or not. Next step is to build primary polar histogram. In order to do that, we need to calculate the vector magnitude $m_{i,j}$ of each cell.

$$m_{ij} = c_{ij}^{\,2}\left(a - bd_{ij}^{\,2}\right)$$

where, $c_{i,j}$ is the certainty value of active cell, $d_{i,j}$ is the distance from active cell to the RCP (Robot Center Point) and the parameters a, b are chosen according to the below given equation:

$$a - b\left(\frac{w_s - 1}{2}\right)^2 = 1$$

Based on the obstacle vectors, the primary polar histogram H^p is built. H^p has an angular resolution α so that $s = 360/\alpha$ is an integer, resulting in a fixed number of angular sectors over which the histograms are built. The method uses an analytically determined low-pass filter to compensate for the

width of the robot. Obstacle cells are enlarged by the $r_{r+s} = r_r + d_s$ where r_r is the robot radius and d_s is the minimum distance between the robot and the obstacle. With the obstacle enlarged by r_{r+s}, the robot can be treated as a point-like vehicle. The width compensation method is implemented efficiently by enlarging the obstacles while building the primary polar histogram. The equation to determine primary polar histogram can be calculated using the enlargement γ_{ij} angle, vector direction β_{ij} and polar obstacle density H_k^p calculated for each k-th sector.

$$\gamma_{i,j} = \arcsin\left(\frac{r_{r+s}}{d_{i,j}}\right); \quad \beta_{i,j} = \arctan\left(\frac{y_j - y_0}{x_i - x_0}\right)$$

$$H_k^p = \sum_{i,j \in C_a} m_{i,j}.h'_{i,j}$$

$$\text{where } h'_{i,j} = \begin{cases} 1, & \text{if } k \cdot \alpha \in \left[\beta_{i,j} - \gamma_{i,j}, \beta_{i,j} + \gamma_{i,j}\right] \\ 0, & \text{otherwise} \end{cases}$$

Here, x_0 , y_0 are present coordinates of the Robot Center point, and x_i , y_j are coordinates of the active cell $c_{i,j}$.

The result of this process as mentioned above is a primary polar histogram that takes into account the width of the robot. The *h'* function also serves as a low-pass filter and smoothes the primary polar histogram. Since the histogram is built around the current robot position, independent of its orientation, this first stage of the algorithm can be implemented very efficiently by the use of tables of the size w_s x w_s. Based on the primary polar histogram H^p and the two thresholds, τ_{low} and τ_{high}, a binary polar histogram(H^b) is built. The sectors of H^b are either blocked(1) or free(0). The binary polar histogram indicates which directions are free for a robot that can instantaneously change its direction of motion. At time n, the binary polar histogram are updated by the following rules.

$$H_{k,n}^b = \begin{cases} 1, \text{ if } H_{k,n}^p > \tau_{\text{high}}, \\ 0, \text{ if } H_{k,n}^p < \tau_{\text{low}}, \\ H_{k,n-1}^b, \text{otherwise} \end{cases}$$

The VFH+ method uses a simple, but closer approximation of the trajectory of most mobile robots. It assumes that the robot's trajectory is based on circular arcs(constant curvature curves) and straight lines. The maximum trajectory curvature of a mobile robot is often a function of the robot velocity. The faster the robot travels, the smaller the maximum curvature. The minimum steering radius can be zero for a differential drive robot if it has zero traversal speed otherwise it has a certain value which is fixed in our software. There is another parameter on the other hand which is called minDistance(the minimum distance between the robot and the obstacle) that can be varied between 1 and 10. It is used in the building of primary polar histogram. The next stage after binary polar histogram is the masked polar histogram that shows which directions of motions are possible. It is determined according to the following rules:

Determine φ_b. Set φ_r and φ_l equal to φ_b.

Here φ_l is left angle, φ_r is right angle, and $\varphi_b = \theta + \pi$ is defined as direction opposite to the current direction. The limit angles φ_l and φ_r are obtained by checking every active cell using following two conditions.

An obstacle cell blocks the directions to its right if:

Condition 1: $d_r^2 < (r_r + r_{r+s})^2$

An obstacle cell blocks the directions to its left if:

Condition 2: $d_l^2 < (r_l + r_{r+s})^2$

For every cell $C_{i,j}$ in the active window C_a with $c_{i,j} > \tau$

If $\beta_{i,j}$ is to the right of θ and to the left of φ_r, check condition 1. If condition is satisfied, set φ_r equal to $\beta_{i,j}$

If $\beta_{i,j}$ is to the left of θ and to the right of φ_l, check condition 2. If condition is satisfied, set φ_l equal to $\beta_{i,j}$.

If the robot's sensors are not very reliable, φ_r and φ_l could also be determined in a more stochastic way. Instead of comparing the cell certainity values to a threshold, one could build a polar histogram whose sector values indicate the certainity that a sector is blocked because of robot dynamics. The values for φ_r and φ_l could then be determined by applying a threshold to this histogram. As the first method is more efficient, the second method should only be applied if really necessary.

With φ_r, φ_l and the binary polar histogram, the *masked polar histogram* can be builded.

$$H_K^m = 0 \text{ if } H_K^b = 0 \ \textit{and} \ (k,\infty) \in \{[\varphi_r,\theta],[\theta,\varphi_l]\}$$

$$H_K^m = 1 \text{ otherwise}$$

Fourth stage is the selection of the new steering direction and is done according to the following rules: The masked polar histogram shows which directions are free of obstacles and which ones are blocked. However, some free directions are better candidates than others for new direction of motion. The VFH+ method first finds all openings in masked polar histogram and then determines a set of

candidate directions. A cost function that takes into account more than just the difference between the candidate and the target direction is then applied to these candidate directions. The candidate direction k_d with the lowest cost is then chosen to be the new direction of motion. In the case of a goal-oriented robot which is our case we get between one and three candidate directions for each opening in the masked polar histogram. Next we need to define an appropriate cost function that selects the new direction of motion φ_d.The cost function has three terms. The first term is responsible for goal-oriented behaviour while the second and the third terms make the mobile robot *commit* to a direction. The cost function is as follows:

$$g(c) = \mu_1 \, \Delta(c_i k_i) + \mu_2 \, \Delta\left(c_i \frac{\theta_a}{\alpha} \right) + \mu_3 \, \Delta(c_i k_{d,n-})$$

where,

$$\Delta(c_1, c_2) = \min\{|c_1 - c_2|, |c_1 - c_2 - s|, |c_1 - c_2 + s|\}$$

The first term of our cost function *g(c)* represents the cost associated with the difference of a candidate direction and the target direction. The larger this difference is, the more the candidate direction will guide the robot away from its target direction, and hence the larger the cost.

The second term represents the cost associated with the difference of a candidate direction and the robot's current wheel orientation. The larger the difference is, the larger the required change of the direction of motion.

The third term represents the cost associated with the difference of a candidate directionand the previously selected direction of motion. The larger the difference is, the larger the change of the new steering command.

The third term represents the cost associated with the difference of a candidate directionand the previously selected direction of motion. The larger the difference is, the larger the change of the new steering command.

Only the relationship between the three parameters are important. To guarantee a goal oriented behaviour, the following condition must be satisfied

$$\mu_1 > \mu_2 + \mu_3$$

If a smooth path is more important than variations in the steering commands, then μ_2 should be set higher than μ_3. If smoothness of the steering commands is more important, then μ_3 should be set

higher than μ_2. Experiments have shown that a good set of parameters for a goal-oriented mobile robot is:

$$\mu_1 = 5, \mu_2 = 2, \ \mu_3 = 2.$$

3.5 Local (Adaptive) Navigation

This technique with obstacle avoidance is proposed for mobile robots in which the dynamics of the robot are taken into consideration. The only information needed about the local environment is the distance between the robot and the obstacles in three specified directions and certainly this is supplied by sensors [27]. The navigation law is a first-order differential equation and navigation to the goal and obstacle avoidance are achieved by switching the direction angle of the robot. There are global and local navigation algorithms depending on the surrounding environment. In global navigation, the environment surrounding the robot is known and the path which avoids the obstacles is selected. In this thesis, potential field and vector field histogram algorithms are algorithms which implement global navigation. In the adaptive or as it is named local navigation algorithm that has been implemented for this thesis, the environment surrounding the robot is unknown, or partially known, and sensors are used to detect the obstacles whose geometry and location is unknown. The mobile robot position is represented by Cartesian coordinates and can move in three directions [27]. Figure 3.1 shows all aspects under which the robot encounters obstacles.
When obstacles are detected in thr-ee directions, and $d_l >= d_r$ as shown in Figure 1(a), the distance in the left direction is greater than in the right direction and the robot should steer to the left. Letting ϵ be the angle shown in Figure 3.1(a), the robot should turn to the left by $\frac{\pi}{2} - \varepsilon$ to avoid the obstacle, and when $d_l < d_r$, the robot should turn to the right by $\frac{\pi}{2} - \varepsilon$.

To determine optimal path the following navigation law is used $\dot{\theta}(t) = -\eta[\theta(t) - \theta^*(t)]$.
Embedding this avoidance behaviour into navigation control law, the desirable angle for avoiding obstacle $\theta_\alpha(t) = \theta(t) + \text{sgn}(d_l - d_r)(\frac{\pi}{2} - \varepsilon)$ where

$\varepsilon = \tan^{-1}\left(\frac{d\cos(\alpha) - d_c}{d \cdot \sin(\alpha)}\right)$, $d = \max(d_r, d_l)$ and $\text{sgn}(x) = \begin{cases} +1 & (x \geq 0) \\ -1 & (x < 0) \end{cases}$

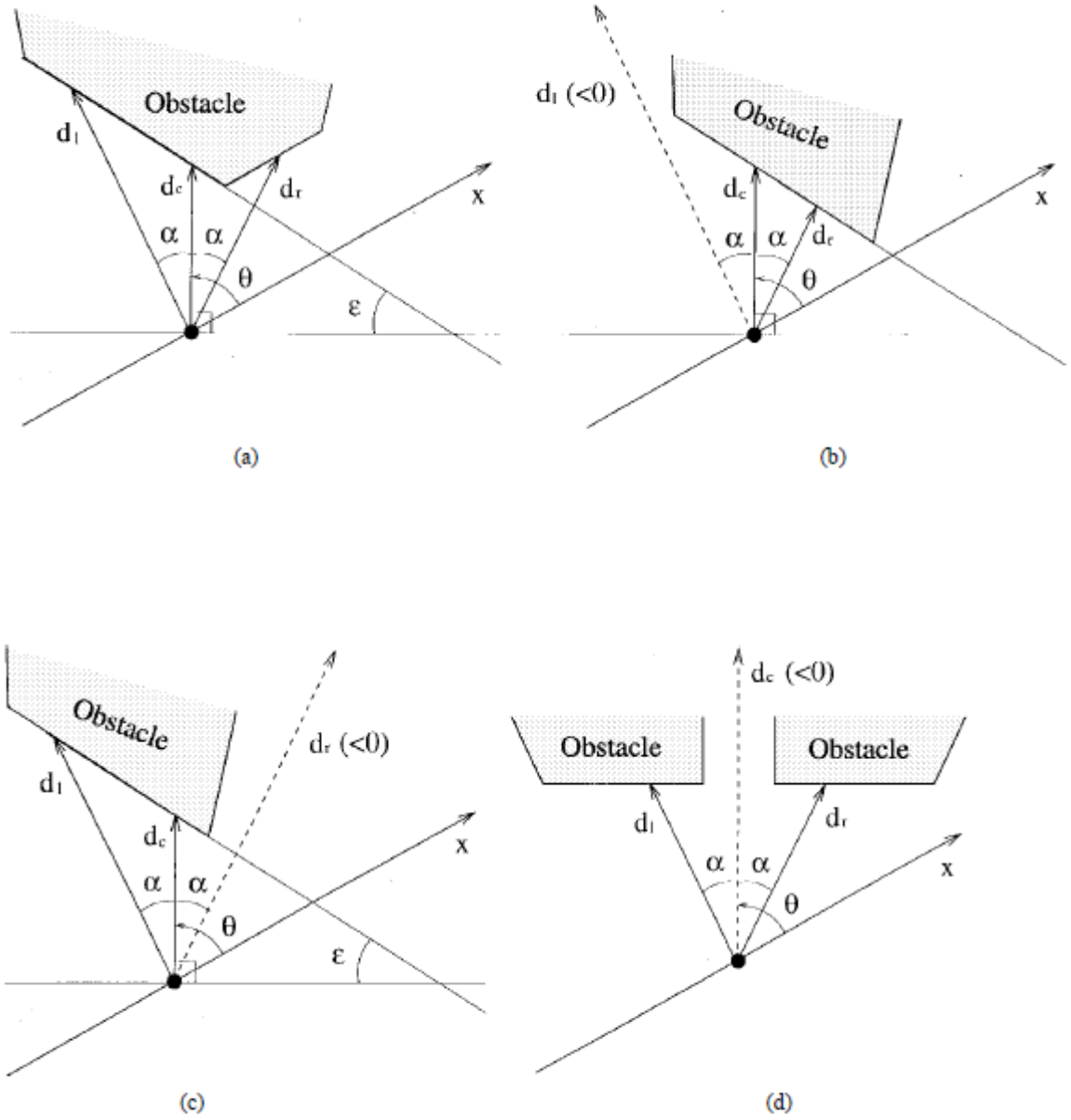


Figure 3.1: Avoiding behaviours of the mobile robot (a) Three positive detections($d_l >= d_r$), (b) Two positive detections, Case 1 ($d_c > d_r$)$\cos^{\alpha}$ (c) Two positive detections, Case 2 ($d_c < d_l$)$\cos^{\alpha}$, $d_r < 0$) (d) Two positive detections, Case 3 ($d_c < 0$).

In Figure 3.1(b)and (c) an obstacle is detected in front of the robot; that is, $d_c > 0$. In the former, $d_c > d_r\cos^{\alpha}$ and $d_l < 0$. This indicates that a collision-free space extends to the left and

$$\theta_{\alpha}(t) = \theta(t) + \mathrm{sgn}(d_l - d_r)(\frac{\pi}{2} - \varepsilon) + \mathrm{sgn}(d_l - d_r)\pi$$

In the latter, since $d_c < d_l \cos$ ∝ and $d_r < 0$, the right may be larger than the left margin. However, it may happen that an obstacle exists in the right side but is not detected at this instant. To assure obstacle avoidance, the robot should turn to the left by $(\frac{\pi}{2} - \varepsilon)$ as shown in Figure 3.1(c), and the desirable direction angle is given by (1). When $d_c < 0$, as shown in Figure 3.1(d), steering is not needed because there is no obstacle in the center direction. Therefore, the desirable direction angle is $\theta_\alpha(t) = \theta(t)$. A basic concept is that when obstacles are detected, priority must be given to avoidance behaviour over navigating to the goal. However, if the navigating action covers the avoidance behaviour, the robot is navigated to the goal according to

$$\theta_t = \frac{(goal's \text{ Y position - robot's Y position})}{(goal's \text{ X position - robot's X position})}$$

Then both θ_t (thetat) and θ_α (thetaalpha) are utilized to obtain the new direction of motion that the robot will take where thetat is adjusted repeatedly as below

if (thetat <= theta)
thetat = thetat + pi
else thetat = thetat – pi where theta is the angle at which the robot is travelling.

Three distance sensors are used which measure the distance between the robot and the obstacle up front(distcenter), the distance at an angle α towards the left from the center(distleft), the distance at an angle α towards the right from the center(distright). These are all used to calculate the new theta by the following algorithm:

```
if ((distcenter < 0)  and ((distleft > 0)  and(distright > 0)))
    thetanew = thetaalpha;
else if (distcenter > 0)
     if (((thetat > thetaalpha) and (thetaalpha > theta)) or ((thetat <= thetaalpha) and (thetaalpha <=
theta)))
          thetanew = thetat;
      else thetanew = thetaalpha;
   else if (((distleft < 0) and(thetat >= theta)) || ((distright < 0) or  (thetat <= theta)))
        thetanew = thetat;
```

else thetanew = thetaalpha;

theta = thetanew

Adaptive(Local) Navigation has given one of the fastest results in our simulation.

4. FUZZY NAVIGATION ALGORITHM AND OBSTACLE AVOIDANCE

4.1. Overview

It is usually difficult or impossible to obtain an accurate model of a dynamic environment when dealing with the navigation problem of mobile robot. During navigation in unknown environments, the robot can safely avoid from obstacles in real time. Nowadays fuzzy logic has become a means of collecting human knowledge and experience and dealing with uncertainties in the robot navigation. Its successful applications to a variety of systems have helped to attract growing attention and interest.

In this chapter the theoretical background of fuzzy system, its structure has been described. The functions of its main blocks are explained. The inference engine scheme and mathematical formulas used for decision making have been described.

4.2. Structure of Fuzzy System

The primary goal of control engineering is to distill and apply knowledge about how to control a process so that the resulting control system will reliably and safely achieve high-performance operation. Fuzzy logic provides a methodology for representing and implementing our knowledge about how best to control a process.

A block diagram of a fuzzy control system is shown in Figure 1. The fuzzy controller is composed of the following four elements:

1.A rule-base (a set of If-Then rules), which contains a fuzzy logic quantification of the expert's linguistic description of how to achieve good control.

2. An inference mechanism (also called an "inference engine" or "fuzzy inference" module), which emulates the expert's decision making in interpreting and applying knowledge about how best to control the plant.

3. A *fuzzification interface,* which converts controller inputs into infornıation that the inference mechanism can easily use to activate and apply rules.

4. A defuzzification interface, which converts the conclusions of the inference mechanism into actual inputs for the process.

A fuzzy system is a static nonlinear mapping between its inputs and outputs (i.e., it is not a dynamic system). It is assumed that the fuzzy system has inputs $u_i \in U_i$ where i = 1, 2, . . . , n and outputs $y_i \in Y_i$ where i = 1, 2, . . ., m, as shown in Figure 17. The inputs and outputs are "crisp"-that is, they are real numbers, not fuzzy sets. The fuzzification block converts the crisp inputs to fuzzy sets, the inference mechanism uses the fuzzy rules in the rule-base to produce fuzzy conclusions (e.g., the implied fuzzy sets), and the defuzzification block converts these fuzzy conclusions into the crisp outputs.

The ordinary ("crisp") sets U_i and Y_i are called the "universes of discourse" for u_i and y_i, respectively (in other words, they are their domains). In practical applications, most often the universes of discourse are simply the set of real numbers or some interval or subset of real numbers. Note that sometimes for convenience we will refer to an "effective" universe of discourse $[\alpha,\beta]$ where α and β are the points at which the outermost membership functions saturate for input universes of discourse, or the points beyond which the outputs will not move for the output universe of discourse.

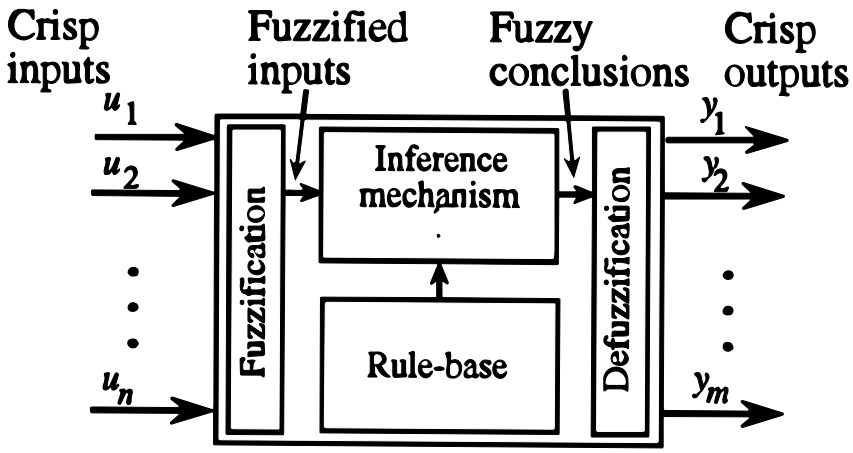


Figure 4.1. Structure of Fuzzy system

To specify rules for the rule-base, the expert will use a "linguistic description"; hence, linguistic expressions are needed for the inputs and outputs and the characteristics of the inputs and outputs. We will use "linguistic variables" to describe fuzzy system inputs and outputs. For our fuzzy system, linguistic variables denoted by $\tilde{u}$ are used to describe the

inputs u_i . Similarly, linguistic variables denoted by $\tilde{y}$ are used to describe outputs y_i. Linguistic values are generally descriptive terms such as "positive large," "zero," and "negative big" (i.e., adjectives).

The mapping of the inputs to the outputs for a fuzzy system is in part characterized by a set of *conditions action* rules, or in *modus ponens* (If Then) form,

If premise **Then** consequent (4.1)

Usually, the inputs of the fuzzy system are associated with the premise, and the outputs are associated with the consequent. These If-Then rules can be represented in many forms. Two standard forms, multi-input multi-output (MIMO) and multi-input single-output (MISO), are considered here. In the thesis the MISO form of a linguistic rule is considered

If ü$_1$ is $\hat{A}_1$ **and** ü$_2$ is $\hat{A}_2$ **and,** . . . , **and** ü$_n$ is $\hat{A}_n$ **Then** y is *B* *(4.2)*

It is an entire set of linguistic rules of this form that the expert specifies on how to control the system. Here $\tilde{A}_1,...,\tilde{A}_n$ and *B* are linguistic values.

It can be easily shown that the MIMO form for a rule (i.e., one with consequents that have terms associated with each of the fuzzy controller outputs) can be decomposed into a number of MISO rules using simple rules from logic. For instance, the MIMO rule with n inputs and m = 2 outputs

If ü$_1$ is $\hat{A}_1$ **and** ü$_2$ is $\hat{A}_2$ **and,** . . . , **and** ü$_n$ is $\hat{A}_n$ **Then** y_1 is B_1 *and* y_2 is B_2
is linguistically (logically) equivalent to the two rules

If ü$_1$ is $\hat{A}_1^j$ **and** ü$_2$ is $\hat{A}_2^k$ **and,** . . . , **and** ü$_n$ is $\hat{A}_n^l$ **Then** y_1 is B_1
If ü$_1$ is $\hat{A}_1^j$ **and** ü$_2$ is $\hat{A}_2^k$ **and,** . . . , **and** ü$_n$ is $\hat{A}_n^l$ **Then** y_2 is B_2 *(4.3)*

A mebership function is used in order to describe linguistic values in rule base. The membership function quantifies, in a continuous manner, whether values of *u* belong to (are members of) the set of values that are "possmall," and hence it quantifies the meaning of the linguistic statement "u is possmall". In fig.5 triangle form mebership function is shown.

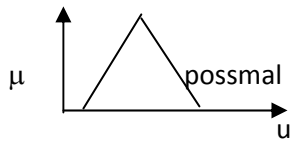
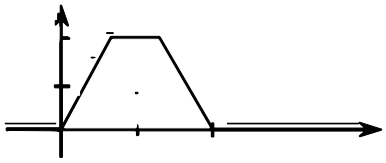


Fig. 4.2. Membership function

We could use other form membership function also: a bell-shaped function, a trapezoid, or many others.



a)Trapezoid

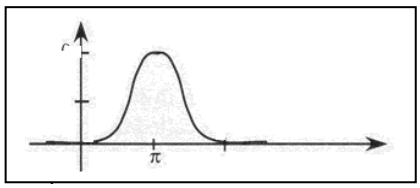


Fig.4.3. Forms of membership function

Depending on the application and the designer (expert), many different choices of membership functions are possible.

The rule-base of the fuzzy controller holds the linguistic variables, linguistic values, their associated membership functions, and the set of all linguistic rules**.**

In next section the above described structure fig.4.1 will be used for coctructing fuzzy inference mchanizm of robot navigation system.

4.3 Fuzzy Navigation

In robot navigation the fuzzy system focuses on the goal reaching, while keeping obstacle avoidance. Robot navigation includes goal seeking and obstacle avoidances behaviours. Robot uses goal seeking behaviour when no obstacles are detected and robot directly follows to the goal. Obstacle avoidance behaviours include left_turn, right_turn, backward and forward behaviours. The implementation of these behaviours are carried out using measured left, direct and right distances from the obstacles. These distances could be used to design fuzzy knowledge base. The knowledge base constructed using left, direct and right distances is given in appendix. In the work for designing knowledge base the left and right angles

between the line connecting robot and goal, and the line connecting robot and left and right sides of obstacle are used. Using left, direct and right distances the left and right angles between robot and obstacle are determined (fig.4.4). These angles are used in knowledge base for making decision by robot. The distance for detection of obstacle is taken as constant. Depend of considered environment it is set by programmer. This distance is called sense distance of robot. An example scenario that demonstrates obstacle avoidance is shown in Fig. 4.4. The mobile robot detects obstacle by measuring the distance between the robot and obstacle. Then local sensor detects the boundary of the obstacle in the in the local sensing region. The sensor region is determined by the user on the base of characteristics of the sensor used. It is impossible to know the shapes and sizes of all the obstacles outside of the sensing region. During avoidance of obstacle the boundary of the obstacle is further enlarged to ensure the safety of the robot. This expanded boundary is called the "safe boundary". This ensures the safety of the robot during avoidance of obstacles. Each obstacle will have two different boundaries, the "real boundary" and the "safe boundary", as shown in Fig. 1. The robot detects the real boundary of obstacles. Then the safe bondaries of obstacles are determined by controller calculations performed within the navigation algorithm. Both boundaries inside the local sensor region are available to the controller. Fig. 2 represents the scan distances to the obstacle boundaries shown in Fig. 4.4.

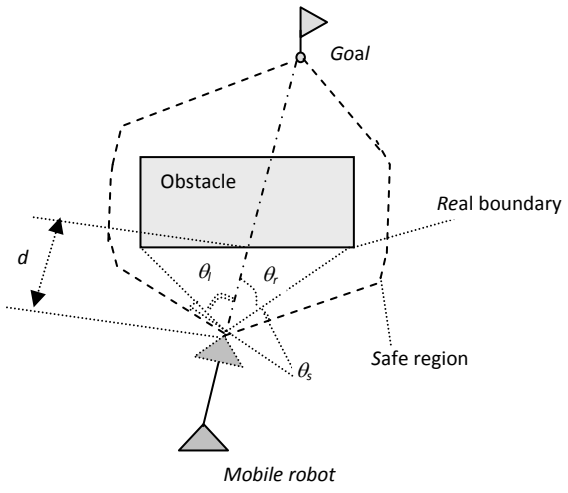


Fig. 4.4. Mobile robot avoiding obstacle

During navigation of robot the angle between the direct line connecting robot and target, and the line connecting robot and safe obstacle boundary are determined for the left and right

side of the robot. This is easily done by inspecting left and right side from the target direction in 60^o. After detecting the left and right boundaries of obstacles, the corresponding left $\theta_l(k)$ and right $\theta_r(k)$ angles are determined. Using these angles a decision is made by the robot. The basic strategy is to select the direction that has the smallest angle to the goal direction. Then the safe angle characterising safe region is used for final decision. The calculated new angle to avoid an obstacle will be

$$\theta(k)=F(\theta_l(k),\theta_r(k),\theta_s(k) \qquad (4.4)$$

$\theta_l(k)$ and $\theta_r(k)$ are angles based on the real boundary and the goal directions for left and right sides, respectively, and $\theta_s(k)$ is the safe angle. Table 1 shows the selection rule. In designing the fuzzy navigation algorithm the Gaussian membership functions are used. Meanwhile, it can also be assumed that there are five membership functions for each of the input linguistic variables θ_l and θ_r , respectively. The membership functions will be denoted as *VS* (Very Small), *S* (Small), *M* (Medium), *L* (Large), and *VL* (Very Large), respectively.

Table 1. The rule base

$\theta(k)$		$\theta_l(k)$				
		Very Small	Small	Medium	Large	Very Large
$\theta_r(k)$	Very Small	$\theta_{ls}(k)$	$\theta_{rs}(k)$	$\theta_{rs}(k)$	$\theta_{rs}(k)$	$\theta_{rs}(k)$
	Small	$\theta_{ls}(k)$	$\theta_{ls}(k)$	$\theta_{rs}(k)$	$\theta_{rs}(k)$	$\theta_{rs}(k)$
	Medium	$\theta_{ls}(k)$	$\theta_{ls}(k)$	$\theta_{ls}(k))$	$\theta_{rs}(k)$	$\theta_{rs}(k)$
	Large	$\theta_{ls}(k)$	$\theta_{ls}(k)$	$\theta_{ls}(k)$	$\theta_{ls}(k)$	$\theta_{rs}(k)$
	Very Large	$\theta_{ls}(k))$	$\theta_{ls}(k))$	$\theta_{ls}(k)$	$\theta_{ls}(k)$	$\theta_{ls}(k)$

With reference to Table 1, $\theta_{ls}(k)=\theta_l(k)+\theta_s(k)$ and $\theta_{rs}(k)=\theta_r(k)+\theta_s(k)$. Using this table the TSK (Takagi-Sugeno-Kang) fuzzy rule base [49,50] is organised. Since there are five membership functions for $\theta_l(k)$ and $\theta_r(k)$ respectively, there will be 25 fuzzy rules as shown in Fig. 3.

If $\theta_l(k)$ is "Very Small" and $\theta_r(k)$ is "Small" **then** $\theta(k) = \theta_{rs}(k)$

If $\theta_l(k)$ is "Very Small" and $\theta_r(k)$ is "Medium" **then** $\theta(k) = \theta_{rs}(k)$

If $\theta_l(k)$ is "Large" and $\theta_r(k)$ is "Small" **then** $\theta(k) = \theta_{ls}(k)$ *(4.5)*

. . . . . .

If $\theta_l(k)$ is "Very Large" and $\theta_r(k)$ is "Medium" **then** $\theta(k) = \theta_{rs}(k)$

Fig. 3. Fragment of the fuzzy rule base

The values of left and right angels $\theta_l(k)$ and $\theta_r(k)$ are defined with fuzzy linguistic values. Fig. 4.5 presents the linguistic values defined for left and right angles. The form of membership functions are accepted in triangle form.

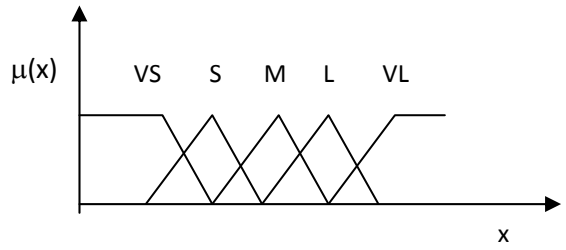


Fig. 4.5. Linguistic values

Membership functions of linguistic values are determined by the following formula.

$$\mu(x) = \begin{cases} 1 - \dfrac{\bar{x} - x}{\alpha}, & \bar{x} - \alpha \le x \le \bar{x} \\ 1 - \dfrac{x - \bar{x}}{\beta}, & \bar{x} < x \le \bar{x} + \beta \\ 0, \quad \text{in other case} \end{cases} \tag{4.6}$$

Also, using the distance from the obstacle and possibility of collision, the rule base for determining the value of velocity is constructed. Logic machine inference of fuzzy system is performed by the following formula.

$$\widetilde{X}1, \widetilde{X}2 \to \widetilde{Y} \quad \widetilde{\theta}_l, \widetilde{\theta}_r \to \widetilde{Y} \tag{4.7}$$

The membership degrees of the input signals for each active rules in knowledge base are defined. This is carried out in fuzzification block.

Fuzzy logic inference is performed using max-min composition of Zadeh [51].

$$\mu(y) = \max_{x1,x2,x3} \min\{\mu_{X1}(x1), \mu_{X2}(x2), \mu_R(x1, x2, y)\} \tag{4.8}$$

Using the "Center of average" method the defuzzification process of fuzzy output signal is performed.

$$y = \frac{\sum_{i=1}^{n} \mu(y_i) * y_i}{\sum_{i=1}^{n} \mu(y_i)} \tag{4.9}$$

The formulas (4.8) and (4.9) are used to determine output of fuzzy logic system.

4.4 Interval Type-2 Fuzzy Logic for robot navigation

The design of intelligent robots needs the considering of uncertainties inherent to unstructured environments, unreliable and incomplete perceptual information and imprecise actuators. In many applications the robot's environment changes with time in a way that is not predictable by the designer in advance. In addition, the knowledge about the environment is often imprecise and incomplete due to the limited perceptual quality of sensors. Fuzzy system employ a mode of approximate reasoning that makes them a suitable tool to implement a robust robot behaviour tolerating noisy and unreliable sensor information [52]. The implementation of fuzzy system in robot control are based on the traditional type-1 FLC. The type-1 FLCs have cannot fully handle or accommodate for the linguistic and numerical uncertainties associated with changing and dynamic environment because they use precise type-1 fuzzy sets [53,54]. Recently, Mendel and Karnik [55, 56] have developed a type-2 fuzzy logic system, this method can handles imprecise and uncertainty data to produce complex decision outcomes, it fuzzy sets let us model and minimized the effects of uncertainties.

In this section the development of type-2 fuzzy system for detection local obstacle and recognition an unknown environment for making navigation decision is considered.

Interval type-2 FLCs that use interval type-2 fuzzy sets is able to model and handle the uncertainties to give a good performance [53-56]. However, the interval type-2 FLC involves a computational overhead with the type-reduced fuzzy sets using Karnik-Mendel procedure which can reduce the robustness preformance of the type-2 FLC especially when operating on embedded platforms. Wu and Mendel [57] introduced the type reduced inner and outer bound sets, thus avoiding the use of Karnik-Mendel procedure.

An interval type-2 fuzzy set $\widetilde{F}$ can be written as follows:

$$\widetilde{F} = \int_{x \in X} \left[\int_{u \in \left[\underline{\mu}_{\widetilde{F}}(x), \overline{\mu}_{\widetilde{F}}(x) \right]} 1/u \right] / x \tag{4.10}$$

Where $\underline{\mu}_{\widetilde{F}}(x), \overline{\mu}_{\widetilde{F}}(x)$ epresent the upper and lower membership functions (MFs) respectively.

The type-1 is extended to an interval type-2 fuzzy system by adding uncertainties in both antecedent and consequent part of each rule The MFs values are spread for the antecedent part by +-A% , and for the consequent part by +-C%. For each input k and rule i, MF is represented by triangular membership function with uncertain mean.

The upper and lower MFs for this interval type-2 fuzzy set can be written in equation (2) and (3) respectively.

$$\bar{f}_A(x) = \begin{cases} 0, & x < l_1 \\ \dfrac{x - l_1}{p_1 - l_1}, & l_1 \le x < p_1 \\ 1, & p_1 \le x \le p_2 \\ \dfrac{r_2 - x}{r_2 - p_2}, & p_1 < x \le r_2 \\ 0, & x > r_2 \end{cases} \quad ; \quad \underline{f}_A(x) = \begin{cases} 0, & x < l_2 \\ \dfrac{x - l_2}{p_2 - l_2}, & x \le \dfrac{r_1(p_2 - l_2) + l_2(r_1 - p_1)}{(p_2 - l_2) + r_1 - p_1)} \\ \dfrac{r_2 - x}{r_2 - p_2}, & x > \dfrac{r_1(p_2 - l_2) + l_2(r_1 - p_1)}{(p_2 - l_2) + r_1 - p_1)} \\ 0, & x > l_2 \end{cases} \tag{4.11}$$

In this research, we use the type-2 singleton fuzzifier. The upper $\overline{\mu}_{\widetilde{F}}(x)$ and lower $\underline{\mu}_{\widetilde{F}}(x)$ membership values are calculated using (2) and (3) respectively.

Using type-2 fuzzy membership function the fuzzy type-2 TSK rule base is designed for robot. Te rules in this rule base has the following form..

The rules used in this paper are such that the consequent parts are of TSK type, as indicated below.

$$\text{IF } x_1 \text{ is } \widetilde{A}_{j1} \text{ and } x_2 \text{ is } \widetilde{A}_{j2} \text{ and } \ldots \text{ and } x_m \text{ is } \widetilde{A}_{jm} \text{ THEN } y_j = B_j \tag{4.12}$$

where $x_1, x_2, \ldots, x_m$ are the input variables, y_j $(j=1,..,n)$ are the output variables, $\widetilde{A}_{ij}$ are type-2 membership functions for the j-th rule of the i-th input defined as a Gaussian membership function, B_j $(i=1,..m, j=1,\ldots,n)$ are parameters.
The firing strength of the j^{th} rule is an interval type-2 set determined by the left most point $\underline{f}_i$ and its rightmost point $\bar{f}_i$ which is calculated as follows:

$$\begin{aligned} \underline{f} &= \underline{\mu}_{\widetilde{A}_1}(x_1) * \underline{\mu}_{\widetilde{A}_2}(x_2) * \ldots * \underline{\mu}_{\widetilde{A}_n}(x_n) \\ \overline{f} &= \overline{\mu}_{\widetilde{A}_1}(x_1) * \overline{\mu}_{\widetilde{A}_2}(x_2) * \ldots * \overline{\mu}_{\widetilde{A}_n}(x_n) \end{aligned} \tag{4.13}$$

The inference engine computes the degree of firing of each rule by using the operation under the product t-norm between the antecedent membership grades of each rule. The defuzzification layer approximates the type-reduced set using Wu-Mendel Uncertainty Bounds. It provides mathematical formulas for the inner and outer bound sets which can be used to approximate the type-reduced set . Equations (6) and (7) define the inner bound set while (8) and (9) define the outer bound set.

$$\bar{y}_l = \min\left\{\frac{\sum_{i=1}^{M} \underline{f}_i w_i^l}{\sum_{i=1}^{M} \underline{f}_i} + \frac{\sum_{i=1}^{M} \overline{f}_i w_i^l}{\sum_{i=1}^{M} \overline{f}_i}\right\}; \tag{4.14}$$

$$\underline{y}_r = \max\left\{\frac{\sum_{i=1}^{M} \underline{f}_i w_i^r}{\sum_{i=1}^{M} \underline{f}_i} + \frac{\sum_{i=1}^{M} \overline{f}_i w_i^r}{\sum_{i=1}^{M} \overline{f}_i}\right\} \tag{4.15}$$

$$\underline{y}_l = \bar{y}_l - \left[\frac{\sum_{i=1}^{M} (\overline{f}_i - \underline{f}_i)}{\sum_{i=1}^{M} \bar{f}_i \sum_{i=1}^{M} \underline{f}_i} \cdot \frac{\sum_{i=1}^{M} \underline{f}_i (w_i^l - w_1^l) \sum_{i=1}^{M} \overline{f}_i (w_M^l - w_i^l)}{\sum_{i=1}^{M} \underline{f}_i (w_i^l - w_1^l) + \sum_{i=1}^{M} \bar{f}_i (w_M^l - w_i^l)}\right] \tag{4.16}$$

$$\bar{y}_r = \bar{y}_r + \left[\frac{\sum_{i=1}^{M} (\bar{f}_i - \underline{f}_i)}{\sum_{i=1}^{M} \bar{f}_i \sum_{i=1}^{M} \underline{f}_i} \cdot \frac{\sum_{i=1}^{M} \underline{f}_i (w_i^r - w_1^r) \sum_{i=1}^{M} \bar{f}_i (w_M^r - w_i^r)}{\sum_{i=1}^{M} \underline{f}_i (w_i^r - w_1^r) + \sum_{i=1}^{M} \bar{f}_i (w_M^r - w_i^r)} \right] \tag{4.17}$$

The crisp outputs in defuzzification layer can be computed as follows:

$$y(\bar{x}) = \frac{\frac{\underline{y}_l + \bar{y}_l}{2} + \frac{\underline{y}_r + \bar{y}_r}{2}}{2} \tag{4.18}$$

The robot behaviours was divided into obstacle avoidance, goal seeking and emergency condition behaviours. Linguistic for obstacle distance is represented by fuzzy set smal, medium, large. Output fuzzy set is represented by steering angle with linguistic are small, medium, large as shown in Fig 2c.

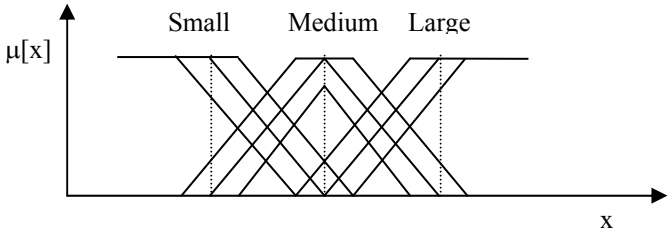


Figure 4.6. Membership function for obstacle distance

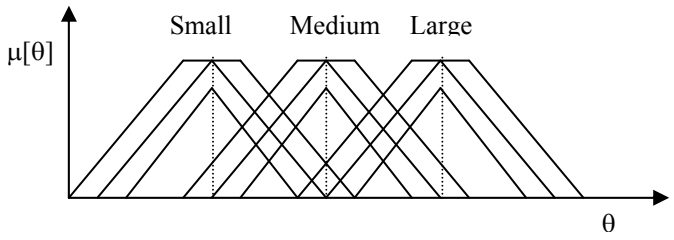


Figure 4.7. Membership function steering angle

When the robot is very close to the target, the attractive force between the robot and the target causes the robot seeking towards the target. Similarly when the robot is very close to an obstacle, because of repulsive force developed between the robot and the obstacle, the robot must change its speed and heading angle to avoid the obstacle .

5. DEVELOPMENT OF SIMULATOR FOR ROBOT NAVIGATION ALGORITHMS

5.1. Overview

In this chapter design and simulation of navigation algorithms were given. Fig. 5.1 shows the navigation algorithms implemented. The software package has been developed for implementation of navigation algorithms on the PC, using visual C# in Windows environment. The flowchart of fuzzy navigation has been described. The steps of constructions of navigation algorithms are given. Simulation of navigation algorithms is followed by comparative results demonstrating the advantages of fuzzy navigation.

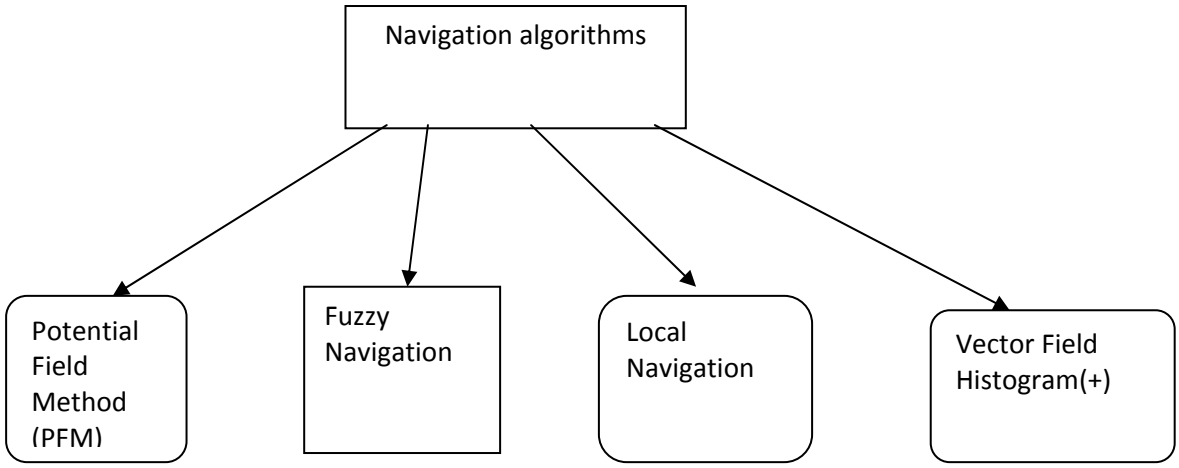


Fig.5.1. Navigation algorithms

5.2. Design of Robot navigation algorithms

5.2.1 Steps in the Potential Field Based Algorithm

Objective of Potential Field Based Algorithm is tha following. Given an object, a source location and orientation, a destination location and orientation and a description of spatial occupancy by objects, obtain a path for the object, its orientation along the path, and its position along the path as a function of time. The following steps are important for implementation of potential fiel algorithm.

1. Assign a potential function to the obstacles.
2. Find minimum potential valleys (MPV).
3. Select the best candidate path from MPV.
4. Assign initial orientation of the moving object along the selected path.
5. Use the find path algorithms to modify the path and orientations.
6. Repeat 3-5 until a solution is found.

5.2.2 The Vector Field Histogram algorithm

The VFH, permits the detection of unknown obstacles and avoids collisions while simultaneously steering the mobile robot toward the target. The VFH method uses a two-dimensional Cartesian *histogram grid* as a world model. This world model is updated continuously with range data sampled by on-board range sensors. The VFH method subsequently employs a *two-stage* data-reduction process in order to compute the desired control commands for the vehicle. In the first stage the *histogram grid* is reduced to a one dimensional *polar histogram* that is constructed around the robot's momentary location. Each sector in the *polar histogram* contains a value representing the *polar obstacle density* in that direction. In the second stage, the algorithm selects the most suitable sector from among all *polar histogram* sectors with a low *polar obstacle density*, and the steering of the robot is aligned with that direction. This method, named the *vector field histogram* (VFH), permits the detection of unknown obstacles and avoids collisions while simultaneously steering the mobile robot toward the target.

The VFH method includes the following basic steps:

1. A two-dimensional Cartesian *histogram grid* is continuously updated in real-time with range data sampled by the on-board range sensors.
2. The *histogram grid* is reduced to a one-dimensional *polar histogram* that is constructed around the momentary location of the robot. The *polar histogram* is the most significant distinction between the VFF and the VFH method as it allows a spatial interpretation (called *polar obstacle density*) of the robot's instantaneous environment.
3. Consecutive sectors with a *polar obstacle density* below threshold are called "*candidate valleys*." The *candidate valley* closest to the target direction is selected for further processing.
4. The direction of the center of the selected *candidate direction* is determined and the steering of the robot is aligned with that direction.

5. The speed of the robot is reduced when approaching obstacles head-on.

The characteristic behavior of a VFH-controlled mobile robot is best described as follows: The response of the vehicle is dependent on the *likelihood for the existence of an obstacle*. In the *histogram grid*, this likelihood is expressed in terms of size and *certainty values* of a cluster. This information is transformed into height and width of an elevation in the *polar histogram*. The strength of the VFH method lies thus in its ability to maintain a statistical obstacle representation at both the *histogram grid* level as well as at the intermediate data level, the *polar histogram*. This feature makes the VFH method especially suited to the accommodation of inaccurate sensor data, such as that produced by ultrasonic sensors, as well as sensor fusion.

5.2.3 The Local Navigation algorithm

The local navigation problem take into consideration the dynamics of the robot. In this algorithm the goal which the robot should reach is known, but the geometry and the location of the obstacles are unknown. The robot can move in three directions, forward, left or right, and has sensors to detect the obstacles in real-time in these directions. The direction angle of the robot's movement is determined by the direction angle of the goal and sensor signals. The navigation law is given by a first-order differential equation, by means of which the goal can be reached while avoiding the obstacles. The condition for obstacle avoidance is based on measuring the distances from the obstacles. Here local information is required for obstacle avoidance and the navigation law is simpler and flexible than ones using the artificial potential field an vector field histogram methods. The steps of local navigation are the followings.

1. On the base of input data determinig initial navigation angle of robot.
2. Testing the presency of obstacle. If obstcle is not detected continue else go to step 3.
3. Test condition for obstacle avoidance for distance d_{max} [27]
4. Using d_l left and d_r right distances determine new avoidance angle $\theta(t)$ from the obstacle and moving to the new position.
5. Using avoidance angle $\theta(t)$ determining desirable direction angle $\theta_\alpha(t)$ for new position .
6. Repeat steps 2-5 until reach the goal.

5.2.4 Fuzzy navigation algorithm algorithm

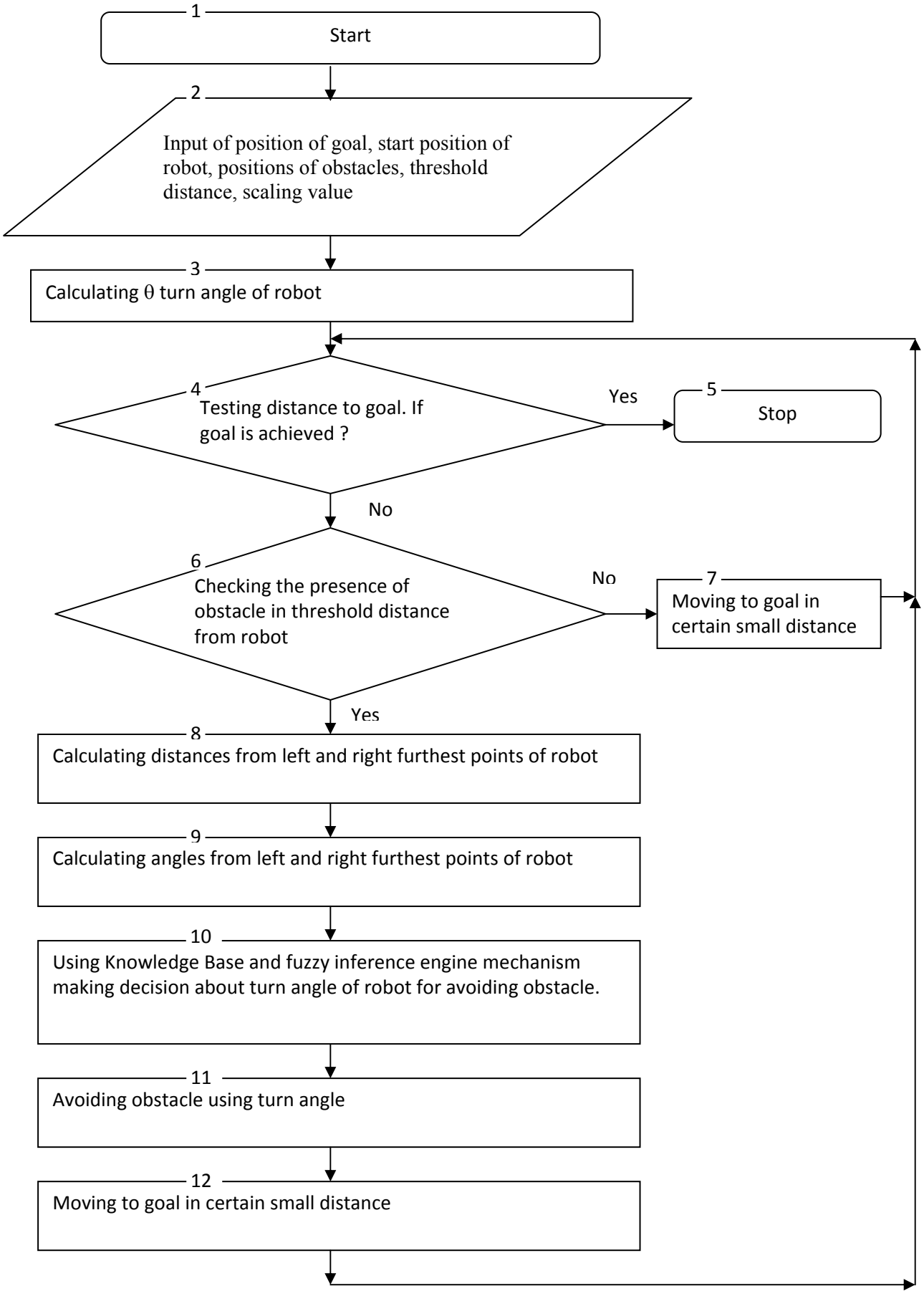


Fig.5.1. Flowchart of fuzzy navigation

Fig.5.2 demonstrates the flowchart of fuzzy navigation algorithm. Block 1 is the start. Block 2 is used to input position of goal, start position of robot, positions of obstacles, threshold distance, scaling value. Block 3 is used for calculating θ turn angle of robot using position of goal, start position of robot. After determining θ turn angle robot starts to move towards goal. An each certain small distance moved by the robot it test the distance from the goal in block 4. If goal is achieved robot stop moving (block6). In other case robot check the presence of obstacle in the distance defined by programmer which is called threshold distance of sensor. If no obstacle is detected robot moves forward towards to goal usind certain small distance (block 7). In other case robot try to avoid from the obstacle. For this navigation system of robot measure the distance from left and right furthest points of robot (block 8). The safe distances are added to find left and right distances for obstacle avoidance. Using these distances, the angles from left and right furthest points of robot is calculated (block 9). In the thesis for this operation the cosines theorem is used. These left and right angles were used as input for knowledge base. Using Knowledge Base robot is making decision about turn angle of robot for avoiding obstacle. The decision is made using fuzzy inference engine scheme (block 10). Using output of inference engine robot change direction according turn angle and avoid from obstacle (block 11). After avoiding obstacle robot moves to robot using certain small distance (block 12). During moving robot in each certain small distance test achieving the goal and presence of obstacles [49,58,59].

5.3 Simulator Software

The software package has been developed for implementation of navigation algorithms on the PC, using Visual C# in Windows environment. The selection of a navigation algorithm and algorithm parameters can be observed and modified via user interface. The program menu includes File, Edit, Draw, Simulate, Settings, Parameters, and Help. Under the **File** menu, there are the options New, Open, Save, Save As, and Quit that will allow users to save the configuration to a file and later retrieve it. Under the **Edit** Menu, there is Cut, Copy, Paste, Cancel, and Clear.

The procedure which was followed to develop the software was to first design a GUI based user interface where one can draw obstacles of standard shapes (rectangles and ellipses) and sizes using the **Draw** menu.

The next step was to develop a user interface under the **Settings** menu where the user can enter the starting and ending coordinates of the path of the mobile robot. Under this menu, there is also a dialog box where users can enter the unit size of the grid and the number of grid lines along the x axis and the number of grid lines along the y axis as well as an interface which allows to adjust the linelength between 0.1 and 1 (linelength is the amount of length on the path undertaken before the algorithm re-checks the surroundings).

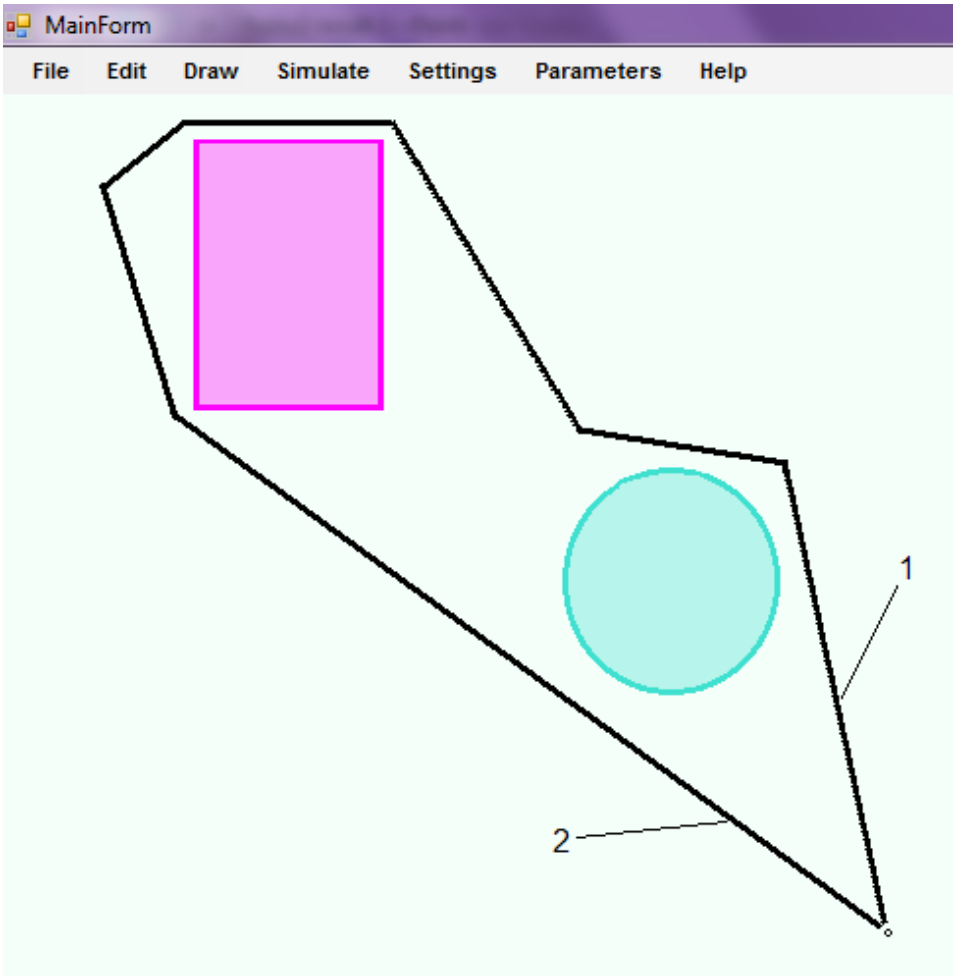
Under **Simulate** menu, users are given the option to run potential field algorithm, adaptive navigation algorithm, vector field histogram plus algorithm, or fuzzy algorithm on the same configuration. One of the nice points about the simulator is that the four algorithms can be run at the same time and this enables users to compare the results and learn more about the differences between the algorithms. The software also lets the user know about the running times of each algorithm and the number of points taken to reach the goal.

A pull-down **Parameters** menu is created which allows the user to change the appropriate parameters for each algorithm. For potential field method, the parameters are number of iterations and gridSize (the length of each grid cell). For adaptive navigation, these parameters are alpha and dmax. For vector field histogram plus, the parameters are workspace size (w_s), b, alpha (the meaning of which is different from the alpha in adaptive navigation). Note the fact that these are the parameters that can be changed by the user, not necessarily the only parameters within the algorithms.

5.4 Simulation Results

The developed robot navigation software includes the implementations of four algorithms. These are Potential Field Method (PFM), Local Navigation (LN), Vector Field Histogram Plus (VFH+) and Fuzzy Navigation (FN). During a typical run of the program using the GUI interface, different obstacles can be drawn, and start and goal positions of the robot can be specified. Then, an algorithm is selected and path planning or navigation of the mobile robot is performed. In all of the software runs below with algorithms, the gridSize has been taken as 1.0 and linelength has been taken as 1. The number of points generated indicates the total number of points required to reach the goal with the selected linelength. That is, the array that contains the path points has this number of members.

Before runing the algorithms we test fuzzy navigation algorithm using two different knowledge bases(KB): KB using angle measures (left and right angels) and KB using distance measures (left and right distances). In our similation KB using angle measures have given better results than other KB. Fig.4 demonstrate graphical simulation results. Table 2 demonstrates test results of FN algorithm with two different knowledge bases.

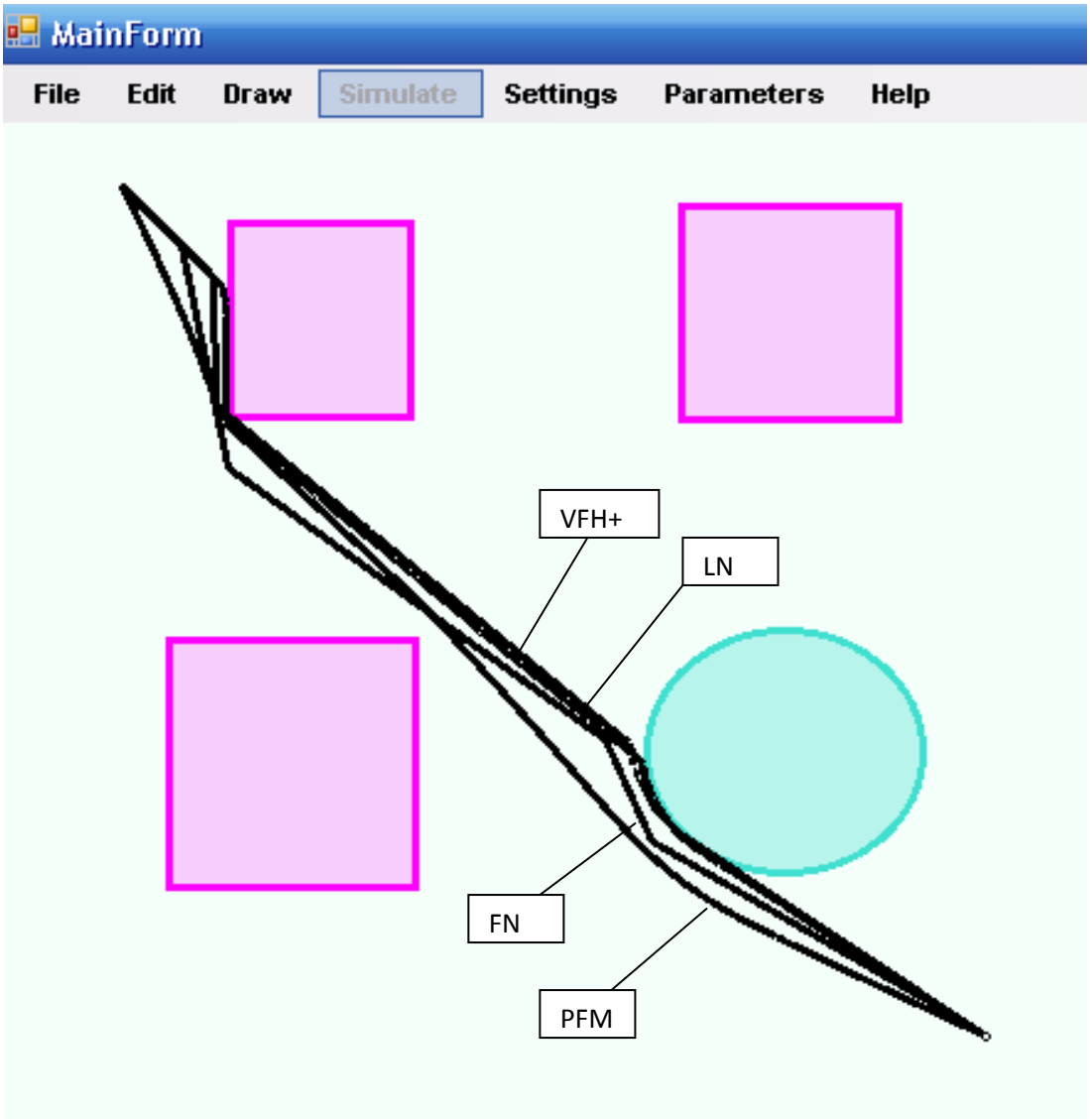


[Fig. 4 Simulation result of Fuzzy navigation algorithm using two different KBs: 1-KB using distance measures, 2- KB using angle measures]

Table 2 Statistics for Figure 4

Algorithm	Time taken to reach goal (ms)	Number of points generated
KB with distance measures	46	720
KB with angle measures	38	674

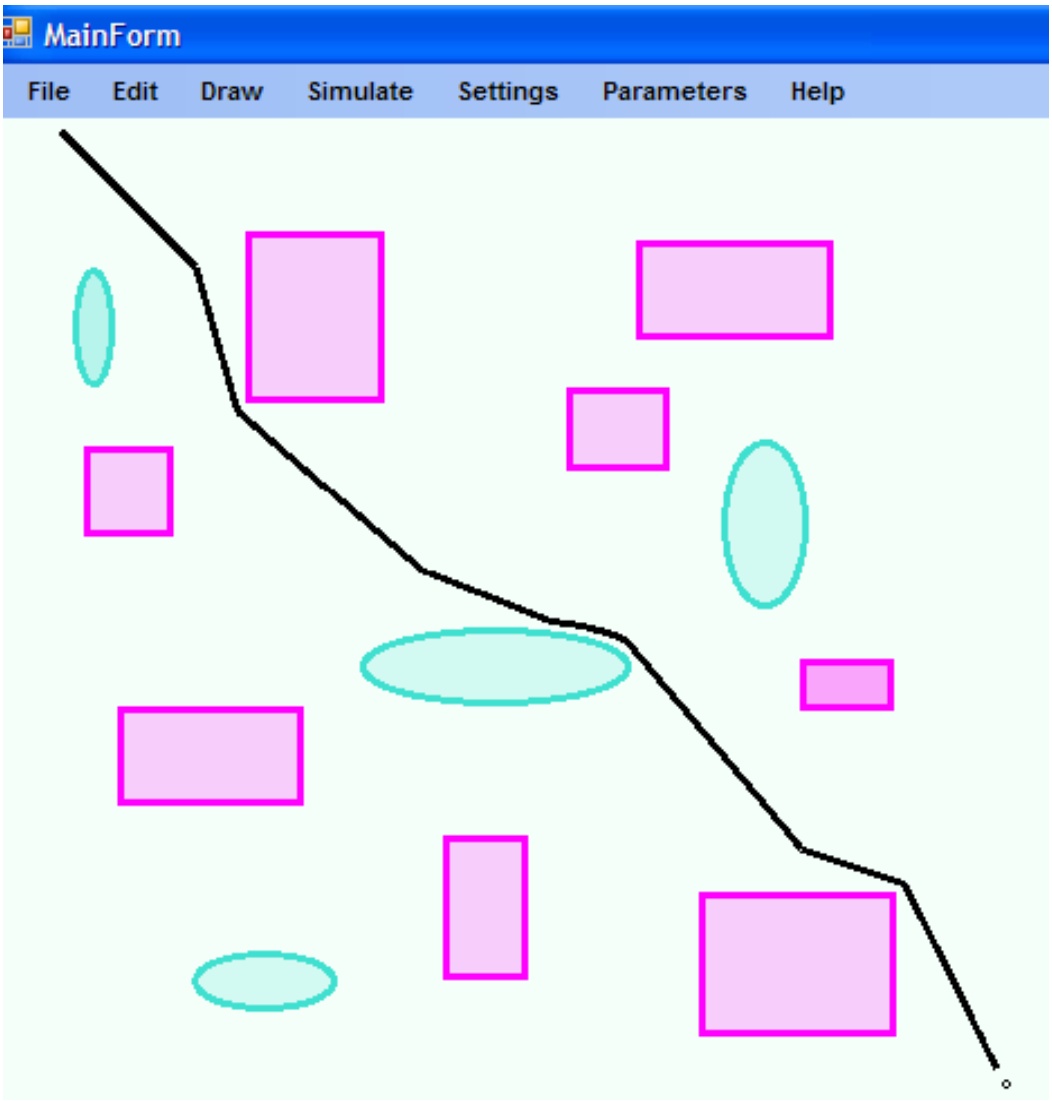
Fig.5 depicts a typical run of the Robot navigation software with only two obstacles. As can be observed from the figure, both obstacles were avoided successfully by the robot using all the algorithms. Local navigation and Vector Field Histogram (plus) algorithms gave similar results while Potential Field seemed to stay away from the obstacles. The Fuzzy algorithm on the other hand successfully avoided the obstacles making decisions of where to change direction as it approached the obstacles. As has been stated in the theoretical explanation, in Fuzzy algorithm the decisions are made according to angle to the left and to the right of the robot with respect to the obstacle. The fastest algorithm was found to be the fuzzy algorithm (see Table 3).



[Fig. 5 Obstacle avoidance with two obstacles]

Table 3 Statistics for Figure 4

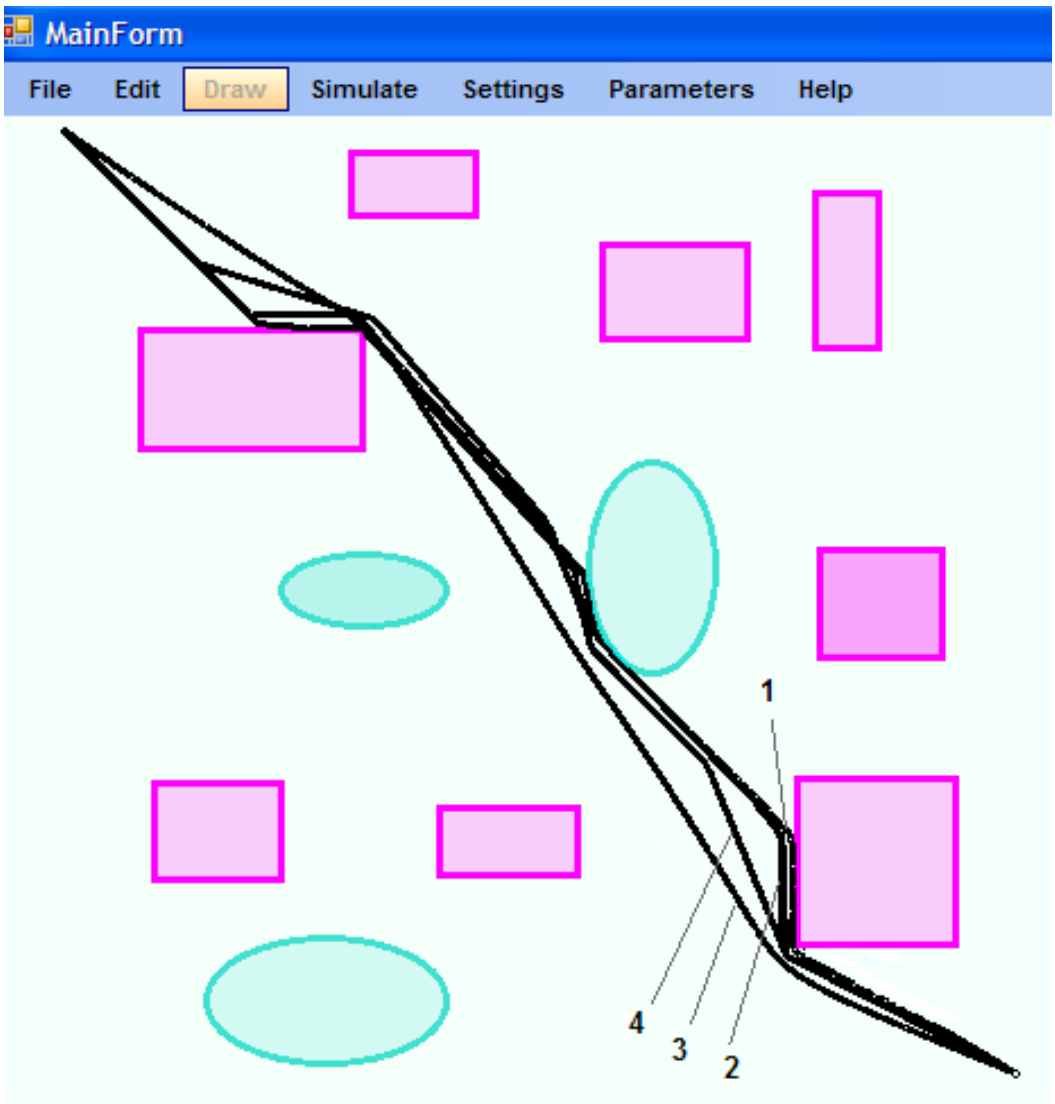
Algorithm	Time taken to reach goal (ms)	Number of points generated
PFM	24950	5806
VFH+	5062	5823
LN	40	5816
FN	31	5780



[Fig. 6 Simulation result of Fuzzy navigation algorithm]

Fig. 6 is a typical run of the Robot Navigation software with obstacles, using only Fuzzy navigation. As can be observed from the figure, all three obstacles were avoided successfully by the robot. The fuzzy algorithm avoided the obstacles by making decisions where to

change direction as it approached the obstacles. Fuzzy rule based with fuzzy inference mechanism is applied to find turning angle of the robot. As it is seen from the figure robot uses safe region during avoidance of obstacles. Here, as has been explained for Figure 5, both rectangle-shaped and ellipse-shaped obstacles are avoided. As has been stated in the theoretical explanation of the algorithm in section 2.4, the decisions are made according to angle to the left and to the right of the robot with respect to the obstacle.



[Fig. 7 Obstacle avoidance with three obstacles, 1-LN, 2-VFH+, 3-PFM, 4-FN]

In Fig. 7, three obstacles are used with all four algorithms. The results in terms of time taken to reach goal are different compared to the case shown in Fig. 5. The number of points generated in all cases is shown in Table 4. As shown, the number of points obtained for local

navigation is lower than with PFM and with VFH+. The best result was again obtained with fuzzy navigation algorithm. It is interesting to note that there are very large differences in the time taken to reach goal and the number of points generated between the fuzzy algorithm and other algorithms. The superiority of the fuzzy algorithm is probably because of its resemblance to the intelligent human reasoning and decision making process.

Table 4 Statistics for Figure 7

Algorithm	Time taken to reach goal (ms)	Number of points generated
PFM	28562	5574
VFH+	8859	5662
LNM	52	5624
FN	41	5594

5.3 Parameter Analysis on the Project

The last simulation is shown in Fig. 8 and Fig. 9 where the parameters are changed from their default values. The first figure shows the results of the three algorithms before the parameters were changed and second figure shows the situation after the parameters were changed. The values of the parameters are shown underneath the figure. What has happened is that in potential field method the got out of line as the gridSize was increased to 5.00 from 1.00 and the time increased twofold as the number of iterations was increased from 1000 to 1500. For Vector Field Histogram plus the workspace size(w_s) was increased from 12 to 22 which dramatically increased the ***time taken to reach goal*** from 11325 msecs to 24070 msecs. The fact that we increased minDistance from 1 to 10 has taken the path away from the obstacles. The other parameters for VFH+ remained constant (namely b and alpha). Finally there is the Local Navigation parameters where the parameter d_{max} has been increased from 4 units to 8.5 alpha for local navigation had to be increased anyways from 30 to 85 degrees to fix the path. The change was again slight movement of the path away from the obstacles.

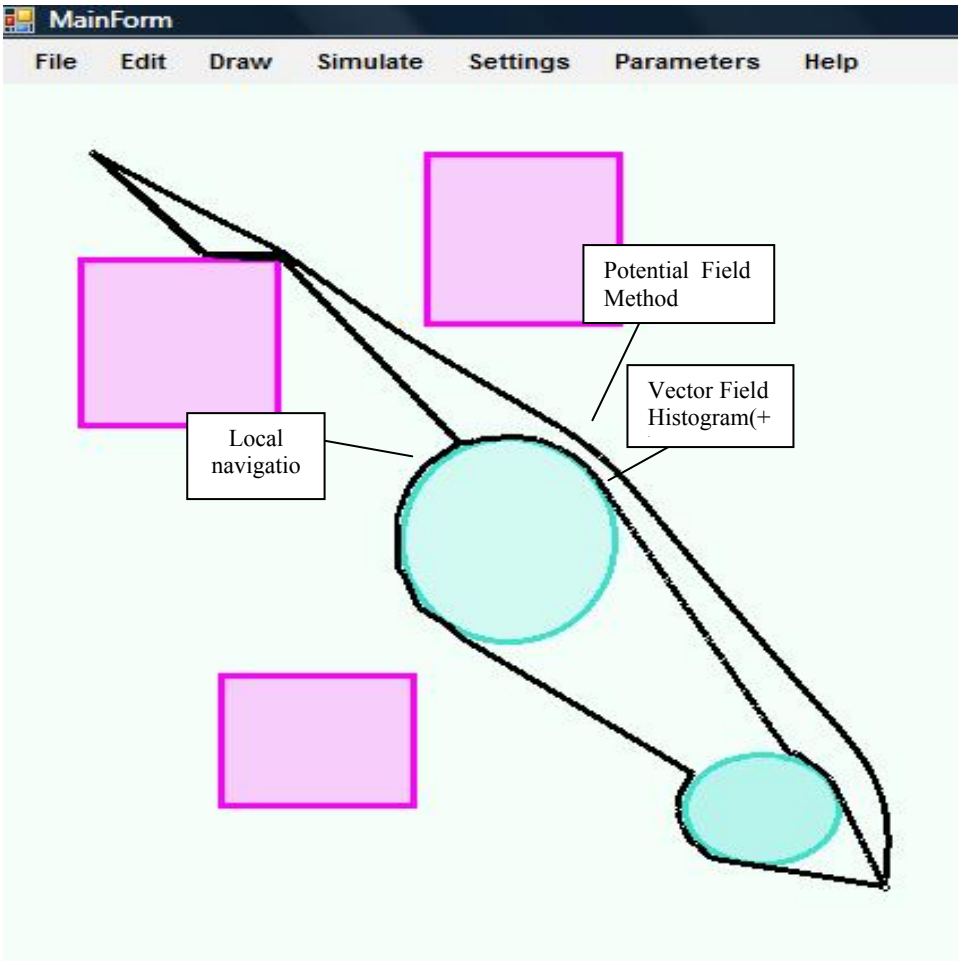


Fig. 8: Graph before parameters were changed

(The parameters are:

PFM: Number of Iterations: 1000, gridSize = 1.00

VFM(+): w_s = 12, minDistance = 1, b = 1.5, alpha = 2.0

LNM: d_{max} = 4 units, alpha = 85 degrees(It had to be fixed from 30 degrees to get the correct path), linelength=0.1)

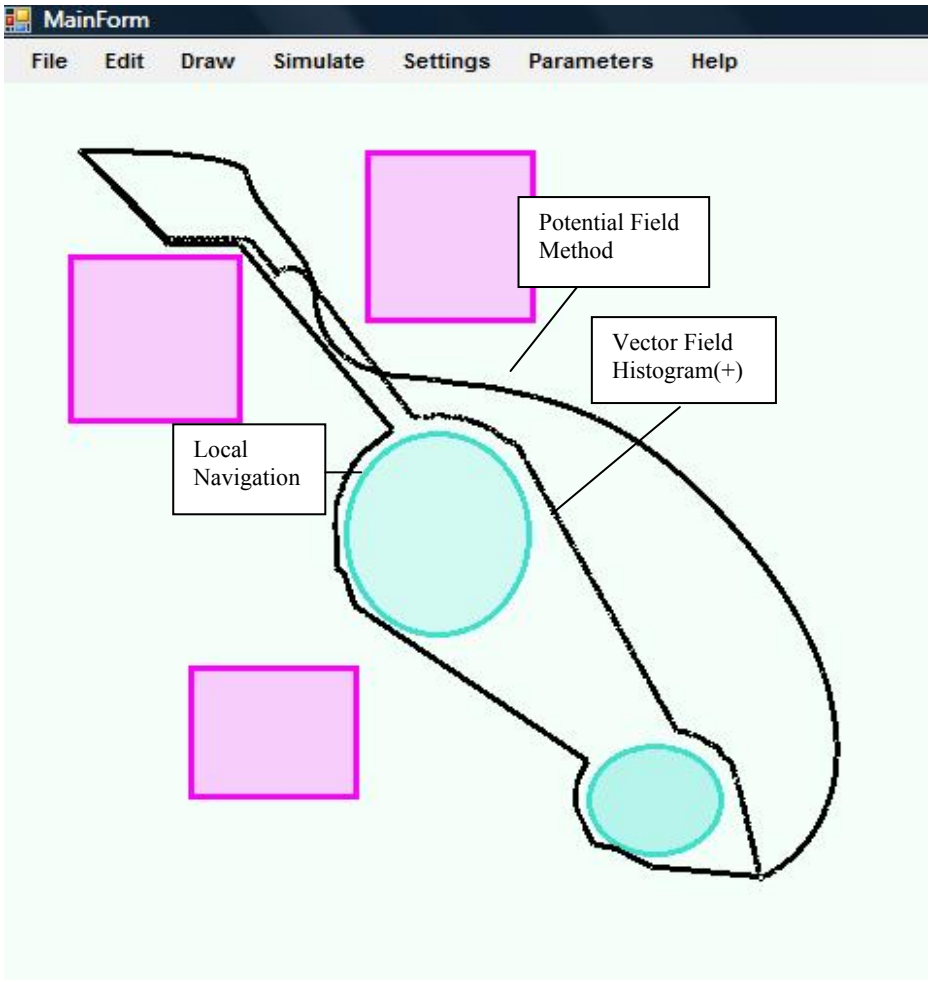


Fig. 9. Graph after parameters are changed

(The parameters are:

PFM: Number of Iterations: 1500, gridSize = 5.00

VFH(+): w_s = 22, minDistance = 10, b = 1.5, alpha = 2.0

LNM: d_{max} = 8.5 units, alpha = 85 degrees(It had to be fixed from 30 degrees to get the correct path), linelength=0.1)

Table 6 Statistics for Figure 5

Algorithm	Time taken to reach goal (ms)	Number of points generated
PFM	45396	6436
VFH+	24070	5779
LNM	1544	6465

5.5. Case Study

A pilot study was carried out at the Near East University to find out the opinions of students to using the EDURobot simulation program. The study consisted of carefully prepared questionnaire, completed by undergraduate students who used the EDURobot simulation software as part of their normal lecture sessions during one month. The aim of the questionnaire was to learn the opinions of students about the usefulness of the system.

Fourty-four questions from eight subjects were prepared using the Likert-5 scale. An extract from the questionnaire is shown in Table 7. The participants were given time to complete the questionnaires at the end of their training. Training includes the presentation of theoretical and practical sections. In theoretical section the navigation problem of mobile robots, the methodologies used for navigation were explained. After the theoretical section students attended their laboratory sessions. In these laboratories the EDURobot simulation software was used with different parameters. After completion of practical section the evaluation of the simulator software was done by students by completing the questionnaire. The results of the questionnaire were analysed using the SPSS statistical package and below is a summary of the important outcomes:

- A majority of students (85%) strongly agree that the EDURobot simulation software is easy to use and is user friendly.
- Over 90% of students agreed that the effects of parameter changes can easily be observed.
- A majority of students (89%) agree that the simulator has helped them to understand the functional differences between the three different algorithms.
- Over 94% of students agree that they were able to thoroughly understand the PFM, VFH+ and LNM algorithms.
- Finally, all of the students agree that a computer simulation is an effective way for them to learn.

6.1 Questionaire

Questionaire

Criteria	Sub-Criteria	Strongly Agree	Agree	Neutal	Disagree	Strongly Disagree
User Friendliness	EduRobot is easy to use and user-friendly	17	4			
	My interaction with the EDURobot tool is clear and understandable.	10	11			
	EDURobot is not difficult to use	12	9			
	The tool has visual capability to facilitate the understanding of the various path planning algorithms	8	13			

	Changing of parameters is easy and has significant impact	15	6		
Application-specific self-efficiency	I have the ability to set the start and and goal positions of the path plan	14	6	1	
	I have the ability to place obstacles anywhere in the grid	7	14		
	I have the ability to run any of the three different algorithms for any configuration	13	5	2	1
	I can observe the effects of changes in the parameters easily	12	8	1	
	I can thoroughly understand the differences in functioning between the three different algorithms	10	8	2	1
The features and functionality of the software	I can observe the numerical results easily	12	9		
	I can observe the graphical results of EDURobot easily	11	10		
	I can observe the location, sizes and shapes of the obstacles easily	13	8		
	I can easily record the planning algorithm parameters	10	8	3	
The suitability of the tool for educational requirements	I liked working with EDURobot in teams	18	3		
	Working in groups with EDURobot developed my interpersonal skills	9	8	4	
	Working with EDURobot enabled me to develop a better understanding of path planning algorithms	14	7		
	It was beneficial to turn my theoretical experience to practice	9	10	2	
	The simulator can help me to improve the quality quality of my vocational education	10	8	3	
Learning and cognitive aspect	Working with the tool promoted my existing knowledge	7	10	4	
	The simulation helped me to develop the ability to think deeply	11	8	1	1
	It helped me to evaluate my learning	14	8		
	Rather than using the tool, I would prefer to read text and journals	9	7	5	
	Tools are an effective way for me to learn	13	8		
Functionality of PFM	Varying number of iterations significantly changes simulation time	6	10	5	
	Varying number of iterations causes change in path	6	15		
	Decreasing paramater gridSize makes the path approach the obstacles	13	6	2	
	Varying gridSize causes change in path	10	8	3	
Functionality of Local Navigation	Increasing parameter dmax takes the generated path away from the obstacles	6	14	2	

	Dropping parameter alpha below 10^{o} distorts the path	5	11	2	3	
	Increasing linelength obove 1.0 distorts the path	7	9	2	3	
	Increasing alpha may sometimes be necessary to correct the path	7	13	1		
Functionality of of VFH+	Increasing w_s causes a dramatic change in path	5	10	1	3	2
	Increasing minDistance takes the path away from the obstacles (true up to 10 units)	7	12	1	1	
	Changing parameter b has no effect on the final path	5	10	2	2	2
	Increasing alpha distorts the path					
	Changing linelength has no effect on the path	8	11	2		
Cumulative	I was able to see different simulation results with different set of parameters	9	8	2	2	
	I was able to thoroughly understand PFM, Local navigation and VFH+ algorithms	10	10	1		
	PFM gives the smoothest path	5	15	2	2	
	Local navigation is the fastest					
	Local navigation and VFH+ sometimes give the same result	6	12	3		
	I recommend EduRobot to others who want to learn path planning algorithms	13	6	2		

6. CONCLUSION

This thesis is devoted to one of the most challenging robot navigation problems, namely the navigation of mobile robots in uncertain environments that are densely cluttered with obstacles. In order to find a solution to this problem, the following have been done by the author in the thesis.

A literature search has been carried out to find out what has been done in this field previously. It is found that the theory of obstacle avoidance is the most important problem in mobile robot navigation, especially when the robot is surrounded with a large number of obstacles. The literature search has already shown that more used algorithms of robot navigation in the presence of densely populated obstacles are the potential field method, vector field histogram method, and local adaptive navigation scheme. A study of these methods showed that they all suffer from the same problem that the theory it too complex and require powerful processing capabilities. In addition the theory is hard to implement on an actual mobile robot as a result of the need for fast floting-point operations.

The author has proposed the use of fuzzy control based novel algorithm for the navigation of robots in areas with densely populated obstacles. Full theory of the proposed algorithm is given in detail in the Thesis. The control rules of navigation, inference engine operation of fuzzy navigation are described. The new knowledge base is designed for fuzzy navigation system. In addition, the author has developed a Visual C++ (2008) program based simulator that can be used to simulate all of the commonly used robot navigation algorithms. The program is user friendly and is based on graphical application. Users can place obstacles anywhere on the screen. The simulator program is then started where the robot movements can be observed as the robot moves from the starting point to its final goal point. The developed simulation program supports the potential field method, vector field histogram method, local adaptive navigation scheme, and the fuzzy control algorithm. The structure and control algorithms of mobile robots using above methods are presented. The simulation program shows the path of the robot as it moves to its goal point by avoiding the obstacles on its way. All four algorithms can be implemented at the same time so that the user can compare the advantages and disadvantages of the various algorithms. Users can also change parameters

of the algorithms so that the behaviour of each algorithm can be observed easily with the new parameters.

The comparative simulation results have shown that the fuzzy navigation based algorithm has superior performance compared to the other three algorithms. With the fuzzy algorithm the robot reaches its final goal point with the least number of steps and in the shortest possible time. In addition, the fuzzy navigation algorithm theory is simpler and is easier to implement in real-time.

REFERENCES

1. Siegwart Roland. Introduction to autonomous mobile robots. MIT press. 2004.
2. Nourbakhsh, Roland Siegwart and Illah R. *Introduction to Autonomous Mobile Robots.* p.l. : The MIT Press Cambridge, Massachusetts.
3. Choset, Howie.(2005) Principles of Robot Motion: Theory, Algorithms, and Implementation. p.l. : The MIT Press .
4. M. Yousef Ibrahim, Allwyn Fernmdes.(2004,December 8-10) *Study on Mobile Robot Navigation Techniques* IEEE International Conference on Industrial Technology. Vol. 1, p.230 – 236.
5. Juang, Yen-Sheng Chen and Jih-Gau.(2009, August 18-21) *Intelligent Obstacle Avoidance Control Strategy for Wheeled Mobile Robot.* Japan. International Joint Conference. p. 1-6
6. Johann Borenstein and Koren Yoram.(1991, June) *The Vector Field Histogram-Fast Obstacle Avoidance for Mobile Robots.* IEEE Transactions on Robotics and Automation, Vol. 7, No. 3 p. 278 - 288.
7. Iwan Ulrich and Johann Borenstein.(1998, May)*VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots.* Leuven, Belgium : IEEE International Conference on Robotics & Automation. p. 1573-1577.
8. Xiaoyu Yang, Mehrdad Moallem,Rajni V.(2005, December) Patel. *A Layered Goal-Oriented Fuzzy Motion Planning Strategy for Mobile Robot Navigation* IEEE Cybernetics Vol. 35, No. 6, p. 1214.
9. Y. Koren , J . Borenstein.(1991) *Potential field methods and their inherent limitations for mobile robot navigation.* In Proceedings of the IEEE International Conference on Robotics and Automation.
10. K.S . Al-Sultan and M. D. S. Aliyu.(1996) *A new potential field-based algorithm for path planning.* Journal of Intelligent and Robotics Systems.
11. Dieter Fox, Sebastian Thrun, Wolfram Burgard.(1997, March) *The Dynamic Window Approach to Collision Avoidance.* IEEE Magazine on Robotics &Automation, Vol. 4, No. 1, p. 23.
12. M. Yousef Ibrahim.(2002 28 June - 2 July) *Mobile robot navigation in a cluttered environment using free space attraction" agoraphilic" algorithm.* 9th International Computers and Industrial Engineering Conference on, Vol. 1, p. 377 – 382.
13. K. Ehjimura.(1991) *Motion Planning in Dynamic Environments*. Springer-Verlag,

14. T. Aoki, T. Suzuki, and S. Okuma.(1995) *Acquisition of optirnal action selection in autonomous mobile robot using learning automats (experimental evaluation).* IEEE Conference on Fuzzy Logic and Neural Networks/Evolutzonarg Computation.
15. Stentz, A., Hebert, M. (1995) *A Complete Navigation System for Goal Acquisition in Unknown Environments. Intelligent Robots and Systems*. Proceedings. IEEE/RSJ International Conference on Human Robot Interaction and Cooperative Robots, vol.1 p.425 – 432.
16. Matt Zucker, James Kuffner, Michael Branicky.(2007, April 10-14) *Multipartite RRTs for Rapid Replanning in Dynamic Environments.* IEEE, International Conference on Robotics and Automation. p. 1603.
17. T. Tsubouchi, A. Hirose, and S. Arimoto. A navigation scheme with learning for a mobile robot among multiple moving objects. In Proceedings of IROS'93 Intelligent Robots for Flexibility, 1993.
18. Andrew Howard. Probabilistic Navigation: Copying with uncertainity in Robot Navigation Tasks. PhD thesis.The University of Melbroune, 1999.
19. Anthony R. Cassandra, Leslie Pack Kaelbling, and James A. Kurien (1996). *Acting under uncertainty: discrete bayesian models for mobile-robot navigation.*: IEEE International Conference on Intelligent Robots and Systems.
20. Julio K. Rosenhlatt (1997, January). DAMN: *A Distributed Architecture for Mobile Navigation.* PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh.
21. Andrew Howard (1999). Probabilistic Navigation: *Coping with uncertainty in Robot Navigation Tasks*. PhD thesis, The University of Melbourne.
22. R. C. Arkin (1987). *Motor schema based navigation for a mobile robot: an approach to programming by behavior.* In Proceedings of the 1987 IEEE International Conference on Robotics and Automation.
23. K. H. Low, W. K. Leow, and M. H. Ang (2003). *Enhancing the reactive capabilities of integrated planning and control with cooperative extended Kohcnen maps.* IEEE International Conference on Robotics and Automation.
24. Jing Xiao, Zbigniew Michalewicz, Lixin Zhang, Krzysztof Trojanowski (1997). *Adaptive evolutionary planner/navigator for mobile robots.* IEEE Transactions on Evolutionary Computation.

25. Koran Yorem, Borenstein Johann**,** Potential Field Methods and Their Inherent limitations for Mobile Robot Navigation, Proceedings of the 1991 IEEE International Conference on Robotics and Automation Sacramento. California - April 1991.

26. Borenstein, Iwan Ulrich and Johann (2000, April). *VFH*: Local Obstacle Avoidance with Look-Ahead Verification.* San Francisco, CA : IEEE International Conference on Robotics and Automation Vol.3, p. 2505 - 2511.

27. Atsushi Fujimori, Peter N. Nikiforuk, and Madan M. Gupta. Adaptive Navigation of Mobile Robots with Obstacle Avoidance. IEEE Transactions on Robotics and Automation, Vol 13. No 4, August 1997

28. GUO, Enxiu SHI and Junjie. Jinan (2007, August 18 - 21) *Study of the New Method for Improving Artifical Potential Field in Mobile Robot Obstacle Avoidance.* China : IEEE,. International Automation and Logistics Conference on. p. 282.

29. Warren, Charles W.(1990, May 13-18) *Multiple Robot Path Coordination Using Artificial Potential Fields.* IEEE International Robotics and Automation Conference. p. 500 – 505.

30. Marta C. Mora and Josep Tornero, Prahlad Vadakkepat, Kay Chen Tan and Wang Ming-Liang. *Evolutionary Artificial Potential Fields and Their Application in Real Time Robot Path.* p. 256

31. Cao Qixin, Huang Yanwen, Zhou Jingliang (2006,October 9 - 15). *An Evolutionary Artificial Potential Field Algorithm for Dynamic Path Planning of Mobile Robot.* Beijing, China. IEEE International Conference on Intelligent Robots and Systems. p. 3331.

32. Elon Rimon, Daniel E. Kodischek. Exact Robot Navigation using Artificial Potential Functions. IEEE Transactions on Robotics and Automation, Vol.8, No.5 October 1992

33. Brooks, Rodney A (1986, March). *A Robust Layered Control System For A Mobile Robot.* IEEE Journal Of Robotics and Automation, Vol.2 , No.1, p. 14

34. M. Yousef Ibrahim, L. McFetridge (8-12 July 2001). *The Agoraphilic Algorithm:A New Optimistic Approach for Mobile Robot Navigation.* IEEE International Conference on Advanced Intelligent Mechatronics Proceedings. p. 1334.

35. István Engedy, Gábor Horváth (2009, August 26–28). *Artificial Neural Network based Mobile Robot Navigation.* Budapest, Hungary : IEEE 6th International Symposium on Intelligent Signal Processing. p. 241-246

36. Theodore W. Manikas, Kaveh Ashenayi, and Roger L. Wainwright,Genetic Algorithms for Autonomous Robot Navigation. IEEE Instrumentation and Measurement Magazine, December 2007. pg 26-31

37. Barbara Siemtiawskoska. Cellular Neural Networks for Mobile Robot Navigation. CNNA 1994, Third IEEE International Workshop on Cellular Neural Networks and their Applications, Rome, Italy, December 18-21, 1994

38. Panagiotis G. Zavlangas, Spyros G. Tzafestas, Motion Control for Mobile Robot Obstacle Avoidance and Navigation: A Fuzzy Logic-Based Approach, Systems Analysis Modelling Simulation, Vol.43, No.12, December 2003, pp. 1625-1637

39. Vamsi Mohan Peri, Dan Simon (2005). *Fuzzy Logic Control for an Autonomus Robot.* Annual Meeting of the North American Fuzzy Information Processing Society. p. 338-342.

40**.** Xiaoyu Yang, Student Member, IEEE, Mehrdad Moallem, Member, IEEE, and Rajni V. Patel, Fellow, IEEE. A Layered Goal-Oriented Fuzzy Motion Planning Strategy For Mobile Robot Navigation, IEEE Transactions on Systems, Man. and Cybernetics-Part B: Cybernetics, Vol.35, NO.6, December 2005.pg. 1214-1224

41. Hongche Guo, Cheng Cao, Junyou Yang and Qiuhao Zhang (2009). *Research on Obstacle-avoidance Control Algorithm of Lower Limbs Rehabilitation Robot Based on Fuzzy Control.* 6th International Conference on Fuzzy Systems and Knowledge Discovery. p. 151.

42. Shirinzadeh', S.Parasuraman' V.Ganapathy' and Bijan (2003, Nov 2-6). *Fuzzy Decision Mechanism Combined with Neuro-Fuzzy Controller for Behavior Based Robot Navigation.*. The 29th Annual Conference of the IEEE.

43. Byung Cheol Min, Moon-Su Lee, Donghan Kim. Fuzzy Logic Path Planner and Motion Controller by Evolutionary Programming for Mobile Robots. International Journal of Fuzzy Systems, Vol. 11, Issue 3, September 2009. pg154-163.

44. Xiaohong Cong, Hui Ning, and Zhibin Miao. A Fuzzy Logical Application in a Robot Self Navigation, Proceedings of the 2007 Second IEEE Conference on Industrial Electronics and Applications, pg. 2905-2907

45. S.Parasuraman, V.Ganapathy, and Bijan Shirinzadeh, Mobile Robot Navigation using Alpha-Level Fuzzy Logic System: Experimental Investigations. 2008 IEEE International Conference on Systems, Man and Cybernetics (SMC 2008) pg. 1878-1884

46. Cang Ye, Nelson H. C. Yung, Danwei Wang, A Fuzzy Controller With Supervised Learning Assisted Reinforcement Learning Algorithm for Obstacle Avoidance, IEEE Transactions on Systems, Man and Cybernetics- Part B: Cybernetics, Vol. 33, No.1, February 2003. pg. 17-27.

47. Boubertakh, H. , Tadjine M. , Glorennec, P. A new mobile robot navigation method using fuzzy logic and a modified Q-learning algorithm, Journal of Intelligent and Fuzzy Systems , Vol.21 Issue 1/2, 2010. pg.113-119.

48. Dongqing Shi, Damion Dunlap, Emmanuel G. Collins, Jr., A Comparison Betweeen a Fuzzy Behavioural Algorithm, and a Vector Polar Histogtram Algorithm for Mobile Robot Navigation, Proceedings of the 2007 IEEE International Symposium on Compuıtational Intelligence in Robotics and Automation, Jacksonville, FL, USA, June 20 23, 2007. pg. 260-264

49. R.Abiyev, B.Erin, and D.İbrahim. Navigation of mobile robots in the presence of obstacles. Advances in Engineering Software Journal, Vol.41, Issues 10-11, 2010.pp.1179-1186

50. R.Abiyev, D.Ibrahim, and B.Erin. Fuzzy Obstacle Avoidance and Navigation of Mobile Robot. 6th International Architecture and Engineering Symposium.Electrical and Electronics Systems and Computer Symposium. Creating the Future", 2010, pg62-66

51. Zadeh L.A.(1975). The concept of linguistic variable and its application to approximate reasoning. Information Sciences, v.8.

52. Wei Li, Xun Feng (2006, 8-11 October). *Behaviour Fusion For Robot Navigation In Uncertain Environments using Fuzzy Logic.* IEEE International Conference on Systems, Man and Cybernetics Vol.6 p. 4910 – 4915

53. A. Fob and P.Trahanias.(2003) *Predictive control of robot velocity to avoid obstacles in dynamic environments.* IEEE/RS.J International Conference on Intelligent Robots and Systems (IROS).

54. H. Hagras, "A Hierarchical Type-2 Fuzzy Logic Control Architecture For Autonomous Mobile Robots," *IEEE Trans. on Fuzzy Syst*., 12,(4), pp. 524–539, 2004.

55. J. M. Mendel, Uncertain Rule-Based Fuzzy Logic System: Introduction and New Directions, Prentice Hall, Upper Saddle River, NJ, 2001.

56. N.N.Karnik and J.M.Mendel, Liang Q, "Type-2 fuzzy logic systems", IEEE Trans. Fuzzy Syst., 7, (1999), pp.643–658.

57. H. Wu and J. Mendel, "Uncertainty bounds and their use in the design of interval type-2 fuzzy logic systems," *IEEE Trans. on Fuzzy Systems*, vol.10, pp. 622-639, October 2002.

58. Q. Liang And J.M. Mendel. 'Interval Type-2 Logic Systems: Theory And Design', *IEEE Trans. Fuzzy Syst*., vol. 8, pp. 535–550, 2000

59. J. M. Mendel and R.I .John and F. Liu, "Interval Type-2 Fuzzy Logic Systems Made Simple", *IEEE Trans. Fuzzy Syst*., vol. 14, N.6, pp.808-821, 2006.

60. J.M.Mendel, and Robert I.Bob, "Type-2 fuzzy sets made simple", IEEE Trans. Fuzzy Syst., vol 10, (2002),pp.117-127.

58. R. Abiyev, D. Ibrahim, and B. Erin. EDURobot: An Educational Computer Simulation Programme for Navigation of Mobile Robots in the Presence of Obstacles. International Journal of Engineering Education. Vol.26. No.1, pp.18-29, 2010.

59. R. Abiyev, B. Erin, and D. Ibrahim. Teaching robot navigation in the presence of obstacles using a computer simulation program. Procedia – Social and Behavioral Sciences. Elsevier, Vol. 2, No. 2, pages 565-571, 2010.

APPENDIX

Appendix: Knowledge Base using Left, Centre and Right distances

If $D_l(k)$ is "Small" and $D_c(k)$ is "Small" and $D_r(k)$ is "Small" **then** $\theta(k)$= *Turn_Right by* $\theta_{rs}(k)$

If $D_l(k)$ is "Small" and $D_c(k)$ is "Small" and $D_r(k)$ is "Medium" **then** $\theta(k)$= *Turn_Left by* $\theta_{ls}(k)$

If $D_l(k)$ is "Small" and $D_c(k)$ is "Small" and $D_r(k)$ is "Large" **then** $\theta(k)$= *Turn_Left by* $\theta_{ls}(k)$

If $D_l(k)$ is "Small" and $D_c(k)$ is "Medium" and $D_r(k)$ is "Small" **then** $\theta(k)$= *Turn_Right by* $\theta_{rs}(k)$

If $D_l(k)$ is "Small" and $D_c(k)$ is "Medium" and $D_r(k)$ is "Medium" **then** $\theta(k)$= *Turn_Left by* $\theta_{ls}(k)$

If $D_l(k)$ is "Small" and $D_c(k)$ is "Medium" and $D_r(k)$ is "Large" **then** $\theta(k)$= *Turn_Left by* $\theta_{ls}(k)$

If $D_l(k)$ is "Small" and $D_c(k)$ is "Large" and $D_r(k)$ is "Small" **then** $\theta(k)$= *Turn_Right by* $\theta_{rs}(k)$

If $D_l(k)$ is "Small" and $D_c(k)$ is "Large" and $D_r(k)$ is "Medium" **then** $\theta(k)$= *Turn_Left by* $\theta_{ls}(k)$

If $D_l(k)$ is "Small" and $D_c(k)$ is "Large" and $D_r(k)$ is "Large" **then** $\theta(k)$= *Turn_Left by* $\theta_{ls}(k)$

If $D_l(k)$ is "Medium" and $D_c(k)$ is "Small" and $D_r(k)$ is "Small" **then** $\theta(k)$= *Turn_Right by* $\theta_{rs}(k)$

If $D_l(k)$ is "Medium" and $D_c(k)$ is "Small" and $D_r(k)$ is "Medium" **then** $\theta(k)$= *Turn_Right by* $\theta_{rs}(k)$

If $D_l(k)$ is "Medium" and $D_c(k)$ is "Small" and $D_r(k)$ is "Large" **then** $\theta(k)$= *Turn_Left by* $\theta_{ls}(k)$

If $D_l(k)$ is "Medium" and $D_c(k)$ is "Medium" and $D_r(k)$ is "Small" **then** $\theta(k)$= *Turn_Right by* $\theta_{rs}(k)$

If $D_l(k)$ is "Medium" and $D_c(k)$ is "Medium" and $D_r(k)$ is "Medium" **then** $\theta(k)$= *Turn_Left by* $\theta_{ls}(k)$

If $D_l(k)$ is "Medium" and $D_c(k)$ is "Medium" and $D_r(k)$ is "Large" **then** $\theta(k)$= *Turn_Left by* $\theta_{ls}(k)$

If $D_l(k)$ is "Medium" and $D_c(k)$ is "Large" and $D_r(k)$ is "Small" **then** $\theta(k)$= *Turn_Right by* $\theta_{rs}(k)$

If $D_l(k)$ is "Medium" and $D_c(k)$ is "Large" and $D_r(k)$ is "Medium" **then** $\theta(k)$= *Turn_Left by* $\theta_{ls}(k)$

If $D_l(k)$ is "Medium" and $D_c(k)$ is "Large" and $D_r(k)$ is "Large" **then** $\theta(k)$= *Turn_Left by* $\theta_{ls}(k)$

If $D_l(k)$ is "Large" and $D_c(k)$ is "Large" and $D_r(k)$ is "Small" **then** $\theta(k)$= *Turn_Right by* $\theta_{rs}(k)$

If $D_l(k)$ is "Large" and $D_c(k)$ is "Large" and $D_r(k)$ is "Medium" **then** $\theta(k)$= *Turn_Right by* $\theta_{rs}(k)$

If $D_l(k)$ is "Large" and $D_c(k)$ is "Large" and $D_r(k)$ is "Large" **then** $\theta(k)$= *Turn_Right by* $\theta_{rs}(k)$

Appendix: Excerpts from code

1. Potential Field Method

```
namespace mobileRobot
{
7.      public class pField2D
8.      {
9.          public double robx, roby, top, bottom, left, right, alfa;
10.             protected int numberOfIterations;
11.             protected double[,] gridValue;
12.             protected bool[,] savePoint;
13.             protected int numberOfGridLinesX, numberOfGridLinesY;
14.             public float gridSize;
15.             public int goalX, goalY;
16.             public ArrayList arPathPoints = new ArrayList();
17.             public int noofpathpoints;
18.             public double fieldvalue= -Math.Pow(2,124);
19.             public void calculateField(int numberOfGridLinesX, int
    numberOfGridLinesY, double robx, double roby, int goalX, int goalY,
    float gridSize, int numberOfIterations, ArrayList arShape)
20.             {
21.                 this.numberOfGridLinesX = numberOfGridLinesX;
22.                 this.numberOfGridLinesY = numberOfGridLinesY;
23.                 this.gridSize = gridSize;
24.                 this.goalX = Convert.ToInt32(goalX/gridSize);
25.                 this.goalY = Convert.ToInt32(goalY/gridSize);
26.                 this.robx = robx;
27.                 this.roby = roby;
28.                 this.numberOfIterations = numberOfIterations;
29.                 DateTime t1 = DateTime.Now;
30.                 gridValue = new double[numberOfGridLinesX+10,
    numberOfGridLinesY+10];
31.                 savePoint = new bool[numberOfGridLinesX+10,
    numberOfGridLinesY+10];
32.                 for (int i = 0; i <= numberOfGridLinesX; i++){
33.                     for (int j = 0; j <= numberOfGridLinesY; j++){
34.                         gridValue[i, j] = 0;
35.                         savePoint[i, j] = true;
36.                     }
37.                 }
38.                 for (int i = 1; i < numberOfGridLinesX; i++)
39.                     for (int j = 1; j < numberOfGridLinesY; j++)
40.                         savePoint[i, j] = false;
41.                 for (int k = 0; k < arShape.Count; k++)
42.                 { shape s = (shape)arShape[k];
43.                     int initialY=s.y/Convert.ToInt32(gridSize);
44.                     int initialX=s.x/Convert.ToInt32(gridSize);
45.                     if (s.itsShape == "Rectangle")
46.                     {
47.                         for (int j = initialY; j <= initialY +
    s.height / gridSize; j++)
48.                         {
49.                             for (int i = initialX; i <= initialX +
    s.width / gridSize; i++)
50.                             {
51.                                 savePoint[i, j] = true;
52.                             }
53.                         }
54.                     }
55.                     if (s.itsShape == "Ellipse")
56.                     {
```

```
57.                         for (int i = initialX; i <= initialX +
   s.width / gridSize; i++)
58.                         {
59.                             for (int j = initialY; j <= initialY +
   s.height / gridSize; j++)
60.                             {
61.                                 if ((Math.Pow((i * gridSize - (s.x
   + s.width / 2)), 2) / Math.Pow(s.width / 2, 2)) + (Math.Pow((j *
   gridSize - (s.y + s.height / 2)), 2) / Math.Pow(s.height / 2, 2)) <=
   1)
62.                                 {
63.                                     savePoint[i, j] = true;
64.                                 }
65.                             }
66.                         }
67.                     }
68.                 }
69.
70.                 gridValue[Convert.ToInt32(this.goalX),
   Convert.ToInt32(this.goalY)] = fieldvalue;
71.                 savePoint[Convert.ToInt32(this.goalX),
   Convert.ToInt32(this.goalY)] = true;
72.                 for (int iter = 0; iter < numberOfIterations;
   iter++){
73.                     for (int j = 1; j <= numberOfGridLinesY; j++){
74.                         for (int i = 1;i <= numberOfGridLinesX;
   i++){
75.                             if (!savePoint[i, j])gridValue[i, j] =
   (gridValue[i, j - 1] + gridValue[i, j + 1] + gridValue[i - 1, j] +
   gridValue[i + 1, j]) / 4.0;
76.                         }
77.                     }
78.                 }
79.             long delta = (long)((TimeSpan)(DateTime.Now -
   t1)).TotalMilliseconds;
80.             }
81.             public double field(double a, double b)
82.             {
83.                 double dx, dy, bot, bot1, bot2; Int32 sx, sy;
84.                 double x = Convert.ToDouble(a / gridSize);
85.                 double y = Convert.ToDouble(b / gridSize);
86.                 int gridx = numberOfGridLinesX + 1;
87.                 int gridy = numberOfGridLinesY + 1;
88.                 if ((x == gridx) || (y == gridy)) return 0;
89.                 dx = x - Math.Floor(x);
90.                 dy = y - Math.Floor(y);
91.                 sx = Convert.ToInt32(Math.Floor(x));
92.                 sy = Convert.ToInt32(Math.Floor(y));
93.                 bot1 = (1.0 - dy) * gridValue[sx, sy] + dy *
   gridValue[sx, sy + 1];
94.                 bot2 = (1.0 - dy) * gridValue[sx + 1, sy] + dy *
   gridValue[sx + 1, sy + 1];
95.                 bot = dx * bot2 + (1.0 - dx) * bot1;
96.                 return (bot);
97.             }
98.
99.
100.            public void getArPathPointsArray(double lineLength){
101.                double xmin, ymin;
102.                arPathPoints.Clear();
103.                while (true){
```

```
104.                     top = field(robx, roby - 1);
105.                     bottom = field(robx, roby + 1);
106.                     left = field(robx - 1, roby);
107.                     right = field(robx + 1, roby);
108.                     alfa = Math.Atan2(top - bottom, left - right);
109.                     xmin = lineLength * Math.Cos(alfa); ymin =
    lineLength*Math.Sin(alfa);
110.                     robx = robx + xmin; roby = roby + ymin;
111.                     pathline p=new pathline();
112.                     p.x = robx; p.y = roby;
113.                     arPathPoints.Add(p);
114.                     noofpathpoints = noofpathpoints + 1;
115.                    if ((robx > (goalX*gridSize - 2) && robx <
    (goalX*gridSize + 2)) && (roby > (goalY*gridSize - 2) && roby <
    (goalY*gridSize + 2)))
116.                     {
117.                         break;
118.                     }
119.                 }
120.             }
121.         }
122.     }
```

2. Vector Field Histogram Plus

```
private void form_grid()
    {
        length = 0;
        cell = new PointD[ws * ws];
        for (int i = Convert.ToInt32((robx - (ws / 2))); i <
Convert.ToInt32((robx + (ws / 2))); i++)
        {
            for (int j = Convert.ToInt32(roby); j < Convert.ToInt32(roby +
ws); j++)
            {
                if ((i < numberOfGridLinesX) && (j < numberOfGridLinesY) &&
(i>0) && (j>0))
                {


                    cell[length] = new PointD();
                    cell[length].x = i;
                    cell[length].y = j;
                    if (cv[i,j] == 1)
                     cell[length].certainity = 1;
                    else
                     cell[length].certainity = 0;
                    length++;
                }
            }
        }

       }

private void buildPrimaryPolarHistogram()
    {
        double curSectorAngle, sum, rightLimit, leftLimit;
        int enlargedRadius;


        distance = new double[length];
        direction = new double[length];
        enlargementAngle = new double[length];
```

```
        magnitude = new double[length];

        for (int i = 0; i < length; i++)
        {

            direction[i] = Math.Atan2(cell[i].y - roby, cell[i].x - robx);
            distance[i] = Math.Sqrt(Math.Pow(cell[i].y - roby, 2) +
Math.Pow(cell[i].x - robx, 2));
            enlargedRadius = minDistance;
            direction[i] = resizeAngle(direction[i]);
            double theSineOfEnlargementAngle = (enlargedRadius /
distance[i]);
            if (theSineOfEnlargementAngle > 1.0)
                enlargementAngle[i] = angleDifference(direction[i], curDir)
+ ((alpha * 3.14) / 180);
            else enlargementAngle[i] =
Math.Asin(theSineOfEnlargementAngle);
            enlargementAngle[i] = resizeAngle(enlargementAngle[i]);
            magnitude[i] = Math.Pow(cell[i].certainity, 2)*(a - b *
Math.Pow(distance[i], 2));
        }


        for (int k = 0; k < numSectors; k++)
        {
            curSectorAngle = ((k * alpha) * 3.14 )/ 180;

            sum = 0;


            for (int i = 0; i < length; i++)
            {

                rightLimit = direction[i] - enlargementAngle[i];
                leftLimit = direction[i] + enlargementAngle[i];

                if (magnitude[i] > 0 && (isAngleBetween(curSectorAngle,
rightLimit, leftLimit)))
                    sum += magnitude[i];
            }


            primaryPolarHistogram[k] = sum;

        }
    }



    private void buildBinaryPolarHistogram(double thresholdLow, double
thresholdHigh)
    {
        int p;

        for (int i = 0; i < numSectors; i++)
            if (primaryPolarHistogram[i] > thresholdHigh)
                binaryPolarHistogram[i] = BLOCKED;
            else if (primaryPolarHistogram[i] < thresholdLow)
                binaryPolarHistogram[i] = FREE;
            else
            {
```

```
                p = ((i - 1) >= 0) ? (i - 1) : (numSectors - 1);
                binaryPolarHistogram[i] = binaryPolarHistogram[p];
            }
    }

    private void buildMaskedPolarHistogram()
    {
        PointD trajCenterRight, trajCenterLeft;
        distancetoTrajCenterRight = new double[length];
        distancetoTrajCenterLeft  = new double[length];
        double trajangle;
        trajCenterRight = new PointD();
        trajCenterLeft = new PointD();
    //    minSteeringRadius = enlargedRadius;

        trajangle = curDir - 1.57;

        trajCenterRight.x = robx + minSteeringRadius * Math.Cos(trajangle);
        trajCenterRight.y = roby + minSteeringRadius * Math.Sin(trajangle);
        trajCenterLeft.x = robx - minSteeringRadius * Math.Cos(trajangle);
        trajCenterLeft.y = roby - minSteeringRadius * Math.Sin(trajangle);

        double reverseDir = curDir + 1.57;
        double rightLimitAngle = reverseDir, leftLimitAngle = reverseDir;


        for (int i = 0; i < length; i++)
        {
            distancetoTrajCenterRight[i] = Math.Sqrt(Math.Pow(cell[i].x -
trajCenterRight.x, 2) + Math.Pow(cell[i].y - trajCenterRight.y, 2));
            distancetoTrajCenterLeft[i] = Math.Sqrt(Math.Pow(cell[i].x -
trajCenterLeft.x, 2) + Math.Pow(cell[i].y - trajCenterLeft.y, 2));



                if (isAngleBetween(direction[i], rightLimitAngle, curDir)
&& (distancetoTrajCenterRight[i] < (minSteeringRadius + enlargedRadius)))

                    rightLimitAngle = direction[i];

                if (isAngleBetween(direction[i], curDir, leftLimitAngle) &&
(distancetoTrajCenterLeft[i] < (minSteeringRadius + enlargedRadius)))


                    leftLimitAngle = direction[i];


        }

        for (int k = 0; k < numSectors; k++)
        {

            double curSectorAngle = ((k*alpha*3.14)/180);

            if ((binaryPolarHistogram[k] == 0) &&
                isAngleBetween(curSectorAngle, rightLimitAngle,
leftLimitAngle))
            {


                maskedPolarHistogram[k] = FREE;
```

```
            }
            else
            {
                maskedPolarHistogram[k] = BLOCKED;

            }  // end if

        }  // end for
    }



    public void getArPathPointsArray(double lineLength)
    {
        double xmin, ymin;
        double targetDir, prevDir, curDirnew;
        int targetK;
        targetDir = Math.Atan2(goalY - roby, goalX - robx);
        targetDir = resizeAngle(targetDir);
        targetK = (int)(((targetDir * 180) / 3.14) / alpha);
        curDir = targetDir;
        prevDir = targetDir;
        arPathPoints.Clear();

        while (true)
   {
       //  robx < goalX) && (roby < goalY))

           form_grid();
           buildPrimaryPolarHistogram();
           buildBinaryPolarHistogram(0.001, 1);
           buildMaskedPolarHistogram();

           curDirnew = chooseSteeringDirection(10, targetK, curDir,
prevDir, targetDir);
           curDirnew = resizeAngle(curDirnew);
           //determine target sector(i.e. the sector the goal point is in)
and call it target
           //if (!(curDir == -1))
           //else { xmin = 0; ymin = 0; }
      //     while (noofcycles < 1)
        //   {
               agentPos = new PointD();
               agentPos.x = robx;
               agentPos.y = roby;
               noofpathpoints = noofpathpoints + 1;
               arPathPoints.Add(agentPos);
               xmin = lineLength * Math.Cos(curDirnew); ymin = lineLength *
Math.Sin(curDirnew);
               robx = robx + xmin; roby = roby + ymin;

           targetDir = Math.Atan2(goalY - roby, goalX - robx);
           targetDir = resizeAngle(targetDir);
           targetK = (int)(((targetDir * 180) / 3.14) / alpha);
           prevDir = curDir;
           curDir = curDirnew;


           if ((robx > (goalX - 2)) && (robx < (goalX + 2)) && (roby >
(goalY - 2) && roby < (goalY + 2)))
           {
```

```
                break;
            }


        }
    }
  } private void buildPrimaryPolarHistogram()
    {
        double curSectorAngle, sum, rightLimit, leftLimit;
        int enlargedRadius;


        distance = new double[length];
        direction = new double[length];
        enlargementAngle = new double[length];
        magnitude = new double[length];

        for (int i = 0; i < length; i++)
        {

            direction[i] = Math.Atan2(cell[i].y - roby, cell[i].x - robx);
            distance[i] = Math.Sqrt(Math.Pow(cell[i].y - roby, 2) +
Math.Pow(cell[i].x - robx, 2));
            enlargedRadius = minDistance;
            direction[i] = resizeAngle(direction[i]);
            double theSineOfEnlargementAngle = (enlargedRadius /
distance[i]);
            if (theSineOfEnlargementAngle > 1.0)
                enlargementAngle[i] = angleDifference(direction[i], curDir)
+ ((alpha * 3.14) / 180);
            else enlargementAngle[i] =
Math.Asin(theSineOfEnlargementAngle);
            enlargementAngle[i] = resizeAngle(enlargementAngle[i]);
            magnitude[i] = Math.Pow(cell[i].certainity, 2)*(a - b *
Math.Pow(distance[i], 2));
        }


        for (int k = 0; k < numSectors; k++)
        {
            curSectorAngle = ((k * alpha) * 3.14 )/ 180;

            sum = 0;


            for (int i = 0; i < length; i++)
            {

                rightLimit = direction[i] - enlargementAngle[i];
                leftLimit = direction[i] + enlargementAngle[i];

                if (magnitude[i] > 0 && (isAngleBetween(curSectorAngle,
rightLimit, leftLimit)))
                    sum += magnitude[i];
            }


            primaryPolarHistogram[k] = sum;

        }
    }
```

```
    private void buildBinaryPolarHistogram(double thresholdLow, double
thresholdHigh)
    {
        int p;

        for (int i = 0; i < numSectors; i++)
            if (primaryPolarHistogram[i] > thresholdHigh)
                binaryPolarHistogram[i] = BLOCKED;
            else if (primaryPolarHistogram[i] < thresholdLow)
                binaryPolarHistogram[i] = FREE;
            else
            {
                p = ((i - 1) >= 0) ? (i - 1) : (numSectors - 1);
                binaryPolarHistogram[i] = binaryPolarHistogram[p];
            }
    }

    private void buildMaskedPolarHistogram()
    {
        PointD trajCenterRight, trajCenterLeft;
        distancetoTrajCenterRight = new double[length];
        distancetoTrajCenterLeft  = new double[length];
        double trajangle;
        trajCenterRight = new PointD();
        trajCenterLeft = new PointD();
    //    minSteeringRadius = enlargedRadius;

        trajangle = curDir - 1.57;

        trajCenterRight.x = robx + minSteeringRadius * Math.Cos(trajangle);
        trajCenterRight.y = roby + minSteeringRadius * Math.Sin(trajangle);
        trajCenterLeft.x = robx - minSteeringRadius * Math.Cos(trajangle);
        trajCenterLeft.y = roby - minSteeringRadius * Math.Sin(trajangle);

        double reverseDir = curDir + 1.57;
        double rightLimitAngle = reverseDir, leftLimitAngle = reverseDir;


        for (int i = 0; i < length; i++)
        {
            distancetoTrajCenterRight[i] = Math.Sqrt(Math.Pow(cell[i].x -
trajCenterRight.x, 2) + Math.Pow(cell[i].y - trajCenterRight.y, 2));
            distancetoTrajCenterLeft[i] = Math.Sqrt(Math.Pow(cell[i].x -
trajCenterLeft.x, 2) + Math.Pow(cell[i].y - trajCenterLeft.y, 2));



                if (isAngleBetween(direction[i], rightLimitAngle, curDir)
&& (distancetoTrajCenterRight[i] < (minSteeringRadius + enlargedRadius)))

                    rightLimitAngle = direction[i];

                if (isAngleBetween(direction[i], curDir, leftLimitAngle) &&
(distancetoTrajCenterLeft[i] < (minSteeringRadius + enlargedRadius)))


                    leftLimitAngle = direction[i];
```

```
        }

        for (int k = 0; k < numSectors; k++)
        {

            double curSectorAngle = ((k*alpha*3.14)/180);

            if ((binaryPolarHistogram[k] == 0) &&
                isAngleBetween(curSectorAngle, rightLimitAngle,
leftLimitAngle))
            {


                maskedPolarHistogram[k] = FREE;
            }
            else
            {
                maskedPolarHistogram[k] = BLOCKED;

            }  // end if

        }  // end for
    }


    public void getArPathPointsArray(double lineLength)
    {
        double xmin, ymin;
        double targetDir, prevDir, curDirnew;
        int targetK;
        targetDir = Math.Atan2(goalY - roby, goalX - robx);
        targetDir = resizeAngle(targetDir);
        targetK = (int)(((targetDir * 180) / 3.14) / alpha);
        curDir = targetDir;
        prevDir = targetDir;
        arPathPoints.Clear();

        while (true)
   {
       //  robx < goalX) && (roby < goalY))

           form_grid();
           buildPrimaryPolarHistogram();
           buildBinaryPolarHistogram(0.001, 1);
           buildMaskedPolarHistogram();

           curDirnew = chooseSteeringDirection(10, targetK, curDir,
prevDir, targetDir);
           curDirnew = resizeAngle(curDirnew);
           //determine target sector(i.e. the sector the goal point is in)
and call it target
           //if (!(curDir == -1))
           //else { xmin = 0; ymin = 0; }
      //     while (noofcycles < 1)
        //   {
             agentPos = new PointD();
             agentPos.x = robx;
             agentPos.y = roby;
             noofpathpoints = noofpathpoints + 1;
             arPathPoints.Add(agentPos);
```

```
                xmin = lineLength * Math.Cos(curDirnew); ymin = lineLength *
Math.Sin(curDirnew);
                robx = robx + xmin; roby = roby + ymin;

                //          agentPos = new PointD();
                //          agentPos.x = robx;
                //           agentPos.y = roby;
                //       arPathPoints.Add(agentPos);
      //          noofcycles = noofcycles + 1;
                //          if ((robx > goalX) || (roby > goalY))
                //             break;
//           }

            targetDir = Math.Atan2(goalY - roby, goalX - robx);
            targetDir = resizeAngle(targetDir);
            targetK = (int)(((targetDir * 180) / 3.14) / alpha);
            prevDir = curDir;
            curDir = curDirnew;


            if ((robx > (goalX - 2)) && (robx < (goalX + 2)) && (roby >
(goalY - 2) && roby < (goalY + 2)))
            {
                break;
            }


        }
    }
      }


3.  Adaptive(Local) Navigation
4.  public void getArPathPointsArray(double lineLength)
5.          {
6.              double xmin, ymin, xleft, yleft, xcenter, ycenter, xright,
    yright;
7.              int xround, yround, xroundl, yroundl, xcenterr, ycenterr;
8.              double thetat, theta, thetanew, thetaalpha, epsilon, d;
9.
10.                arPathPoints.Clear();
11.                theta = 1.57;
12.                while (true)
13.                {
14.                    distcenter = 0;
15.                    distleft = 0;
16.                    distright = 0;
17.                    pathline p = new pathline();
18.                    p.x = robx; p.y = roby;
19.                    noofpathpoints = noofpathpoints + 1;
20.                    arPathPoints.Add(p);
21.
22.                    thetat = Math.Atan2((roby-goalY),(robx-goalX));
23.                    thetat = resizeAngle(thetat);
24.                    if (thetat <= theta)
25.                      thetat = thetat + 3.14;
26.                    else thetat = thetat - 3.14;
27.                    thetat = resizeAngle(thetat);
28.                    xright = robx + lineLength*Math.Cos(theta - alpha);
29.                    yright = roby + lineLength*Math.Sin(theta - alpha);
30.                    xround = Convert.ToInt32(xright);
```

```
31.                     yround = Convert.ToInt32(yright);
32.                     xleft = robx + lineLength*Math.Cos(theta + alpha);
33.                     yleft = roby + lineLength*Math.Sin(theta + alpha);
34.                     xroundl = Convert.ToInt32(xleft);
35.                     yroundl = Convert.ToInt32(yleft);
36.                     xcenter = robx + lineLength*Math.Cos(theta);
37.                     ycenter = roby + lineLength*Math.Sin(theta);
38.                     xcenterr = Convert.ToInt32(xcenter);
39.                     ycenterr = Convert.ToInt32(ycenter);
40.                     while (!(savePoint[xround+5, yround+5]) && (distright
   < dmax+5))
41.                     {
42.                         xright = xright + lineLength*Math.Cos(theta -
   alpha);
43.                         yright = yright + lineLength*Math.Sin(theta -
   alpha);
44.                         distright = Math.Sqrt(Math.Pow(yright - roby, 2)
   + Math.Pow(xright - robx, 2));
45.                         xround = Convert.ToInt32(xright);
46.                         yround = Convert.ToInt32(yright);
47.                     }
48.
49.                     if (distright >= dmax)
50.                         distright = -1;
51.
52.                     //distleft = Math.Sqrt(Math.Pow(yleft - roby, 2) +
   Math.Pow(xleft - robx, 2));
53.                     while (!(savePoint[xroundl+5, yroundl+5]) &&
   (distleft < dmax+5))
54.                     {
55.                         xleft = xleft + lineLength*Math.Cos(theta +
   alpha);
56.                         yleft = yleft + lineLength*Math.Sin(theta +
   alpha);
57.                         distleft = Math.Sqrt(Math.Pow(yleft - roby, 2) +
   Math.Pow(xleft - robx, 2));
58.                         xroundl = Convert.ToInt32(xleft);
59.                         yroundl = Convert.ToInt32(yleft);
60.                     }
61.
62.                     if (distleft >= dmax)
63.                         distleft = -1;
64.
65.                     //distcenter = Math.Sqrt(Math.Pow(ycenter - roby, 2)
   + Math.Pow(xcenter - robx, 2));
66.                     while (!(savePoint[xcenterr+5, ycenterr+5]) &&
   (distcenter < dmax+5))
67.                     {
68.                         xcenter = xcenter + lineLength*Math.Cos(theta);
69.                         ycenter = ycenter + lineLength*Math.Sin(theta);
70.                         distcenter = Math.Sqrt(Math.Pow(ycenter - roby,
   2) + Math.Pow(xcenter - robx, 2));
71.                         xcenterr = Convert.ToInt32(xcenter);
72.                         ycenterr = Convert.ToInt32(ycenter);
73.                     }
74.
75.                         if (distcenter >= dmax)
76.                             distcenter = -1;
77.
78.
79.                     d = max(distleft, distright);
```

```
80.                   epsilon = Math.Atan((d*Math.Cos(alpha) -
   distcenter)/(d*Math.Sin(alpha)));
81.
82.
83.                   if (distcenter > 0)
84.                           if ((distleft < 0) && (distright < 0))
85.                               thetaalpha = theta + 1.57;
86.                           else if ((distleft < 0) || (distright < 0))
87.                                   thetaalpha = theta +
   Math.Sign(distleft - distright) * (1.57 - epsilon) + Math.Sign(distleft
   - distright) * 3.14;
88.                                    else thetaalpha = theta +
   Math.Sign(distleft - distright) * (1.57 - epsilon);
89.                   else
90.                       thetaalpha = theta;
91.
92.                   if ((distcenter < 0) && ((distleft > 0) && (distright
   > 0)))
93.                       thetanew = thetaalpha;
94.                   else if (distcenter > 0)
95.                           if (((thetat > thetaalpha) && (thetaalpha >
   theta)) || ((thetat <= thetaalpha) && (thetaalpha <= theta)))
96.                               thetanew = thetat;
97.                           else thetanew = thetaalpha;
98.                        else if (((distleft < 0) && (thetat >= theta))
   || ((distright < 0) && (thetat <= theta)))
99.                               thetanew = thetat;
100.                            else thetanew = thetaalpha;
101.
102.                  theta = resizeAngle(thetanew);
103.                  //thetat = Math.Atan((goalY - roby) / (goalX -
   robx));
104.                  //theta = thetanew;
105.                  xmin = lineLength*Math.Cos(theta);
106.                  ymin = lineLength*Math.Sin(theta);
107.                  robx = robx + xmin; roby = roby + ymin;
108.                  if (((robx > goalX-1) && (robx < goalX+1)) && ((roby
   > goalY-1) && (roby < goalY+1)))
109.                  {
110.                      break;
111.                  }
```

4. Fuzzy Algorithm

```
  public void getArPathPointsArray(double lineLength)
        {
            //  double xleft, yleft, xcenter, ycenter, xright, yright;
            //  int xround, yround, xroundl, yroundl, xcenterr, ycenterr;
            char ch1 = 'k';
            char ch2 = 'k';
            int safedobst = 5; int xfix, yfix, safed = 10, llflag;
            int newtheta = 0;
            int distg = 5, disto = 15;
            double al, bl, c, ar = 0, br = 0;
            double xcenterr, ycenterr;
            arPathPoints.Clear();
            lineLength = 1;
            theta = Math.Atan2((goalY - roby), (goalX - robx));

            xcenterr = robx; ycenterr = roby;
            xcenter = Convert.ToInt32(xcenterr); ycenter =
Convert.ToInt32(ycenterr);
```

```
            pathline p0 = new pathline(); p0.x = robx; p0.y = roby;
            noofpathpoints = noofpathpoints + 1; arPathPoints.Add(p0);

            while (Math.Abs(goalX - robx) > distg && Math.Abs(goalY - roby)
> distg) //(true) //
            {
                xcenterr = robx; ycenterr = roby;
                xcenter = Convert.ToInt32(xcenterr); ycenter =
Convert.ToInt32(ycenterr);

                // while(!(savePoint[xcenterr+5, ycenterr+5]) &&
(distcenter < dmax+5))
                while (!(savePoint[xcenter, ycenter]) || (newtheta == 1))
                {
                    xcenterr = xcenterr + lineLength * Math.Cos(theta);
                    ycenterr = ycenterr + lineLength * Math.Sin(theta);
                    distcenter = Math.Sqrt(Math.Pow(ycenterr - roby, 2) +
Math.Pow(xcenterr - robx, 2));
                    xcenter = Convert.ToInt32(xcenterr);
                    ycenter = Convert.ToInt32(ycenterr);
                    robx = xcenter;
                    roby = ycenter;

                    if (ch1 == 'k')
                    {
                        if (!(savePoint[xcenter + disto, ycenter + disto]))
                        {
                            pathline p = new pathline();
                            p.x = robx; p.y = roby;
                            noofpathpoints = noofpathpoints + 1;
arPathPoints.Add(p);
                        }
                    }


                    if (ch1 == 'l')
                    {

                        if (!(savePoint[xcenter + disto, ycenter + disto]))
                        {
                            pathline p = new pathline();
                            p.x = robx; p.y = roby;
                            noofpathpoints = noofpathpoints + 1;
arPathPoints.Add(p);
                        }
if (((xcenter <= xl) && (ycenter >= yr)) || ((robx >= xr) && (roby >= yr)))
                        {
                            theta = Math.Atan2((goalY - roby), (goalX -
robx));
                            newtheta = 0;
                            ch1 = 'k';
                        }

                    }


                    if (ch1 == 'u')
                    {

                        if (!(savePoint[xcenter + disto, ycenter + disto]))
                        {
```

```
                        pathline p1 = new pathline();
                        p1.x = robx; p1.y = roby;
                        noofpathpoints = noofpathpoints + 1;
                       arPathPoints.Add(p1);
                    }

                    if ((robx >= xr) && (roby >= yl))
                    {
                        theta = 1.57;
                        newtheta = 0;
                        ch1 = 'l';
                    }
                }



                if (ch1 == 'c')
                {
                    if (!(savePoint[xcenter + disto, ycenter + disto]))
                    {
                        pathline p = new pathline();
                        p.x = robx; p.y = roby;
                       noofpathpoints = noofpathpoints + 1;
                       arPathPoints.Add(p);
                    }

                    if ((robx >= xr) && (roby >= yl))
                    {
                   theta = Math.Atan2((goalY - roby), (goalX - robx));
                        newtheta = 0;
                        ch1 = 'k';
                    }

                    if ((robx >= xl) && (roby >= yr))
                    {
            theta = Math.Atan2((goalY - roby), (goalX - robx)); ;
                        newtheta = 0;
                        ch1 = 'k';
                    }
                }


                if (Math.Abs(goalX - robx) < distg && Math.Abs(goalY -
roby) < distg)
                {
                    newtheta = 1;
                    break;
                }
            }

            pathline p2 = new pathline();
            if (!(savePoint[xcenter, ycenter]))
            {
                robx = xcenterr; roby = ycenterr;
                p2.x = robx; p2.y = roby;
                noofpathpoints = noofpathpoints + 1;
arPathPoints.Add(p2);
            }

            if (newtheta == 0)
            {
```

```
                    //Check for new Obstacle
                    xr = xcenter; yr = ycenter; xl = xcenter; yl = ycenter;
x = xcenter; y = ycenter;

                    if (savePoint[x, y]) //&& (savePoint[xr + 1, yr + 1]))
                    {

robx = robx - disto; roby = roby - disto; xfix = x; yfix = y;
if (!(savePoint[x - 1, y]) && (savePoint[x + 1, y])) { ch1 = 'l'; }
else if (!(savePoint[x, y - 1]) && (savePoint[x, y + 1])) { ch1 = 'u'; }
else if ((savePoint[x - 1, y]) && (savePoint[x, y - 1])) { ch1 = 'c'; }
                    // if (!(savePoint[x - 1, y]) && (!(savePoint[x, y
- 1]))) { ch1 = 'c'; }

                        while (savePoint[x, y])
                        {
                            if ((savePoint[x, y]) && !(savePoint[x, y +
1]))
                            { yr = y; }
                            y = y + 1;
                        }
                        x = xfix; y = yfix;
                        while (savePoint[x, y])
                        {
                            if ((savePoint[x, y]) && !(savePoint[x, y -
1]))
                            { yl = y; }
                            y = y - 1;
                        }
                        x = xfix; y = yfix;
                        while (savePoint[x, y])
                        {
                            if ((savePoint[x, y]) && !(savePoint[x + 1,
y]))
                            { xr = x; }
                            x = x + 1;
                        }

                        x = xfix; y = yfix;
                        while (savePoint[x, y])
                        {
                            if ((savePoint[x, y]) && !(savePoint[x - 1,
y]))
                            { xl = x; }
                            x = x - 1;
                        }
                        int xc = xl, yc = yl; x = xl; y = yl;
                        xc = (xl + xr) / 2; yc = (yl + yr) / 2; x = xc; y =
yc;

                        while (savePoint[xc, y])
                        {
                            if (savePoint[xc, y] && (!savePoint[xc, y +
1])) { yr = y; }
                            y = y + 1;
                        } x = xc; y = yc;

                        while (savePoint[xc, y])
                        {
      if (savePoint[xc, y] && (!savePoint[xc, y - 1])) { yl = y; }
                            y = y - 1;
```

```
                        } x = xc; y = yc;

                        while (savePoint[x, yc])
                        {
      if (savePoint[x, yc] && (!savePoint[x + 1, yc])) { xr = x; }
                            x = x + 1;
                        } x = xc; y = yc;

                        while (savePoint[x, yc])
                        {
                            if (savePoint[x, yc] && (!savePoint[x - 1,
yc])) { xl = x; }
                            x = x - 1;
                        }

                        int xrs = xr, xls = xl, yls = yl, yrs = yr
    xls = xl - safedobst; xrs = xr + safedobst; yls = yl - safedobst; yrs =
yr + safedobst;


                        xr = xrs; xl = xls; yr = yrs; yl = yls; xxx =
xcenter; yyy = xrs;
                         c = Math.Sqrt(Math.Pow((robx - xcenterr), 2) +
Math.Pow((roby - ycenterr), 2));
                    if (ch1 == 'c')
                    {
                        ar = Math.Sqrt(Math.Pow((xcenterr - xr), 2) +
Math.Pow((ycenterr - yl), 2));
                        br = Math.Sqrt(Math.Pow((robx - xr), 2) +
Math.Pow((roby - yl), 2));
                        al = Math.Sqrt(Math.Pow((xcenterr - xl), 2) +
Math.Pow((ycenterr - yr), 2));
                        bl = Math.Sqrt(Math.Pow((robx - xl), 2) +
Math.Pow((roby - yr), 2));
                        thetaleft = Math.Acos((Math.Pow(br, 2) +
Math.Pow(c, 2) - Math.Pow(ar, 2)) / (2 * br * c));
                        thetaright = Math.Acos((Math.Pow(bl, 2) +
Math.Pow(c, 2) - Math.Pow(al, 2)) / (2 * bl * c));
                    }
                    if (ch1 == 'u')
                    {
                        al = Math.Sqrt(Math.Pow((xcenterr - xr), 2) +
Math.Pow((ycenterr - yl), 2));
                        bl = Math.Sqrt(Math.Pow((robx - xr), 2) +
Math.Pow((roby - yl), 2));
                        ar = Math.Sqrt(Math.Pow((xcenterr - xl), 2) +
Math.Pow((ycenterr - yl), 2));
                        br = Math.Sqrt(Math.Pow((robx - xl), 2) +
Math.Pow((roby - yl), 2));
                        thetaleft = Math.Acos((Math.Pow(bl, 2) +
Math.Pow(c, 2) - Math.Pow(al, 2)) / (2 * bl * c));
                        thetaright = Math.Acos((Math.Pow(br, 2) +
Math.Pow(c, 2) - Math.Pow(ar, 2)) / (2 * br * c));
                    }
                    if (ch1 == 'l')
                    {
                        ar = Math.Sqrt(Math.Pow((xcenterr - xl), 2) +
Math.Pow((ycenterr - yr), 2));
                        br = Math.Sqrt(Math.Pow((robx - xl), 2) +
Math.Pow((roby - yr), 2));
```

```
                        al = Math.Sqrt(Math.Pow((xcenterr - xl), 2) +
Math.Pow((ycenterr - yl), 2));
                        bl = Math.Sqrt(Math.Pow((robx - xl), 2) +
Math.Pow((roby - yl), 2));
                        thetaleft = Math.Acos((Math.Pow(bl, 2) +
Math.Pow(c, 2) - Math.Pow(al, 2)) / (2 * bl * c));
                        thetaright = Math.Acos((Math.Pow(br, 2) +
Math.Pow(c, 2) - Math.Pow(ar, 2)) / (2 * br * c));
                    }
                    theta = resizeAngle2(theta);
                    thetaleft = resizeAngle2(thetaleft);
                    thetaright = resizeAngle2(thetaright);
                    //thetanew = theta;  //thetanew =
resizeAngle2(thetanew);
                    // thetaleft = thetaleft; thetaright = thetaright +
safed;

   if ((thetaleft < 0) && (thetaright < 0))
                        thetanew = theta;

   if ((thetaleft < 0) && ((thetaright >= 0) && (thetaright <= 10)))
                        thetanew = theta;

   if ((thetaleft < 0) && ((thetaright > 10) && (thetaright <= 30)))
                        thetanew = theta;

   if ((thetaleft < 0) && ((thetaright > 30) && (thetaright <= 60)))
                        thetanew = theta;

   if ((thetaleft < 0) && ((thetaright > 60) && (thetaright <= 90)))
                        thetanew = theta;

   if ((thetaleft < 0) && (thetaright > 90))
                        thetanew = theta;

                    //BLOWUP

if (((thetaleft >= 0) && (thetaleft <= 10)) && ((thetaright >= 0) &&
(thetaright <= 10)))
                    { thetanew = theta - thetaleft; llflag = 1; }

if (((thetaleft >= 0) && (thetaleft <= 10)) && ((thetaright > 10) &&
(thetaright <= 30)))
                    { thetanew = theta - thetaleft; llflag = 0; }

if (((thetaleft >= 0) && (thetaleft <= 10)) && ((thetaright > 30) &&
(thetaright <= 60)))
                    { thetanew = theta - thetaleft; llflag = 0; }

if (((thetaleft >= 0) && (thetaleft <= 10)) && ((thetaright > 60) &&
(thetaright <= 90)))
                    { thetanew = theta - thetaleft; llflag = 0; }

if (((thetaleft >= 0) && (thetaleft <= 10)) && (thetaright > 90))
                    { thetanew = theta - thetaleft; llflag = 0; }


if (((thetaleft > 10) && (thetaleft <= 30)) && ((thetaright >= 0) &&
(thetaright <= 10)))
                    { thetanew = theta + thetaright; llflag = 1; }
```

```
 if (((thetaleft > 10) && (thetaleft <= 30)) && ((thetaright > 10) &&
(thetaright <= 30)))
                    { thetanew = theta - thetaleft; llflag = 1; }

 if (((thetaleft > 10) && (thetaleft <= 30)) && ((thetaright > 30) &&
(thetaright <= 60)))
                    { thetanew = theta - thetaleft; llflag = 0; }

 if (((thetaleft > 10) && (thetaleft <= 30)) && ((thetaright > 60) &&
(thetaright <= 90)))
                    { thetanew = theta - thetaleft; llflag = 0; }

if (((thetaleft > 10) && (thetaleft <= 30)) && (thetaright > 90))
                    { thetanew = theta - thetaleft; llflag = 0; }



if (((thetaleft > 30) && (thetaleft <= 60)) && (thetaright < 0))
                       thetanew = theta;

if (((thetaleft > 30) && (thetaleft <= 60)) && ((thetaright >= 0) &&
(thetaright <= 10)))
                    { thetanew = theta + thetaright; llflag = 1; }

if (((thetaleft > 30) && (thetaleft <= 60)) && ((thetaright > 10) &&
(thetaright <= 30)))
                    { thetanew = theta + thetaright; llflag = 1; }

if (((thetaleft > 30) && (thetaleft <= 60)) && ((thetaright > 30) &&
(thetaright <= 60)))
                    { thetanew = theta - thetaleft; llflag = 1; }

if (((thetaleft > 30) && (thetaleft <= 60)) && ((thetaright > 60) &&
(thetaright <= 90)))
                    { thetanew = theta - thetaleft; llflag = 0; }

if (((thetaleft > 30) && (thetaleft <= 60)) && (thetaright > 90))
                    { thetanew = theta - thetaleft; llflag = 0; }


if (((thetaleft > 60) && (thetaleft <= 90)) && ((thetaright >= 0) &&
(thetaright <= 10)))
                    { thetanew = theta + thetaright; llflag = 1; }

if (((thetaleft > 60) && (thetaleft <= 90)) && ((thetaright > 10) &&
(thetaright <= 30)))
                    { thetanew = theta + thetaright; llflag = 1;

if (((thetaleft > 60) && (thetaleft <= 90)) && ((thetaright > 30) &&
(thetaright <= 60)))
                    { thetanew = theta + thetaright; llflag = 1; }

if (((thetaleft > 60) && (thetaleft <= 90)) && ((thetaright > 60) &&
(thetaright <= 90)))
                    { thetanew = theta - thetaleft; llflag = 1; }

if (((thetaleft > 60) && (thetaleft <= 90)) && (thetaright > 90))
                    { thetanew = theta - thetaleft; llflag = 1; }
```

```
if ((thetaleft > 90) && ((thetaright >= 0) && (thetaright <= 10)))
     { thetanew = theta + thetaright; llflag = 1; }

if (((thetaleft > 90) && (thetaright > 10)) && (thetaright <= 30))
                    { thetanew = theta + thetaright; llflag = 1; }

if (((thetaleft > 90) && (thetaright > 30)) && (thetaright <= 60))
                    { thetanew = theta + thetaright; llflag = 1; }

if (((thetaleft > 90) && (thetaright > 60)) && (thetaright <= 90))
                    { thetanew = theta + thetaright; llflag = 1; }

if ((thetaleft > 90) && (thetaright > 90))
                        thetanew = theta - thetaleft;
                    //thetanew=finference(thetaleft, thetaright);
                    thetanew = thetanew + safed;
                    theta = resizeAngle(thetanew);
                    newtheta = 1;
            }
       }
 }
```